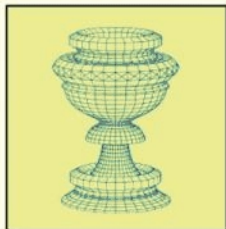
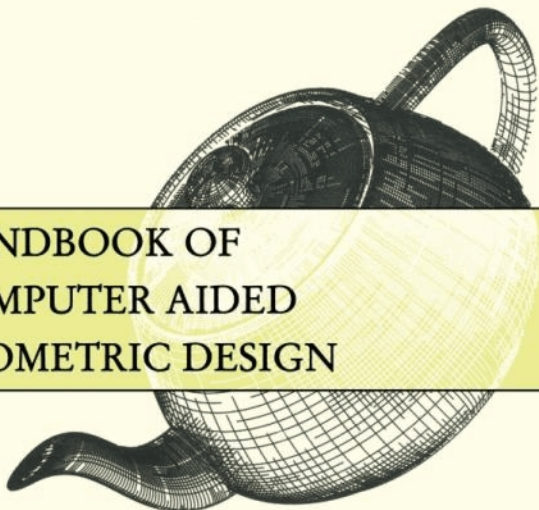
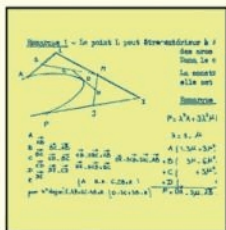


HANDBOOK OF COMPUTER AIDED GEOMETRIC DESIGN



Edited by
G. Farin, J. Hoschek
and M.-S. Kim



NORTH-HOLLAND

ELSEVIER SCIENCE B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

© 2002 Elsevier Science B.V. All rights reserved.

This work is protected under copyright by Elsevier Science, and the following terms and conditions apply to its use:

Photocopying

Single photocopies of single chapters may be made for personal use as allowed by national copyright laws. Permission of the Publisher and payment of a fee is required for all other photocopying, including multiple or systematic copying, copying for advertising or promotional purposes, resale, and all forms of document delivery. Special rates are available for educational institutions that wish to make photocopies for non-profit educational classroom use.

Permissions may be sought directly from Elsevier Science via their homepage (<http://www.elsevier.com>) by selecting 'Customer support' and then 'Permissions'. Alternatively you can send an e-mail to: permissions@elsevier.co.uk, or fax to: (+44) 1865 853333.

In the USA, users may clear permissions and make payments through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA; phone: (+1) (978) 7508400, fax: (+1) (978) 7504744, and in the UK through the Copyright Licensing Agency Rapid Clearance Service (CLARCS), 90 Tottenham Court Road, London W1P 0LP, UK; phone: (+44) 207 631 5555; fax: (+44) 207 631 5500. Other countries may have a local reprographic rights agency for payments.

Derivative Works

Tables of contents may be reproduced for internal circulation, but permission of Elsevier Science is required for external resale or distribution of such material.

Permission of the Publisher is required for all other derivative works, including compilations and translations.

Electronic Storage or Usage

Permission of the Publisher is required to store or use electronically any material contained in this work, including any chapter or part of a chapter.

Except as outlined above, no part of this work may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without prior written permission of the Publisher.

Address permissions requests to: Elsevier Science Global Rights Department, at the mail, fax and e-mail addresses noted above.

Notice

No responsibility is assumed by the Publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the material herein. Because of rapid advances in the medical sciences, in particular, independent verification of diagnoses and drug dosages should be made.

First edition 2002

Library of Congress Cataloging in Publication Data

A catalog record from the Library of Congress has been applied for.

British Library Cataloguing in Publication Data

A catalogue record from the British Library has been applied for.

ISBN: 0-444-51104-0

© The paper used in this publication meets the requirements of ANSI/NISO Z39.48-1992 (Permanence of Paper).
Printed in The Netherlands.

Contents

Preface	v
Contributors	xxiii
1 A History of Curves and Surfaces in CAGD (G. Farin)	1
1.1 INTRODUCTION	1
1.2 EARLY DEVELOPMENTS	2
1.3 DE CASTELJAU AND BÉZIER	4
1.4 PARAMETRIC CURVES	6
1.5 RECTANGULAR SURFACES	8
1.6 B-SPLINE CURVES AND NURBS	9
1.7 TRIANGULAR PATCHES	10
1.8 SUBDIVISION SURFACES	11
1.9 SCIENTIFIC APPLICATIONS	12
1.10 SHAPE	13
1.11 INFLUENCES AND APPLICATIONS	14
2 Geometric Fundamentals (W. Boehm and H. Prautzsch)	23
2.1 AFFINE FUNDAMENTALS	23
2.1.1 Points and vectors	23
2.1.2 Affine systems	23
2.1.3 Barycentric coordinates	24
2.1.4 Affine subspaces and parallelism	24
2.1.5 Affine maps and axonometric images	25
2.1.6 Affine combinations and A-frame	26
2.2 CONIC SECTIONS AND QUADRICS	27
2.2.1 Quadrics in affine space	27
2.2.2 Tangents and polar planes	28
2.2.3 Pascal's and Brianchon's theorems	29
2.3 THE EUCLIDEAN SPACE	30
2.3.1 Cartesian coordinates	30
2.3.2 Gram-Schmidt orthogonalization	30
2.3.3 Euclidean motions and orthogonal projections	31
2.3.4 Quadrics in Euclidean space	32
2.4 PROJECTIVE FUNDAMENTALS	32

2.4.1	Homogeneous coordinates	32
2.4.2	Projective coordinates	33
2.4.3	Projective maps	33
2.4.4	The procedure of inhomogenizing	34
2.4.5	Repeated projective combinations	34
2.4.6	Quadrics in projective space	34
2.4.7	Parametrizing a quadric and its equation	35
2.5	DUALITY	35
2.6	OSCULATING CURVES AND SURFACES	36
2.6.1	Curve and surface	36
2.6.2	Curve and curve	37
2.6.3	Surface and surface	37
2.6.4	Contour lines, reflection lines and isophotes	37
2.7	DIFFERENTIAL FUNDAMENTALS	38
2.7.1	Arc length and osculating plane	38
2.7.2	Curvature and torsion	39
2.7.3	The Frenet frame	39
2.7.4	Curves on surfaces	39
2.7.5	Meusnier's sphere and Dupin's indicatrix	40
2.7.6	The curvatures of a surface	41
3	Geometries for CAGD (<i>H. Pottmann and S. Leopoldseder</i>)	43
3.1	CURVES AND SURFACES IN PROJECTIVE GEOMETRY	44
3.1.1	Bézier curves and surfaces as images of normal curves and surfaces	44
3.1.2	NURBS curves and surfaces in projective geometry	46
3.1.3	Duality and dual representation	47
3.1.4	Developable surfaces as dual curves	49
3.2	SPHERE GEOMETRIES	50
3.2.1	Models of Laguerre geometry	50
3.2.2	Möbius geometry	53
3.2.3	Applications of the cyclographic image of a curve in 3-space	54
3.2.4	The medial axis transform in a sphere geometric approach	55
3.2.5	Canal surfaces (in Laguerre and Möbius geometry)	56
3.2.6	Rational curves and surfaces with rational offsets	58
3.3	LINE GEOMETRY	60
3.3.1	Basics of line geometry	60
3.3.2	Linear complexes in kinematics and reverse engineering	61
3.3.3	Ruled surfaces	62
3.3.4	Other applications of line geometry in geometric computing	63
3.4	APPROXIMATION IN SPACES OF GEOMETRIC OBJECTS	63
3.4.1	Approximation in the space of spheres	64
3.4.2	Approximation in the space of planes	64
3.4.3	Approximation in line space	64
3.5	NON-EUCLIDEAN GEOMETRIES	65
3.5.1	Hyperbolic geometry and geometric topology	65

3.5.2	Elliptic geometry and kinematics	66
3.5.3	Isotropic geometry and analysis of functions and images	66
4	Bézier Techniques (<i>D. Hansford</i>)	75
4.1	WHY BÉZIER TECHNIQUES?	75
4.2	BÉZIER CURVES	76
4.2.1	Parametric curves	76
4.2.2	Properties of Bézier curves	77
4.2.3	The de Casteljau algorithm for Bézier curves	80
4.2.4	Bernstein polynomials	81
4.2.5	Derivatives of Bézier curves	83
4.2.6	Degree elevation of Bézier curves	85
4.2.7	Interrogation techniques for Bézier curves	85
4.2.8	Basis conversion	86
4.2.9	Piecewise Bézier curves	87
4.3	RECTANGULAR BÉZIER PATCHES	88
4.3.1	Bilinear patches	89
4.3.2	Bézier patches	90
4.3.3	Properties of Bézier patches	91
4.3.4	Evaluation of Bézier patches	93
4.3.5	Derivatives of Bézier patches	93
4.3.6	Working with Bézier patches	95
4.3.7	C^1 Bézier patches	96
4.4	TRIANGULAR BÉZIER PATCHES	97
4.4.1	Bézier triangles introduced	97
4.4.2	Properties of Bézier triangles	98
4.4.3	The de Casteljau algorithm for Bézier triangles	100
4.4.4	Bivariate Bernstein polynomials	100
4.4.5	Derivatives of Bézier triangles	102
4.4.6	Working with Bézier triangles	103
4.4.7	C^1 Bézier triangles	104
5	Rational Techniques (<i>H.J. Wolters</i>)	111
5.1	INTRODUCTION	111
5.2	RATIONAL BÉZIER CURVES	112
5.2.1	Basic definitions	112
5.2.2	Derivatives	114
5.2.3	Fundamental algorithms	115
5.2.4	Conics	117
5.3	RATIONAL B-SPLINE CURVES	118
5.3.1	Basic definitions	119
5.3.2	Derivatives	120
5.3.3	Fundamental algorithms	121
5.4	GEOMETRIC CONTINUITY FOR RATIONAL CURVES	122
5.5	RATIONAL CURVE APPROXIMATION AND INTERPOLATION	123

5.5.1	Rational curve interpolation	124
5.5.2	Rational curve approximation	126
5.6	RATIONAL BÉZIER SURFACES	127
5.6.1	Basic definitions	127
5.6.2	Derivatives	129
5.6.3	Algorithms	130
5.7	RATIONAL B-SPLINE SURFACES	131
5.7.1	Basic definitions	131
5.7.2	Derivatives	131
5.7.3	Algorithms	133
5.8	GEOMETRIC CONTINUITY FOR RATIONAL PATCHES	133
5.9	INTERPOLATION AND APPROXIMATION ALGORITHMS	134
5.10	RATIONAL SURFACE CONSTRUCTIONS	136
5.10.1	Surfaces of revolution	136
5.10.2	Canal and pipe surfaces	136
5.11	CONCLUDING REMARKS	137
6	Spline Basics (<i>C. de Boor</i>)	141
6.1	PIECEWISE POLYNOMIALS	141
6.2	B-SPLINES DEFINED	141
6.3	SUPPORT AND POSITIVITY	143
6.4	SPLINE SPACES DEFINED	144
6.5	SPECIFIC KNOT SEQUENCES	145
6.6	THE POLYNOMIALS IN THE SPLINE SPACE: MARSDEN'S IDENTITY	147
6.7	THE PIECEWISE POLYNOMIALS IN THE SPLINE SPACE	148
6.8	DUAL FUNCTIONALS AND BLOSSOMS	151
6.9	GOOD CONDITION	152
6.10	CONVEX HULL	152
6.11	DIFFERENTIATION AND INTEGRATION	153
6.12	EVALUATION	154
6.13	SPLINE FUNCTIONS VS SPLINE CURVES	155
6.14	KNOT INSERTION	156
6.15	VARIATION DIMINUTION AND SHAPE PRESERVATION: SCHOENBERG'S OPERATOR	158
6.16	ZEROS OF A SPLINE, COUNTING MULTIPLICITY	159
6.17	SPLINE INTERPOLATION: SCHOENBERG-WHITNEY	159
6.18	SMOOTHING SPLINE	161
6.19	LEAST-SQUARES SPLINE APPROXIMATION	161
6.20	BACKGROUND	162
7	Curve and Surface Constructions (<i>D. Hansford and G. Farin</i>)	165
7.1	INTRODUCTION	165
7.2	POLYNOMIAL CURVE METHODS	165
7.2.1	Point Data Interpolation	166
7.2.2	Point Data Approximation	170

7.2.3	Point and Tangent Data Interpolation	173
7.3	C^2 CUBIC SPLINE INTERPOLATION	175
7.3.1	End Conditions	177
7.3.2	Defining a Knot Sequence	178
7.3.3	The Minimum Property	179
7.4	POLYNOMIAL SURFACE METHODS	179
7.4.1	Discrete Coons Patches	179
7.4.2	Tensor Product Interpolation	181
7.4.3	Approximation with Tensor Product Patches	182
7.4.4	Bicubic Hermite Patches	185
7.5	C^2 BICUBIC SPLINE INTERPOLATION	186
7.5.1	Finding Knot Sequences	187
7.6	VOLUME DEFORMATIONS	188
8	Geometric Continuity (<i>J. Peters</i>)	193
8.1	MOTIVATING EXAMPLES	193
8.1.1	Differentiation and evaluation	197
8.2	GEOMETRIC CONTINUITY OF PARAMETRIC CURVES/SURFACES	198
8.2.1	Joining parametric curve pieces	199
8.2.2	Geometric continuity of edge-adjacent patches	200
8.2.3	Geometric continuity at a vertex	202
8.2.4	Free-form surface splines	209
8.3	EQUIVALENT AND ALTERNATIVE DEFINITIONS	210
8.3.1	Matching intrinsic curve properties	210
8.3.2	C^k manifolds	212
8.3.3	Tangent and normal continuity	212
8.3.4	Global and regional reparametrization	214
8.3.5	Implicit representation	217
8.3.6	Generalized subdivision	217
8.4	CONSTRUCTIONS	218
8.4.1	Free-form surface splines of low degree	220
8.5	ADDITIONAL LITERATURE	220
9	Splines on Surfaces (<i>M. Neamtu</i>)	229
9.1	INTRODUCTION	229
9.2	SCALAR SPLINES ON SMOOTH SURFACES	236
9.3	ALTERNATIVE METHODS FOR FUNCTIONS ON SURFACES	246
9.3.1	Discrete surfaces	246
9.3.2	Radial basis functions	246
9.3.3	Variational methods	247
9.3.4	Distance-weighting methods	247
9.3.5	Transfinite methods	247
9.3.6	Implicit methods	248
9.3.7	Other types of splines	248
9.3.8	Multiresolution methods	248

9.3.9	Visualization of surfaces on surfaces	248
10	Box Splines (<i>H. Prautzsch and W. Boehm</i>)	255
10.1	BOX SPLINES	255
10.1.1	Inductive definition	255
10.1.2	Geometric definition	256
10.1.3	Further definitions of Box splines	257
10.1.4	Basic properties of Box splines	258
10.1.5	Derivatives	259
10.2	BOX SPLINE SURFACES	260
10.2.1	Translates of Box splines	260
10.2.2	Derivatives and polynomial properties	262
10.2.3	Convexity	262
10.2.4	Subdivision	263
10.2.5	General subdivision	264
10.2.6	Convergence under subdivision	265
10.2.7	Bézier representation	266
10.2.8	Bézier representation of symmetric Box splines	268
10.2.9	Generalized Box spline surfaces	269
10.3	HALF-BOX SPLINES	272
10.3.1	Inductive definition	272
10.3.2	Basic properties	273
10.3.3	Derivatives and polynomial structure	274
10.4	HALF-BOX SPLINE SURFACES	275
10.4.1	Translates of Half-Box splines	275
10.4.2	Derivatives and polynomial properties	276
10.4.3	Subdivision	276
10.4.4	Bézier representation	278
10.4.5	Generalized Half-Box spline surfaces	278
11	Finite Element Approximation with Splines (<i>K. Höllig</i>)	283
11.1	INTRODUCTION	283
11.2	SPLINES ON UNIFORM GRIDS	285
11.2.1	Uniform B-splines	285
11.2.2	Splines on bounded domains	286
11.2.3	Hierarchical bases	287
11.3	FINITE ELEMENT BASES	289
11.3.1	Mesh-based elements	289
11.3.2	WEB-basis	290
11.3.3	R-functions	293
11.3.4	Stability	295
11.4	APPROXIMATION OF BOUNDARY VALUE PROBLEMS	296
11.4.1	Essential boundary conditions	296
11.4.2	Natural boundary conditions	298
11.4.3	Mixed and higher order boundary conditions	299

11.4.4	Error estimates	301
11.4.5	Implementation	302
11.5	SUMMARY	304
12	Subdivision Surfaces (<i>M. Sabin</i>)	309
12.1	SUBDIVISION SURFACE DEFINITIONS	309
12.2	INTRODUCTION – SUBDIVISION CURVES	309
12.2.1	The Chaikin construction	309
12.2.2	Higher degree splines	310
12.2.3	The 4-point interpolatory scheme	312
12.3	BOX-SPLINES	312
12.4	GENERALIZATIONS TO ARBITRARY TOPOLOGY	313
12.5	SOME SPECIFIC SCHEMES	313
12.5.1	The Doo-Sabin quadratic scheme	313
12.5.2	The Catmull-Clark cubic scheme	314
12.5.3	The Loop triangular mesh scheme	314
12.5.4	The Butterfly interpolatory scheme	314
12.6	ANALYSIS OF CONTINUITY AT THE SINGULARITIES	315
12.6.1	Support	315
12.6.2	Regular regions	315
12.6.3	Neighbourhoods of singularities	316
12.6.4	Limit point	317
12.6.5	Natural configuration and characteristic map	318
12.6.6	Curvatures	318
12.6.7	Tuning subdivisions for better behaviour	319
12.6.8	Jordan blocks	319
12.6.9	Discontinuities of curvature	320
12.6.10	Higher derivatives	320
12.6.11	Precision set	320
12.7	FIRST STEP ARTIFACTS	320
12.8	CURRENT RESEARCH DIRECTIONS	321
12.8.1	Square root of 3 scheme	321
12.8.2	Subdivision over semiregular lattices	321
12.8.3	Dual schemes and contact elements	321
12.8.4	Unequal intervals	322
12.8.5	Artifact analysis	323
12.9	CONCLUSIONS	323
13	Interrogation of Subdivision Surfaces (<i>M. Sabin</i>)	327
13.1	SUBDIVISION SURFACE INTERROGATIONS	327
13.2	HISTORICAL BACKGROUND	328
13.3	THE CONVEX HULL PROPERTY	329
13.3.1	Other hulls	329
13.3.2	Hulls of non-positive bases	331
13.3.3	Normal hulls	331

13.3.4	Offset subdivision surfaces	332
13.4	AN API FOR SUBDIVISION SURFACES	332
13.5	EXAMPLE INTERROGATIONS	333
13.5.1	Z-buffer imaging	333
13.5.2	Raycast	333
13.5.3	Plane section	335
13.5.4	Surface-surface intersection	336
13.5.5	Silhouette	337
13.6	PERFORMANCE ISSUES	338
13.7	CONCLUSIONS	340
14	Multiresolution Techniques (L.P. Kobbelt)	343
14.1	INTRODUCTION	343
14.2	MULTIRESOLUTION REPRESENTATIONS FOR CURVES	344
14.3	LIFTING	347
14.4	GEOMETRIC SETTING	349
14.5	MULTIRESOLUTION REPRESENTATIONS FOR SURFACES	350
14.5.1	Coarse-to-fine hierarchies	350
14.5.2	Fine-to-coarse hierarchies	353
14.6	APPLICATIONS	357
14.6.1	Multiresolution editing	357
14.6.2	Geometry compression	358
15	Algebraic Methods for Computer Aided Geometric Design	
	<i>(T.W. Sederberg and J. Zheng)</i>	363
15.1	INTRODUCTION	363
15.2	POLYNOMIALS, IDEALS, AND VARIETIES	364
15.2.1	Notation and terminology	364
15.2.2	Ideals and varieties	366
15.2.3	Gröbner bases	366
15.3	RESULTANTS	367
15.3.1	Sylvester's resultant	367
15.3.2	Bezout's resultant	368
15.3.3	Dixon's resultant	369
15.4	CURVE IMPLICITIZATION AND INVERSION	370
15.4.1	Resultant-based method	370
15.4.2	Gröbner basis technique	371
15.4.3	Moving curve technique	372
15.5	CURVE PARAMETRIZATION	373
15.5.1	Planar algebraic curves	373
15.5.2	Genus and rationality	374
15.5.3	Parametrizing curves	374
15.6	INTERSECTION COMPUTATIONS	377
15.6.1	Parametric curve and implicit curve	377
15.6.2	Implicit curve and implicit curve	378

15.6.3 Parametric curve and parametric curve	379
15.7 SURFACES	380
15.7.1 Implicit degree of a rational parametric surface	381
15.7.2 Surface intersection curves	382
15.7.3 Implicitization	382
15.7.4 Parametrizaion	383
15.8 OTHER ISSUES	384
16 Scattered Data Interpolation: Radial Basis and Other Methods (<i>S.K. Lodha and R. Franke</i>)	389
16.1 INTRODUCTION	389
16.2 RADIAL INTERPOLATION	390
16.2.1 Existence and uniqueness	392
16.2.2 Computation of the interpolant	393
16.2.3 Evaluation	396
16.2.4 Applications	397
16.3 OTHER LOCAL METHODS	399
16.4 CONCLUSIONS	401
17 Pythagorean-Hodograph Curves (<i>R.T. Farouki</i>)	405
17.1 PREAMBLE	405
17.2 POLYNOMIAL PH CURVES	406
17.2.1 Planar PH curves	407
17.2.2 Complex representation	408
17.2.3 PH space curves	410
17.3 CONSTRUCTION ALGORITHMS	412
17.3.1 PH quintic Hermite interpolants	412
17.3.2 Shape properties of PH quintics	414
17.3.3 C^2 PH quintic splines	414
17.3.4 Spatial PH quintic Hermite interpolants	416
17.3.5 Geometric Hermite interpolants	417
17.3.6 Further constructions	417
17.4 REAL-TIME CNC INTERPOLATORS	417
17.5 RATIONAL CURVES WITH RATIONAL OFFSETS	419
17.5.1 Rational PH curves	419
17.5.2 Improper parameterizations	421
17.6 MINKOWSKI PH CURVES	421
17.6.1 Minkowski metric of special relativity	421
17.6.2 Minkowski metric defined by convex indicatrix	422
17.7 CLOSURE	423
18 Voronoi Diagrams (<i>K. Sugihara</i>)	429
18.1 ORDINARY VORONOI DIAGRAM	429
18.2 DELAUNAY DIAGRAM	431
18.3 BASIC PROPERTIES OF THE VORONOI AND DELAUNAY DIAGRAMS	432

18.4	ALGORITHMS	433
18.5	APPLICATIONS	435
	18.5.1 Site retrieval	435
	18.5.2 Medial axis	435
	18.5.3 Offset curves and surfaces	436
	18.5.4 Interpolation	437
18.6	EXTENSIONS	441
	18.6.1 Voronoi diagrams for general distances	441
	18.6.2 Additively weighted Voronoi diagram	441
	18.6.3 Multiplicatively weighted Voronoi diagram	441
	18.6.4 Power diagram	442
	18.6.5 Voronoi diagram based on L_p distance	444
	18.6.6 Voronoi diagram based on elliptic distance	444
	18.6.7 Obstacle-avoidance Voronoi diagram	445
	18.6.8 Voronoi diagram in a river	446
	18.6.9 Crystal Voronoi diagram	446
	18.6.10 Voronoi diagram for lines and polygons	446
	18.6.11 Voronoi diagram for general figures	448
18.7	CONCLUSION	448
19	The Medial Axis Transform (<i>H.I. Choi and C.Y. Han</i>)	451
19.1	INTRODUCTION	451
19.2	MATHEMATICAL THEORY OF THE MEDIAL AXIS TRANSFORM	453
	19.2.1 Assumptions on the domain	453
	19.2.2 Medial axis transform	455
	19.2.3 Finiteness results	457
	19.2.4 Graph structure of medial axis transform	458
	19.2.5 Domain decomposition lemma	458
19.3	ALGORITHMS	460
	19.3.1 Piecewise linear and circular arc boundary	460
	19.3.2 Domains with free-form boundaries	461
	19.3.3 Global decomposition algorithm	462
19.4	CONCLUDING REMARKS	464
20	Solid Modeling (<i>V. Shapiro</i>)	473
20.1	INTRODUCTION	473
	20.1.1 A premise of informational completeness	473
	20.1.2 Outline	474
20.2	MATHEMATICAL MODELS	475
	20.2.1 First postulates	475
	20.2.2 Continuum point set model of solidity	476
	20.2.3 Combinatorial model of solidity	477
	20.2.4 Generalizations	480
20.3	COMPUTER REPRESENTATIONS	480
	20.3.1 Implicit and constructive	481

20.3.2	Enumerative and combinatorial	485
20.3.3	Boundary representation: a compromise	488
20.3.4	Unification of representation schemes	490
20.4	ALGORITHMS	491
20.4.1	Fundamental computations	491
20.4.2	Enabling algorithms	494
20.5	APPLICATIONS	497
20.5.1	Geometric design	498
20.5.2	Analysis and simulation	498
20.5.3	Dynamic analysis and lumped-parameter systems	500
20.5.4	Planning and generation	500
20.5.5	Manufacturing	501
20.6	SYSTEMS	502
20.6.1	Classical systems	502
20.6.2	Parametric interaction	503
20.6.3	Standards and interfaces	505
20.7	CONCLUSIONS	506
20.7.1	Unsolved problems and promising directions	506
20.7.2	Summary	510
21	Parametric Modeling (C.M. Hoffmann and R. Joan-Arinyo)	519
21.1	INTRODUCTION	519
21.2	PARAMETRIC MODELS	520
21.3	VARIANT MODELING	521
21.4	CONSTRAINT-BASED MODELING	522
21.4.1	Constraints	522
21.4.2	Modeling with constraints	522
21.4.3	Solving geometric and equational constraints	523
21.4.4	Degrees of freedom analysis	526
21.5	FEATURE-BASED MODELING	527
21.5.1	Features and the feature model	527
21.5.2	A brief feature taxonomy	528
21.5.3	Feature model construction	528
21.5.4	Feature representation	530
21.5.5	Features and constraints	531
21.6	TRENDS	531
21.6.1	Feature libraries	532
21.6.2	Multiple views	532
21.6.3	Semantic features	533
21.6.4	Persistent naming	534
21.7	OPEN PROBLEMS	535
21.7.1	Constraint solving	535
21.7.2	Features	535
21.7.3	Semantics of parametric design	535
21.7.4	Assembly-centric design	536

22 Sculptured Surface NC Machining	543
<i>(B.K. Choi, B.H. Kim, and R.B. Jerard)</i>	
22.1 INTRODUCTION	543
22.1.1 Overview of the sculptured surface machining process	543
22.1.2 Information processing issues	545
22.2 UNIT MACHINING OPERATIONS	547
22.2.1 Tool path topology and milling-strategy options	548
22.2.2 Ball-endmill UMOs	548
22.2.3 Flat-endmill UMOs	549
22.3 INTERFERENCE HANDLING	550
22.3.1 CL-point interference	551
22.3.2 CL-line interference	551
22.3.3 Collisions	552
22.4 TOOL PATH GENERATION METHODS	552
22.4.1 The conventional approach and the C-space approach	552
22.4.2 Geometric issues in conventional approach	555
22.4.3 Geometric issues in the C-space approach	557
22.5 GEOMETRIC ALGORITHMS	558
22.5.1 CL-surface construction	558
22.5.2 2D PS-curve offsetting	561
22.5.3 Area scan algorithm	564
22.5.4 Point-sequence curve fairing	567
22.5.5 Collision detection algorithms	572
22.6 CONCLUSION	572
23 Cyclides (<i>W. Degen</i>)	575
23.1 INTRODUCTION	575
23.2 THE GEOMETRY OF DUPIN CYCLIDES	576
23.2.1 Dupin cyclides in classical differential geometry	576
23.2.2 The three main types of Dupin cyclides and their parameter representations	578
23.2.3 Implicit equations	582
23.3 SUPERCYCLIDES	583
23.3.1 Curves and surfaces in the projective space	583
23.3.2 Basic properties of supercyclides	583
23.4 CYCLIDES IN CAGD	586
23.4.1 Bézier representation of cyclides	586
23.4.2 Using cyclides as blendings	588
23.4.3 Blending with supercyclides	591
23.5 APPENDIX: STUDYING DUPIN CYCLIDES WITH LIE GEOMETRY	593
24 Geometry Processing (<i>T.A. Grandine</i>)	603
24.1 INTRODUCTION	603
24.2 ROOT FINDING	604
24.3 INTEGRATION	613

24.4	COMPUTING MASS PROPERTIES	618
25	Intersection Problems (<i>N.M. Patrikalakis and T. Maekawa</i>)	623
25.1	INTRODUCTION	623
25.2	CLASSIFICATION OF INTERSECTION PROBLEMS	624
25.2.1	Classification by dimension	624
25.2.2	Classification by type of geometric specification	624
25.2.3	Classification by number system	625
25.3	OVERVIEW OF NONLINEAR SOLVERS	625
25.3.1	Brief review of local and global methods	625
25.3.2	IPP algorithm	627
25.4	CURVE/SURFACE INTERSECTION	632
25.4.1	RPP curve/IA surface intersection	632
25.4.2	RPP curve/RPP surface intersection	633
25.4.3	IA curve/IA surface intersection	633
25.4.4	IA curve/RPP surface intersection	634
25.5	SURFACE/SURFACE INTERSECTIONS	634
25.5.1	RPP/IA surface intersection	634
25.5.2	RPP/RPP surface intersection	639
25.5.3	IA/IA surface intersection	642
25.6	CONCLUSION	642
26	Reverse Engineering (<i>T. Varady and R. Martin</i>)	651
26.1	INTRODUCTION	651
26.2	THE BASIC PHASES OF REVERSE ENGINEERING	652
26.3	DATA CAPTURE	653
26.3.1	Laser scanners	654
26.3.2	Multiple view registration	654
26.4	TRIANGULATION AND DECIMATION	654
26.4.1	Triangulation overview	655
26.4.2	Kós's method	655
26.5	RECONSTRUCTING FREE-FORM OBJECTS	656
26.5.1	Segmentation strategies	656
26.5.2	Fitting free-form surfaces	659
26.5.3	Fitting feature surfaces	662
26.6	RECONSTRUCTING CONVENTIONAL ENGINEERING OBJECTS	663
26.6.1	Segmentation	663
26.6.2	Fitting analytic surfaces	665
26.6.3	Fitting extruded and rotational surfaces	667
26.6.4	Constrained fitting for multiple curves and surfaces	668
26.6.5	Reconstructing blend surfaces	672
26.6.6	Building solid models	673
26.6.7	Beautifying solid models	675
26.7	CONCLUSION	676

27	Vector and Tensor Field Visualization (<i>G. Scheuermann and H. Hagen</i>)	683
27.1	INTRODUCTION	683
27.2	VISUALIZATION PROCESS	684
27.3	DATA SET TYPES AND INTERPOLATION METHODS	685
27.4	DIRECT MAPPINGS TO GEOMETRIC PRIMITIVES	686
27.4.1	Point-based methods	686
27.4.2	Line-based methods	687
27.4.3	Surface and volume-based methods	690
27.5	ATTRIBUTE MAPPINGS	691
27.6	STRUCTURE AND FEATURE BASED MAPPINGS	692
27.6.1	Vector field topology	692
27.6.2	Tensor field topology	694
27.6.3	Feature detection algorithms	695
28	Splines over Triangulations (<i>F. Zeilfelder and H.-P. Seidel</i>)	701
28.1	INTRODUCTION	701
28.2	BERNSTEIN-BÉZIER TECHNIQUES	702
28.3	DIMENSION	704
28.4	FINITE AND MACRO ELEMENTS	708
28.5	INTERPOLATION	710
28.6	TRIANGULAR B-SPLINES	714
29	Kinematics and Animation (<i>B. Jüttler and M.G. Wagner</i>)	723
29.1	INTRODUCTION	723
29.2	THE KINEMATICAL MAPPING	724
29.2.1	Coordinates	724
29.2.2	Motions of a rigid body	724
29.2.3	Euler parameters	726
29.2.4	The kinematical mapping	727
29.3	QUATERNIONS	727
29.3.1	Fundamentals	727
29.3.2	Homogeneous quaternions and the kinematical mapping	728
29.3.3	Summary: homogeneous quaternion coordinates for 3D rotations	729
29.4	MOTION DESIGN USING CURVES ON S^3	729
29.4.1	Slerping	730
29.4.2	Problems of slerping	731
29.4.3	Other approaches	731
29.4.4	Motion design - desired features	731
29.5	SPHERICAL RATIONAL MOTIONS	732
29.6	SPATIAL RATIONAL MOTIONS	735
29.6.1	Construction	735
29.6.2	Special cases	737
29.6.3	Affine control structure	737
29.6.4	Some properties	739
29.6.5	Interpolation schemes and applications	741

29.6.6 Rational frames and sweeping surfaces	744
29.7 CLOSURE	745
30 Direct Rendering of Freeform Surfaces (G. Elber)	749
30.1 INTRODUCTION	749
30.2 SCAN-CONVERSION OF CURVES	751
30.2.1 Forward differencing	751
30.2.2 Adaptive forward differencing	752
30.3 SURFACE COVERAGE AND RENDERING USING CURVES	753
30.3.1 Coverage based on adaptive isoparametric curves	756
30.3.2 Rendering using adaptive isoparametric curves	757
30.4 RAY-TRACING	759
30.4.1 Bezier clipping	759
30.4.2 Ruled tracing	761
30.5 EXTENSIONS	765
30.5.1 Isometric texture mapping	765
30.5.2 Machining using adaptive isoparametric curves	770
30.5.3 Line-art rendering	770
30.6 CONCLUSION	773
31 Modeling and Processing with Quadric Surfaces (W. Wang)	777
31.1 DEFINITION AND CLASSIFICATIONS	777
31.1.1 Definition	777
31.1.2 Euclidean classification	778
31.1.3 Affine classification	779
31.1.4 Projective classification	779
31.2 PARAMETRIC REPRESENTATION	780
31.2.1 Global rational parameterization	780
31.2.2 Generalized stereographic projection	782
31.2.3 Surface patches on quadrics	784
31.3 FITTING, BLENDING, AND OFFSETTING	785
31.3.1 Fitting	785
31.3.2 Blending	786
31.3.3 Offsetting	788
31.4 INTERSECTION AND INTERFERENCE	789
31.4.1 Computation of intersection curves	789
31.4.2 Detecting interference	792
31.5 ACKNOWLEDGMENTS	792
Index	797

Preface

CAGD – short for Computer Aided Geometric Design – is the discipline concerned with the computational and geometric aspects of free-form curves, surfaces and volumes as they are used, for example in CAD/CAM, scientific visualization, or computer animation. CAGD started in the 1960s, going back to efforts by Citroën and Renault in France and by Boeing and General Motors in the U.S. Emerging from unrelated parallel developments, a coherent scientific discipline began to form in the 1970s, mainly due to the 1972 conference at the University of Utah, organized by R. Barnhill and R. Riesenfeld. About ten years later, the journal CAGD was founded by R. Barnhill and W. Boehm and published by North-Holland. Since then, the field has progressed significantly, as this handbook intends to document.

Drawing from many areas and influencing others, CAGD is inherently interdisciplinary. The earliest influences came from mechanical engineering in the form of new and puzzling problems in the emerging field of CAD/CAM. Their solutions involved results from approximation theory and differential geometry, but also from computer graphics and new software developments. Owing to these diverse roots, positioning CAGD within the science and engineering fields would be an ambitious endeavor. We think it is best to recognize the multiple origins and, from them, to expect a multitude of contributions to various scientific, engineering, mathematical, and other areas.

This handbook will thus not be able to cover every aspect of CAGD; yet it represents our best effort to provide a comprehensive collection of knowledge that has been collected to date. It contains the basics of curve and surface modeling (the very start of the discipline), many computer science and engineering aspects, and finally a rich coverage of mathematical underpinnings, ranging from geometry to approximation theory.

The intended audience for this volume are researchers from areas outside of CAGD wishing to get a broad yet thorough exposure to the field. Researchers inside the field will find a wealth of material to complement their expertise. Graduate students will find a guide to new and promising research areas, leading to MS and PhD theses. And a layperson should find enough material to simply get an appreciation of an exciting and unfolding discipline.

Gerald Farin, Tempe, AZ
Josef Hoschek, Darmstadt, Germany
Myung-Soo Kim, Seoul, Korea

Contributors

Wolfgang Boehm, Applied Geometry and Computergraphics, Technical University of Braunschweig, Germany

E-mail: W.Boehm@tu-bs.de

Homepage: <http://www.tu-bs.de/institute/cagd>

Carl de Boor, Department of Computer Sciences, University of Wisconsin, 1210 West Dayton Street, Madison, WI 53706-1685, USA

E-mail: deboor@cs.wisc.edu

Homepage: <http://www.cs.wisc.edu/~deboor>

Byoung K. Choi, Department of Industrial Engineering, KAIST (Korea Advanced Institute of Technology), 373-1, Kusung-dong, Yusong-gu, Taejon, 305-701, Korea

E-mail: bkchoi@bezier.kaist.ac.kr

Homepage: <http://bezier.kaist.ac.kr>

Hyeong In Choi, School of Mathematical Sciences, Seoul National University, Seoul 151-742, Korea

E-mail: hichoi@math.snu.ac.kr

Homepage: <http://www.mtl.snu.ac.kr/~hichoi>

Wendelin Degen, University of Stuttgart, Pfaffenwaldring 57, 70569 Stuttgart, Germany

E-mail: degen@mathematik.uni-stuttgart.de

Homepage: <http://www.mathematik.uni-stuttgart.de/mathB/1st2/degen>

Gershon Elber, Department of Computer Science, Technion – Israel Institute of Technology, Haifa, 32000, Israel

E-mail: gershon@cs.technion.ac.il

Homepage: <http://www.cs.technion.ac.il/~gershon>

Gerald Farin, Department of Computer Science and Engineering, Arizona State University, P.O. Box 875406, Tempe, AZ 85287-5406, USA

E-mail: farin@enws05.cagd.eas.asu.edu

Homepage: <http://www.eas.asu.edu/~csdept/people/faculty/farin.html>

Rida T. Farouki, Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA 95616, USA

E-mail: farouki@ucdavis.edu

Homepage: <http://mae.ucdavis.edu/~farouki>

Richard Franke, Department of Mathematics, Naval Postgraduate School, Monterey, CA 93943-5216, USA

E-mail: dick@richardfranke.com, rfranke@nps.navy.mil

Homepage: <http://www.math.nps.navy.mil/~rfranke>

Thomas A. Grandine, The Boeing Company, P.O. Box 3707, MS 7L-21, Seattle, WA 98124, USA

E-mail: Thomas.Grandine@PSS.Boeing.com

Hans Hagen, Computer Science Department, University of Kaiserslautern, P.O. Box 30 49, 67653 Kaiserslautern, Germany

E-mail: hagen@informatik.uni-kl.de

Homepage: <http://davinci.informatik.uni-kl.de/~hagen/>

Chang Yong Han, School of Mathematical Sciences, Seoul National University, Seoul 151-742, Korea

E-mail: cyinblue@hotmail.com

Dianne Hansford, NURBS Depot, 4952 E. Mockingbird Lane, Paradise Valley, AZ 85253, USA

E-mail: hansford@3dcompress.com

Christoph M. Hoffmann, Department of Computer Sciences, Purdue University, 1398 Computer Science Building, West Lafayette, IN 47907-1398, USA

E-mail: cmh@cs.purdue.edu

Homepage: <http://www.cs.purdue.edu/people/cmh>

Klaus Höllich, Mathematical Institute A, University of Stuttgart, Pfaffenwaldring 57, 70569 Stuttgart, Germany

E-mail: hollig@mathematik.uni-stuttgart.de

Homepage: <http://www.mathematik.uni-stuttgart.de/mathA/1st2>

Josef Hoschek, Technische Universität Darmstadt, FB Mathematik, AG 3, Schloßgartenstr. 7, D-64289 Darmstadt, Germany

E-mail: hoschek@mathematik.tu-darmstadt.de

Robert B. Jerard, ME Department, University of New Hampshire, Durham, New Hampshire, USA

E-mail: robert.jerard@unh.edu

Homepage: <http://www.unh.edu/mechanical-engineering/People/jerard.html>

Robert Joan-Arinyo, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Catalonia, Spain

E-mail: robert@lsi.upc.es

Homepage: <http://www.lsi.upc.es/~robert>

Bert Jüttler, Johannes Kepler University, Institute of Analysis, Department of Applied Geometry, Altenberger Str. 69, 4040 Linz, Austria

E-mail: Bert.Juettler@jku.at

Homepage: <http://www.ag.jku.at>

Bo H. Kim, KRISO (Korea Research Institute of Ship and Ocean Engineering), 171, Jang-dong, Yusong-gu, Taejon, 305-343, Korea

E-mail: kbh@kriso.re.kr

Homepage: <http://www.kriso.re.kr>

Myung-Soo Kim, School of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea

E-mail: mskim@cse.snu.ac.kr

Homepage: <http://cse.snu.ac.kr/mskim>

Leif P. Kobbelt, Aachen University of Technology (RWTH), Ahornstr. 55, 52074 Aachen, Germany

E-mail: kobbelt@cs.rwth-aachen.de

Homepage: <http://www-i8.informatik.rwth-aachen.de>

Stefan Leopoldseder, Institut für Geometrie, TU Wien, Wiedner Hauptstraße 8-10, A-1040 Wien, Austria

E-mail: stefan@geometrie.tuwien.ac.at

Homepage: <http://www.geometrie.tuwien.ac.at/leopoldseder>

Suresh K. Lodha, Department of Computer Science, University of California, Santa Cruz, CA 95064, USA

E-mail: lodha@cse.ucsc.edu

Homepage: <http://www.soe.ucsc.edu/~lodha>

Takashi Maekawa, Department of Ocean Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Room 5-426A, Cambridge, MA 02139-4307, USA

E-mail: tmaekawa@mit.edu

Homepage: <http://deslab.mit.edu/DesignLab/people/Maekawa.html>

Ralph Martin, Dept. of Computer Science, Cardiff University, 5, The Parade, P.O. Box 916, Cardiff, Wales, CF24 3XF, UK

E-mail: ralph@cs.cf.ac.uk

Homepage: <http://ralph.cs.cf.ac.uk>

Mike Neamtu, Department of Mathematics, Vanderbilt University, 1326 Stevenson Center, Nashville, TN 37240-0001, USA

E-mail: neamtu@math.vanderbilt.edu

Homepage: <http://atlas.math.vanderbilt.edu/~neamtu>

Nicholas M. Patrikalakis, Department of Ocean Engineering, Massachusetts Institute of Technology, MIT Room 5-428, 77 Massachusetts Avenue, Cambridge, MA 02139-4307, USA

E-mail: nmp@mit.edu

Homepage: <http://oe.mit.edu/people/patrikal.htm>

Jörg Peters, Dept of Computer Information Science and Engineering, University of Florida, Gainesville, FL 32611-6120, USA

E-mail: jorg@cise.ufl.edu

Homepage: <http://www.cis.ufl.edu/~jorg>

Helmut Pottmann, Institut für Geometrie, TU Wien, Wiedner Hauptstraße 8-10, A-1040 Wien, Austria

E-mail: pottmann@geometrie.tuwien.ac.at

Homepage: <http://www.geometrie.tuwien.ac.at/pottmann>

Hartmut Prautzsch, Universität Karlsruhe, IBDS, Geometrische Datenverarbeitung, 76128 Karlsruhe, Germany

E-mail: prau@ira.uka.de

Homepage: <http://i33www.ira.uka.de/~prau>

Malcolm Sabin, University of Cambridge, Numerical Geometry Ltd, 26 Abbey Lane, Lode, CB5 9EP, England

E-mail: malcolm@geometry.demon.co.uk

Homepage: <http://www.damtp.cam.ac.uk/user/na/people/Malcolm/mas.html>

Gerik Scheuermann, Computer Science Department, University of Kaiserslautern, P.O. Box 30 49, 67653 Kaiserslautern, Germany
E-mail: scheuer@informatik.uni-kl.de
Homepage: <http://www-hagen.informatik.uni-kl.de/~scheuer>

Thomas W. Sederberg, Department of Computer Science, Brigham Young University, Provo, UT 84602, USA
E-mail: tom@cs.byu.edu
Homepage: <http://tom.cs.byu.edu/~tom>

Hans-Peter Seidel, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
E-mail: hpseidel@mpi-sb.mpg.de
Homepage: <http://www.mpi-sb.mpg.de/~hpseidel>

Vadim Shapiro, Mechanical Engineering & Computer Sciences, University of Wisconsin, 1513 University Avenue, Madison, WI 53705, USA
E-mail: vshapiro@engr.wisc.edu
Homepage: http://www.engr.wisc.edu/me/faculty/shapiro_vadim.html

Kokichi Sugihara, Department of Mathematical Informatics, University of Tokyo, Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
E-mail: sugihara@simplex.t.u-tokyo.ac.jp
Homepage: <http://www.simplex.t.u-tokyo.ac.jp/~sugihara/Welcomee.html>

Tamas Varady, Geometric Modelling Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences, H-1111 Budapest, Kende u, 13-17, Hungary
E-mail: varady@sztaki.hu
Homepage: <http://www.sztaki.hu/~varady>

Wenping Wang, Department of Computer Science and Information Systems, The University of Hong Kong, Room 421, Chow Yei Ching Building, Pokfulam Road, Hong Kong
E-mail: wenping@csis.hku.hk
Homepage: <http://www.csis.hku.hk/~wenping>

Michael G. Wagner, Department of Computer Science and Engineering, Arizona State University, P.O. Box 875406, Tempe, AZ 85287-5406, USA
E-mail: wagner@asu.edu
Homepage: <http://www.eas.asu.edu/~csdept/people/faculty/wagner.html>

Hans J. Wolters, Signature Bioscience, Inc. 21124 Cabot Boulevard, Hayward, CA 94545, USA
E-mail: hwolters@signaturebio.com

Frank Zeilfelder, Universität Mannheim, Institut für Mathematik, A 5, C 227, 68131 Mannheim, Germany

E-mail: zeilfeld@euklid.math.uni-mannheim.de

Homepage: <http://www.math.uni-mannheim.de/~lsmath4>

Jianmin Zheng, Department of Mathematics, Zhejiang University, Hangzhou, 310027, China

E-mail: zheng@cs.byu.edu

Chapter 1

A History of Curves and Surfaces in CAGD

Gerald Farin

This article provides a historical account of the major developments in the area of curves and surfaces as they entered the area of CAGD – Computer Aided Geometric Design – until the middle 1980s. We adopt the definition that CAGD deals with the construction and representation of free-form curves, surfaces, or volumes.

1.1. INTRODUCTION

The term CAGD was coined by R. Barnhill and R. Riesenfeld in 1974 when they organized a conference on that topic at the University of Utah. That conference brought together researchers from the U.S. and from Europe and may be regarded as the founding event of the field. It resulted in the widely influential proceedings [8]. The first textbook, “Computational Geometry for Design and Manufacture” by I. Faux and M. Pratt [63], appeared in 1979. The journal “Computer Aided Geometric Design” was founded in 1984 by R. Barnhill and W. Boehm. Its cover is shown in Figure 1.1.

Another early conference was one held in Paris in 1971. It focussed on automotive design and was organized by P. Bézier, then president of the Société des Ingénieurs de l’Automobile. The proceedings were published by the journal “Ingénieurs de l’Automobile.”

A series of workshops started in 1982 at the Mathematics Research Institute at Oberwolfach; these were organized by R. Barnhill, W. Boehm, and J. Hoschek. Ten years later, a parallel development started at the Computer Science Research Institute Schloss Dagstuhl initiated by H. Hagen. In the U.S., a conference series was organized by SIAM (Society for Industrial and Applied Mathematics); the first one being held 1983 at Troy, N.Y., and organized by H. McLaughlin. In the U.K., the conference series “Mathematics of Surfaces” was initiated by the IMA (Institute for Mathematics and Applications). A Norwegian/French counterpart was started by L. Schumaker, T. Lyche, and P.-J. Laurent.

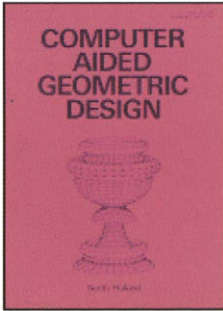


Figure 1.1. The cover of the journal CAGD. It shows a drawing by P. Uccello (ca. 1430).

1.2. EARLY DEVELOPMENTS

The earliest recorded use of curves in a manufacturing environment seems to go back to early AD Roman times, for the purpose of shipbuilding. A ship's ribs – wooden planks emanating from the keel – were produced based on templates which could be reused many times. Thus a vessel's basic geometry could be stored and did not have to be recreated every time. These techniques were perfected by the Venetians from the 13th to the 16th century. The form of the ribs was defined in terms of tangent continuous circular arcs – NURBS in modern parlance. The ship hull was obtained by varying the ribs' shapes along the the keel, an early manifestation of today's tensor product surface definitions. No drawings existed to define a ship hull; these became popular in England in the 1600s. The classical “spline,” a wooden beam which is used to draw smooth curves, was probably invented then. The earliest available mention of a “spline” seems to be [51] from 1752. This “shipbuilding connection,” described by H. Nowacki [103], was the earliest use of constructive geometry to define free-form shapes, see Figure 1.2. More modern developments linking marine and CAGD techniques may be found in [10,100,115,136].

Another key event originated in aeronautics. In 1944, R. Liming wrote a book entitled “Analytical Geometry with Application to Aircraft” [95]. Liming worked for the NAA (North American Aviation) during World War II; this company built fighter planes such as the legendary Mustang. In his book, classical drafting methods were combined with computational techniques for the first time. Conics were used in the aircraft as well as in the shipbuilding industries before, essentially based on constructions going back to Pascal and Monge. Traditionally, these constructions found their way onto the draftsman's drawing board in the form of blueprints which served as the basic product definition. Liming realized that an alternative was more efficient: store a design in terms of numbers instead of manually traced curves. Thus he translated the classical drafting constructions into numerical algorithms. The advantage: numbers can be stored in unambiguous tables and leave no room to individual interpretations of drawings. Liming's work was very influential in the 1950s when it was widely adopted by U.S. aircraft companies. Figure 1.3 shows one of Liming's constructions. Another researcher was also involved in the

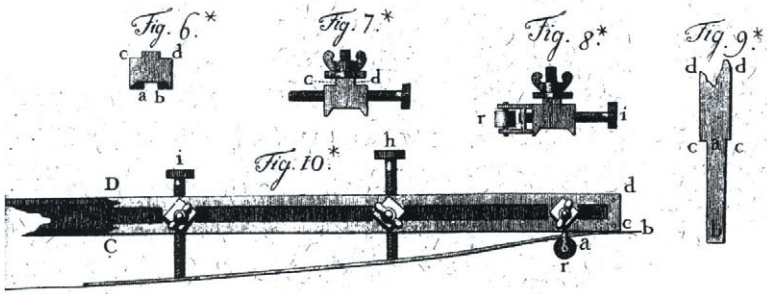


Figure 1.2. Splines: a mechanical spline from the 1700s.

transition of aircraft drawings to computations; this was S. Coons, see [34]. Coons later gained fame for his work at MIT.

Another early influential development for CAGD was the advent of numerical control (NC) in the 1950s. Early computers were capable of generating numerical instructions which drove milling machines used for the production of dies and stamps for sheet metal parts. At MIT, the APT programming language was developed for this purpose. A problem remained: all relevant information was stored in the form of blueprints,¹ and it was not clear how to communicate that information to the computer which was driving a milling machine. Digitizing points off the blueprints and fitting curves using familiar techniques such as Lagrange interpolation failed early on. New blueprint-to-computer concepts were needed. In France, de Casteljaou and Bézier went far beyond that task by enabling designers to abandon the manual blueprint process all together.

In the U.S., J. Ferguson at Boeing and S. Coons at MIT provided alternative techniques. General Motors developed its first CAD/CAM system DAC-I (Design Augmented by Computer). It used the fundamental curve and surface techniques developed at GM by researchers such as C. de Boor and W. Gordon.

In the U.K., A.R. Forrest began his work on curves and surfaces after being exposed to S. Coons' ideas. His PhD thesis (Cambridge) includes work on shape classification of cubics, rational cubics, and generalizations of Coons patches [65]. M. Sabin worked for British Aircraft Corporation and was instrumental in developing the CAD system "Numerical Master Geometry." He developed many algorithms that were later "reinvented." This includes work on offsets [118], geometric continuity [116], or tension splines [119]. Sabin received his PhD from the Hungarian Academy of Sciences in 1977, a seemingly odd choice which is explained by the close collaboration between researchers in Cambridge, U.K., and their counterparts in Hungary, under the leadership of J. Hatvany.

All these approaches took place in the 1960s. For quite a while, they existed in isolation until the seventies started to see a confluence of different research approaches, culminating

¹Liming's conic constructions were an exception but were not widely available outside the aircraft industry.

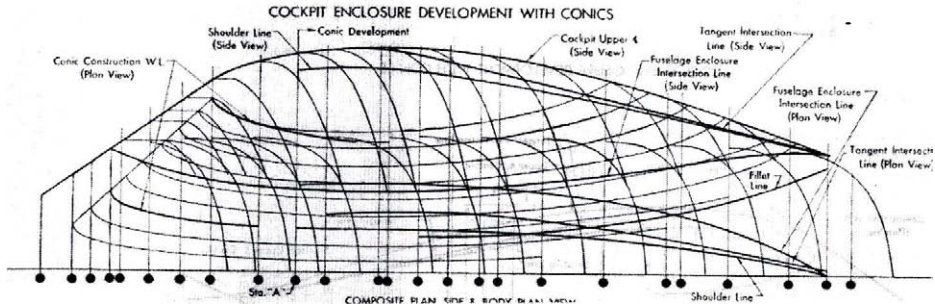


Figure 1.3. R. Liming: conic construction of a fighter aircraft cockpit.

in the creation of a new discipline, CAGD.

Without the advent of computers, a discipline such as CAGD would not have emerged. The initial main use of these computers was not so much to compute complex shapes but simply to produce the information necessary to drive milling machines. That information was typically output to a punch tape by a main frame computer. That tape was then transferred to the control unit of a milling machine.

The main interest of a designer was not so much the milling machine; it was rather a plotter which could quickly graph a designer's concepts. Early plotters were the size of a billiard table or larger; this was natural as drawings for most automotive parts were produced to scale. Plotting, or drafting, was so important that almost all of CAD was aimed at producing drawings – in fact, CAD was often considered to stand for “Computer Aided Drafting” (or “Draughting,” in British English). Before the advent of these systems, trivial-sounding tasks were extremely time consuming. For example, producing a new view of a complex wireframe object from existing views would take a draftsman a week or more; using computers, it became a matter of seconds.

A milestone in display hardware was the use of CRTs, or Cathode Ray Terminals. These went back to oscillographs which were used for many scientific applications. CRTs (not in use for CAD applications any more) displayed an image by “drawing” curves on a screen. Another dimension was added to simple display technology by adding an interactive component to it. The first interactive graphics system was invented by I. Sutherland at MIT in 1963, see [134]. His thesis was part of the CAD project at MIT; S. Coons was a member of his PhD committee. See Figure 1.4 for an illustration of Sutherland's prototype.

1.3. DE CASTELJAU AND BÉZIER

In 1959, the French car company Citroën hired a young mathematician in order to resolve some of the theoretical problems that arose from the blueprint-to-computer challenge.



Figure 1.4. I. Sutherland’s Sketchpad system.

The mathematician was Paul de Faget de Casteljaou, who had just finished his PhD. He began to develop a system which primarily aimed at the *ab initio* design of curves and surfaces instead of focusing on the reproduction of existing blueprints.

He adopted the use of Bernstein polynomials for his curve and surface definitions from the very beginning, together with what is now known as the de Casteljaou algorithm. Figure 1.5 shows a part of his 1963 technical report [44].

The breakthrough insight was to use *control polygons* (*courbes à pôles*), a technique that was never used before. Instead of defining a curve (or surface) through points *on* it, a control polygon utilizes points *near* it. Instead of changing the curve (surface) directly, one changes the control polygon, and the curve (surface) follows in a very intuitive way. In the area of differential geometry, concepts similar to control polygons were devised as early as 1923, see [17], but had no impact on any applications.

De Casteljaou’s work was kept a secret by Citroën for a long time. The first public mention of the algorithm (although not including a mention of the inventor) is [93]. W. Boehm was the first to give de Casteljaou recognition for his work in the research community. He found out about de Casteljaou’s technical reports and coined the term “de Casteljaou algorithm” in the late seventies.

Another place to learn about Citroën’s CAGD efforts was its competitor Renault, also located in Paris. There, during the early 1960s, Pierre Bézier headed the design department and also realized the need for computer representations of mechanical parts. Bézier’s efforts were influenced by the knowledge of similar developments at Citroën, but he proceeded in an independent manner. Bézier’s initial idea was to represent a “basic curve” as the intersection of two elliptic cylinders, see Figure 1.6. The two cylinders were defined inside a parallelepiped. Affine transformations of this parallelepiped would then result in affine transformations of the curve. Later, Bézier moved to polynomial formula-

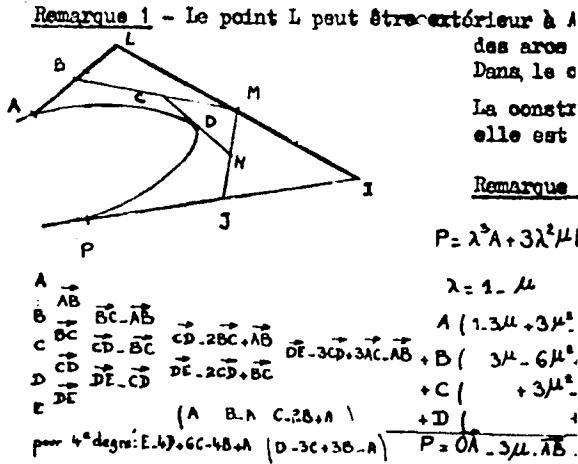


Figure 1.5. De Casteljau's description of his algorithm.

tions of this initial concept and also extended it to higher degrees. The result turned out to be identical to de Casteljau's curves, only the mathematics involved was different. A member of Bézier's team, D. Vernet independently developed the de Casteljau algorithm. See also Bézier's chapter in [59].

Bézier's work was widely published, see [15,12-14,138], and soon came to the attention of A.R. Forrest. He realized that Bézier curves could be expressed in terms of Bernstein polynomials – i.e., in the form that de Casteljau had used since the late fifties! Forrest's article on Bézier curves [66] was very influential and helped popularize Bézier curves considerably. The Renault CAD/CAM system UNISURF was based entirely on Bézier curves and surfaces. It influenced developments by the French aircraft company Dassault who built a system called EVE. Later, that system evolved into CATIA (Computer Aided Three-dimensional Interactive Application). Bézier also invented a method to deform whole assemblies of surfaces by embedding them into a cube and then deforming it using trivariate "Bézier cubes," see [12,16] and Section 1.5.

De Casteljau retired from Citroën in 1989 and became active in publishing. In 1985, he wrote "Formes à Pôles," [45] which introduced the concept of blossoming.²

P. Bézier died in Paris in 1999.

1.4. PARAMETRIC CURVES

Curves were employed by draftsmen for centuries; the majority of these curves were circles, but some were "free-form." Those are curves arising from applications such as ship hull design to architecture. When they had to be drawn exactly, the most common tool was a set of templates known as *French curves*. These are carefully designed wooden curves

²The term "blossoming" is due to L. Ramshaw who independently discovered the concept, see [110].

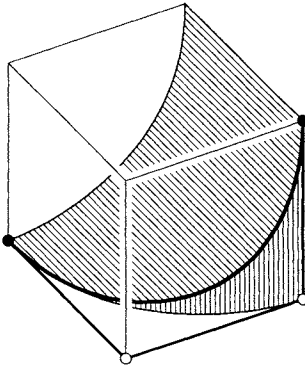


Figure 1.6. Bézier's "basic curve."

and consist of pieces of conics and spirals. A curve is drawn in a piecewise manner by tracing appropriate parts of a French curve.

Another mechanical tool, called a *spline* was also used. This was a flexible strip of wood that was held in place and shape by metal weights, known as *ducks*. When drawings had to be produced to scale, the attics (or lofts) of buildings were used to accommodate the large size drawings – the word *lofting* has its origins here. A spline “tries” to bend as little as possible, resulting in shapes which are both aesthetically pleasing and physically optimal. The mathematical counterpart to a mechanical spline is a *spline curve*, one of the most fundamental parametric curve forms.

The differential geometry of parametric curves was well understood since the late 1800s after work by Serret/Frenet. On the other hand, research in approximation theory and numerical analysis focused entirely on nonparametric functions. Both areas were brought together when they became important building blocks of CAGD.

Since the middle 1950s, the US aircraft company Boeing employed software based on Liming's conic constructions in the design of airplane fuselages. In a different part of the company, J. Ferguson and D. MacLaren developed a different kind of curve for the design of wings. They had the idea to piece cubic space curves together so that they formed composite curves which were overall twice differentiable [97,64]. These curves could easily interpolate to a set of points. They were referred to as spline curves since they minimize a functional similar to the physical properties of mechanical splines.

The meaning of the term “spline curve” has since undergone a subtle change. Instead of referring to curves that minimize certain functionals, spline curves are now mostly thought of as piecewise polynomial (or rational polynomial) curves with certain smoothness properties.

Ferguson derived his spline equations using the piecewise monomial form. But he also used the cubic Hermite form (then referred to as F-curves) which defines a cubic in terms of two endpoints and two endpoint derivatives. S. Coons used this curve type to build the patches which were named after him. In the UK, A.R. Forrest continued Coons' ideas and extended cubic Hermite curves to rational cubics, see [65].

The most fundamental parametric curve form are the Bézier curves; see Section 1.3.

Many of the basic properties can be found in the papers by Bézier, Vernet, and Forrest (see above). Some later results include conditions for C^r joins between Bézier curves, see Stárk [131], the discovery that Bézier curves are numerically more stable than other curve forms, see Farouki/Rajan [62], and the development of the blossoming principle, see Ramshaw [110] and de Casteljau [46]. A symbolic technique for Bézier curves and surfaces was developed by M. Hosaka and F. Kimura [83], although it was known to W. Boehm in 1972.³

As an early alternative to parametric curves were explicit curve segments with individual local coordinate systems. These curves are known as Wilson-Fowler splines [68]. A similar curve type was employed in the TABCYL (TABulated CYLinder) routines of the APT (Automatic Programmed Tool) language. After the advent of parametric curves, these piecewise explicit curves began to disappear.

Another early curve scheme are *biarcs*. These are piecewise circular arcs which are pieced together to allow for tangent continuity. It is possible to fit two tangent continuous circles to two points and two tangents. If several points and tangents are given, one obtains a *circle spline*. The advantage of these curves is the fact that NC machines can process circular arcs directly, i.e., without a conversion to a dense polygon as is needed for standard parametric splines. A drawback of circle splines is their piecewise constant and hence discontinuous curvature. The first developments are due to K. Bolton [23], followed by M. Sabin [121]; a generalization to 3D was given by T. Sharrock [127].

1.5. RECTANGULAR SURFACES

Parametric surfaces were well understood after early work by Gauss and Euler. They were immediately adopted in early CAD/CAM developments: A standard application is tracing a surface for plotting or for driving a milling tool. Parametric surfaces are well-suited for both tasks. The most popular of all surface methods was to become the tensor product surface. It was first introduced by C. de Boor [39] for the case of bicubic spline interpolation. Theoretical studies of parametric surfaces for the purpose of interpolation and approximation go back to [33,88,87,135,122,132] but had little influence on the development of industrial methods.

In the late 1950s, parametric surfaces were studied at several companies in Europe and the U.S. The first published result is due to J. Ferguson at Boeing, see [64]. Ferguson used an array of bicubic patches which interpolated to a grid of data points. While Ferguson developed C^2 cubic spline curves in the same paper, his surfaces were only C^1 .⁴ This was due to the introduction of zero twists at the corner of every bicubic patch.⁵ Ferguson's bicubic patches were also known as F-patches, and were also attributed to S. Coons.

Coons devised a simple formula to fit a patch between any four arbitrary boundary curves [35], known as the *bilinearly blended Coons patch*. These surfaces were used in the sixties by Ford (Coons was a consultant). A generalization, capable of interpolating a rectangular network of curves, was devised by W. Gordon at General Motors, see [71,

³Private communication

⁴Clearly, he was unaware of de Boor's paper [39] – it appeared in a journal not likely to be read by practitioners of the time.

⁵This fact, often leading to unsatisfactory shapes, was not explicitly mentioned in the article and is hidden among pseudo-code.

72]. All these methods are sometimes labeled “transfinite interpolation,” in that they interpolate to arbitrary boundary curves (having a “transfinite” number of points on them).

While the basic Coons patch had no restrictions on the boundary curves other than they have to meet at the patch corners, a common use was to restrict the boundary curves to be parametric cubics in Hermite form. Then the use of zero corner twists led to the above F-patch.

The basic (bilinearly blended) Coons patch does not lend itself to the construction of composite smooth surfaces. Additions to the basic method led to the *bicubically blended Coons patch*. It is the generalization of cubic Hermite curve interpolation to the transfinite surface case and allows for the prescription of tangent data in addition to the boundary curves. As a consequence, certain incompatible situations could arise. J. Gregory was the first to address this problem and also to devise a “compatibly corrected” interpolant, see [77]. When applied to cubic boundary curves and cubic derivative information, this interpolant yields a rational patch. A “translation” of this approach into a Bézier-like form was carried out by H. Chiyokura and F. Kimura [29,28]. It led to the Japanese CAD/CAM system DESIGNBASE.

Rectangular surfaces are a map of a rectangular domain into 3D. As a special case, we may map the domain to a 2D parametric surface, resulting in a distortion of the domain rectangle. If we embed a curve in this domain rectangle, we will obtain a deformed curve. A 3D surface may be embedded inside a 3D cube. This cube may be distorted using *trivariate* polynomials, resulting in a deformed surface. Such deformations are useful if global shape changes in a surface are wanted which would be too tedious to describe in terms of moving control points. The first mention of these volume deformations appears to be in J. Ferguson’s article [64], although no applications are given. Coons was also aware of the possibility of trivariate volumes, see [35]. The first practical use is due to Bézier who described how to use volume deformations in car design [16]. Volume deformations in Bézier form were rediscovered by Sederberg and Parry [126], who used them in a graphics environment.

1.6. B-SPLINE CURVES AND NURBS

B-splines (short for Basis Splines) go back to I. Schoenberg who introduced them in 1946 [123] for the case of uniform knots. B-splines over nonuniform knots go back to a review article by H. Curry in 1947 [38]. In 1960, C. de Boor started to work for the General Motors Research labs and began using B-splines as a tool for geometry representation. He later became one of the most influential proponents of B-splines in approximation theory. The recursive evaluation of B-spline curves is due to him and is now known as the de Boor algorithm [40].

It is based on a recursion for B-splines which was independently discovered by de Boor, L. Mansfield, and M. Cox [37]. It was this recursion that made B-splines a truly viable tool in CAGD. Before its discovery, B-splines were defined using a tedious divided difference approach which was numerically very unstable. For a detailed discussion, see [42].

Spline functions are important in approximation theory, but in CAGD, *parametric spline curves* are much more important. These were introduced in 1974 by R. Riesenfeld

and W. Gordon [73] (the paper is a synopsis of Riesenfeld’s PhD thesis [112]) who realized that de Boor’s recursive B-spline evaluation was the natural generalization of the de Casteljaou algorithm. B-spline curves include Bézier curves as a proper subset and soon became a core technique of almost all CAD systems. A first B-spline-to-Bézier conversion was found by W. Boehm [19]. Several algorithms were soon developed that simplified the mathematical treatment of B-spline curves; these include Boehm’s knot insertion algorithm [20], the Oslo algorithm by E. Cohen, T. Lyche, and R. Riesenfeld [32], and the introduction of the blossoming principle by L. Ramshaw [110] and P. de Casteljaou [45].

The generalization of B-spline curves to NURBS – Nonuniform rational B-splineS – has become the standard curve and surface form in the CAD/CAM industry. They offer a unified representation of spline and conic geometries: every conic as well as every spline allows a piecewise rational polynomial representation. The origin of the term NURBS is unclear; but the term was certainly a bad choice: it explicitly excludes the popular uniform B-spline curves. The first systematic NURB treatment goes back to K. Versprille’s PhD thesis [139]. Versprille was a student of S. Coons’ who started working with rational curves in the sixties [36]. Coons’ work on rational curves also influenced A.R. Forrest, who wrote his PhD thesis in 1968 [65].

Versprille based rational curves and surfaces in homogeneous (or projective) space. This kind of geometry had already gained importance in the graphics community because of the widespread use of central projections, see Riesenfeld [114].

The development at Boeing is exemplary for the emergence of NURBS. The company realized that different departments employed different kinds of geometry software; worse, those geometries were incompatible. The Liming-based conic software produced elliptic arcs which could not directly be imported into the Ferguson-based spline system and vice-versa. Thus NURBS were adopted as a standard since they would allow a unified geometry representation.⁶ Companies such as Boeing, SDRC, or Unigraphics (Versprille’s first employer) soon initiated making NURBS an IGES⁷ standard.

A special case of NURBS is given by *rational Bézier curves*. Even more specialized are conic sections, or rational quadratic Bézier curves. Their treatment goes back to Forrest’s PhD thesis [65]. A rational generalization of the de Casteljaou algorithm was given by G. Farin [57] in 1983; a (dual) projective formulation was discovered by J. Hoschek [84].

1.7. TRIANGULAR PATCHES

There are (at least) two ways to describe a bivariate polynomial surface. One is as a tensor product, using a rectangular domain. The other one is to write it in terms of barycentric coordinates with respect to a triangular domain. Both methods are outlined in the chapter on Bézier Techniques.

While tensor product surfaces are far more often encountered, triangular ones have been around for a long time also. The first uses of these surfaces goes back to Finite Elements, where they are referred to as “elements.” The simplest type is the linear element, which is

⁶As it turned out, this was not entirely true: some of Liming’s more complicated surface constructions do not allow a NURB representation.

⁷Initial Graphics Exchange Standard, developed to facilitate geometry data exchange between different companies.

simply a planar triangular facet. Its use goes back to the very beginning of Ritz-Galerkin methods. Higher order elements use C^1 quintic elements [92] or C^1 split triangle cubic elements. The latter one, devised by Clough and Tocher as a finite element [31], gained some popularity in the context of scattered data interpolation, see [58].

From a historical perspective, it is interesting to observe that early finite element research on triangular patches did not make use of the elegant formalism offered by the use of barycentric coordinates or the Bernstein-Bézier form. Consequently, those papers were fairly tedious – using barycentric coordinate techniques, many pages of proofs can be reduced to a few lines of geometric arguments about Bézier meshes.

Triangular patches in Bernstein form (called Bézier triangles although Bézier never made any mention of them) are due to P. de Casteljaeu; however, that work was never published (see above). Since Bézier triangles use trivariate Bernstein polynomials which did exist in approximation theory, several researchers developed concepts that were closely related to Bézier triangles: Stancu [133], Frederickson [69], Sabin [120,121].

M. Sabin gave conditions for smooth joins between adjacent triangular patches; G. Farin [55] gave conditions for C^r joins. Early research on Bézier triangles focussed on equilateral domain triangles; Farin [56,58] discussed the case of arbitrarily shaped domain triangles.

Bézier triangles, first conceived by an automotive researcher, found their way into approximation theory in the 1980's. Spaces of piecewise polynomials over triangulations were studied by L. Schumaker and Alfeld see [1,2].

Coons-like triangular patches were studied in the US during the 1970's and 1980's. See Barnhill *et al* [4], Barnhill and Gregory [7], or Nielson [102].

1.8. SUBDIVISION SURFACES

At the 1974 CAGD conference at the University of Utah, one of the presenters was the graphics artist G. Chaikin. He presented a curve generation method that did not fit the mold of any of the other methods of the conference. Starting from a closed 2D polygon, and using a process of continual “chopping off corners,” he arrived at a smooth limit curve, see [27]. At the conference, both R. Riesenfeld and M. Sabin argued that Chaikin had invented an iterative way to generate uniform quadratic B-spline curves, see [113].

In 1987, C. de Boor discovered that “corner cutting” generalizations of Chaikin's algorithm also produce continuous curves [43]. He also pointed out that Chaikin's algorithm was a special case of a class of algorithms described by G. de Rham much earlier [47,48]. Similar results were discovered by J. Gregory and R. Qu in 1988 – although that work was only published in 1996, see [78].

Chaikin's algorithm was the starting point for the initial work on subdivision surfaces, going back to two articles in the no. 6 volume of the journal *Computer Aided Design*, 1978. They were authored by D. Doo and M. Sabin [50] and E. Catmull and J. Clark [26].

Both papers have a similar flavor. Chaikin's algorithm can be generalized to tensor product surfaces in a straightforward way. Such surfaces have a rectilinear control mesh, and after analyzing the tensor product algorithm, Doo and Sabin reformulated it such that it could also be applied to control meshes of arbitrary topology. Catmull and Clark first

generalized Chaikin's algorithm to uniform cubic B-spline curves and its tensor product counterpart. Then, they also reformulated it for the case of control meshes of arbitrary topology. Both surface schemes yield smooth (G^1) surfaces. The Doo/Sabin surfaces have a piecewise biquadratic flavor; the Catmull/Clark ones have a piecewise bicubic flavor.

In 1987, C. Loop generalized triangular spline surfaces to a new G^1 subdivision surface type, see [96]. Its input control polygon can be any triangular mesh. Loop's algorithm is based on a subdivision scheme for so-called box-splines by W. Boehm [21] and H. Prautzsch [108].

All three of the above algorithms produce C^1 (for Doo/Sabin and Loop) or C^2 (for Catmull/Clark and Loop) surfaces if the control meshes are regular, i.e., rectilinear or regular triangular (all triangle vertices have valence six) for Loop. Where the control meshes do not behave like that, the surfaces will have *singular points*. These singular points hampered the analysis and practical use of subdivision surfaces. A first attempt at investigating these points was undertaken by D. Storry and A. Ball in 1986, see [3], although there is an eigenvalue analysis of the Doo-Sabin process in the original paper [50]. Subsequently, subdivision surfaces gained more popularity, most notably in the computer animation industry.

Early work on subdivision surfaces focussed on *approximating* surfaces. In 1987, N. Dyn, J. Gregory, and D. Levin discovered an *interpolating* subdivision scheme, called the 4-point curve scheme, see [54]. It was generalized to surfaces – to the so-called “butterfly algorithm” – by the same authors in 1990, see [53].

1.9. SCIENTIFIC APPLICATIONS

Many areas of science need to model phenomena for which only a set of discrete measurements is available – an example is a weather map where data are collected at a set of weather stations, but a continuous model of temperature, pressure, etc., is desired. Since the location of the data sites has no structure (such as being on a regular grid), the term “scattered data” was coined. If function values are assigned to these data sites, a *scattered data interpolant* is a function which assumes the given function values at the data sites. The data sites are typically 2D, but may also be 3D.

A scattered data interpolant is a function which interpolates the given data values and gives reasonable estimates in between. One of the first scattered data interpolants is *Shepard's method*, see [74,5,128]. It computes the function value at an arbitrary (evaluation) point as a linear combination of all given function values, the coefficients being related to the distance of the data sites to the evaluation point. The method itself is too poorly behaved to be viable on its own, but it was used as an ingredient in other methods (see the above references).

Another approach was taken by R. Hardy [80] in 1971, who generalized the concept of splines to surfaces. His surfaces utilize *radial basis functions* which are bell-shaped functions (reminiscent of univariate B-splines) with their maxima at the data sites. Hardy's method was used by many practitioners, but it was not known if the method would work in all cases. A proof for this was given by C. Micchelli in 1986 at the Mathematics Research Center in Oberwolfach, Germany.

A different generalization of splines to surfaces is the method of *thin plate splines* due

to J. Duchon [52]. Thin plate splines minimize an “energy functional”, similar to the one that spline curves minimize. While spline curves simulate the behavior of an elastic beam, thin plate splines minimize the behavior of a thin elastic plate.

The U.K. statistician R. Sibson developed a scheme that he called *nearest neighbor interpolation*. It is based on the concept of Voronoi diagrams, also known as Dirichlet tessellations. Sibson showed that any point in the convex hull of a given set of points may be written as a unique linear combination of its neighbors [129]. If the coefficients of this combination are then used to blend given function values, a scattered data interpolant arises [130]. The interpolant is only C^0 at the given data points, but it is C^1 otherwise. Uses for this method are in the geosciences as well as in image processing.

1.10. SHAPE

When a designer studied a curve on a full size drawing, he could visually detect shape defects such as “flat spots”, unwanted inflections, etc. A good CAD system would then provide methods to improve the given shape.

As the design process moved away from the drawing board and into computer screens, this visual inspection process was not feasible any more since a full scale drawing was out of the question. The scaled down display offered by the computer did not allow to detect shape imperfections in a direct way. Consequently, computer methods had to be developed that allowed easy assessment of shape.

Among the first published methods are curve hodographs, see [11]. These are the plot of a curve’s first derivative curve. Since differentiation is a roughing process, curve imperfections appear “magnified” in the hodograph.

Hodographs do not display the geometry of the curve alone; they depend on the parametrization as well. A more geometric tool is the *curvature plot*; it plots the curvature of a curve. Since it involves second order derivatives as well as first order ones, it is also a more sensitive tool. An early paper on the use of curvature plots is by Nutbourne *et al.* [104].

In unpublished work, curvature plots were used by H. Burchardt at General Motors. After de Boor left the company in 1964, interpolating cubic and quintic spline curves had become a tool of choice. Since they are C^2 , they are also curvature continuous. In many cases, however, this does not guarantee pleasant shape, and so Burchardt developed a proprietary scheme that was *shape optimizing*. A published version is in [24].

Shape optimization became an important element early on in the development of CAGD. Since interpolating spline curves were known to exhibit unwanted undulations, alternatives were studied, typically involving curvature – see [99,100,82,125].

A different approach is to take an existing curve or surface and inspect its shape: if it is imperfect, apply a *fairing* procedure. Such methods typically aim at removing noise from either data points or control polygons; early work is reported by J. Kjellander [89], J. Hoschek [86] and Farin *et al* [60].

A curve which is curvature continuous may not be twice differentiable. This fact, when properly exploited, leads to curve generation schemes which have more degrees of freedom than do “standard” spline curves. The extra degrees of freedom are referred to as *shape parameters* and the resulting curves are called *geometrically continuous* or G^2 . Early work

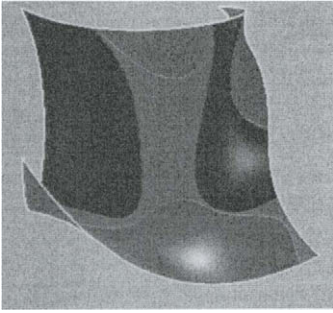


Figure 1.7. Gaussian curvature. Figure courtesy A.R. Forrest

on this subject is due to G. Geise [70]. Interpolation schemes based on G^2 continuity were independently developed by J. Manning [98] and G. Nielson [101] in 1974. Later work on the subject includes B. Barsky's β -splines [9], W. Boehm's γ -splines [22], and H. Hagen's τ -splines [79].

Shape does not only play a vital role in interpolation, but also in approximation. The *least squares method* is the most widespread approximation scheme. It was used early on in most industries; publications include [81,106,142]. The least squares method lends itself to the inclusion of conditions that aim at the shape of the result, not just at the closeness of fit. These conditions are typically the result of minimizing certain functionals (such as minimizing “wiggles”); an influential early example is the “smoothing spline” of Schoenberg and Reinsch, see [124,111], and also Powell [107].

For curves, curvature is a reliable shape measure. For surfaces, several such measures exist, including Gaussian, mean, or total curvatures. A.R. Forrest [67] was the first to use computer graphics for the interrogation of surface shape using curvatures as texture maps; see Fig. 1.7.

Another important shape measure for surfaces comes from the automotive industry. It is customary to put a car prototype in a showroom where the ceiling is lined with florescent light strips. Their reflections are carefully examined before the prototype is accepted. Early computer simulations of this procedure are reported in [90].

1.11. INFLUENCES AND APPLICATIONS

CAGD emerged through the influence of several areas in the 1950s and 1960s, but interactions with other fields of science and engineering were not limited to those years.

The first text on CAGD goes back to I. Faux and M. Pratt. Its title is “Computational Geometry for Design and Manufacture.” The meaning of the term “Computational Geometry” has since changed; it is used to describe a discipline which is concerned with the complexity of algorithms mostly dealing with discrete geometry. The defining text for this area is the one by Preparata and Shamos [109]. An important area of overlap between CAGD and Computational Geometry is that of *triangulation algorithms*. These

are concerned with finding a set of triangles having a given 2D point set as vertices. The first algorithm was published in 1971 by C. Lawson [94]. An algorithmic connection between triangulations and Voronoi diagrams was presented by Green and Sibson [76]. The n -dimensional case was covered by D. Watson [140].

2D triangulations are an important preprocessing tool for many surface fitting operations. 3D point sets arise when physical objects are digitized. Methods to triangulate them go back to Choi *et al.* [30].

Computer Graphics is an area with many CAGD interactions. Computer Graphics needs CAGD to model objects to be displayed, and for the very same reason, CAGD needs Computer Graphics. It was only after Sutherland's 1963 development of interactive graphics that one could interactively change control polygons of Bézier or B-spline objects. Display techniques for parametric surfaces go back to H. Gouraud [75] and were later improved by B. Phong [105] and J. Blinn [18], all at the University of Utah.

A fundamental display technique is *ray tracing*, due to T. Whitted [141]. It makes extensive use of computations of the intersection between a ray and the scene to be displayed. Hence the development of efficient intersection algorithms became important for Graphics.

Intersection algorithms are also important in many areas of CAD/CAM, where planar sections were the method of choice (and tradition) to display/plot objects. Algorithms for this task were developed by many companies, and were mostly kept confidential. Some early published work is by W. Carlson [25], T. Dokken [49] Barnhill *et al* [6]. The earliest reference seems to be by M. Sabin [117].

Another important type of numerical algorithms are offset curves and surfaces – early work includes papers by R. Farouki [61], J. Hoschek [85], R. Klass [91], W. Tiller and E. Hanson [137], the earliest one being a technical report by M. Sabin [118] from 1968.

REFERENCES

1. P. Alfeld. A case study of multivariate piecewise polynomials. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 149–160. SIAM, Philadelphia, 1987.
2. P. Alfeld and L. Schumaker. The dimension of bivariate spline spaces of smoothness r and degree $d \geq 4r + 1$. *Constructive Approximation*, 3:189–197, 1987.
3. A. Ball and D. Storry. A matrix approach to the analysis of recursively generated B-spline surfaces. *Computer-Aided Design*, 18(8):437–442, 1986.
4. R. Barnhill, G. Birkhoff, and W. Gordon. Smooth interpolation in triangles. *J Approx. Theory*, 8(2):114–128, 1973.
5. R. Barnhill, R. Dube, and F. Little. Properties of Shepard's surfaces. *Rocky Mtn. J of Math.*, 13(2):365–382, 1983.
6. R. Barnhill, G. Farin, M. Jordan, and B. Piper. Surface / surface intersection. *Computer Aided Geometric Design*, 4(1-2):3–16, 1987.
7. R. Barnhill and J. Gregory. Polynomial interpolation to boundary data on triangles. *Math. of Computation*, 29(131):726–735, 1975.
8. R. Barnhill and R. F. Riesenfeld, editors. *Computer Aided Geometric Design*. Academic Press, 1974.

9. B. Barsky. *The Beta-spline: a local representation based on shape parameters and fundamental geometric measures*. Ph.D. Thesis, Dept. of Computer Science, U. of Utah, 1981.
10. S. Berger, A. Webster, R. Tapia, and D. Atkins. Mathematical ship lofting. *J Ship Research*, 10:203–222, 1966.
11. P. Bézier. Définition numérique des courbes et surfaces I. *Automatisme*, XI:625–632, 1966.
12. P. Bézier. Définition numérique des courbes et surfaces II. *Automatisme*, XII:17–21, 1967.
13. P. Bézier. Procédé de définition numérique des courbes et surfaces non mathématiques. *Automatisme*, XIII(5):189–196, 1968.
14. P. Bézier. Mathematical and practical possibilities of UNISURF. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 127–152. Academic Press, 1974.
15. P. Bézier. *Essay de définition numérique des courbes et des surfaces expérimentales*. Ph.D. Thesis, University of Paris VI, 1977.
16. P. Bézier. General distortion of an ensemble of biparametric patches. *Computer-Aided Design*, 10(2):116–120, 1978.
17. W. Blaschke. *Differentialgeometrie*. Chelsea, 1953. Reprint of the original 1923 edition.
18. J. Blinn. *Computer display of curved surfaces*. Ph.D. Thesis, Dept. of Computer Science, U. of Utah, 1978.
19. W. Boehm. Cubic B-spline curves and surfaces in computer aided geometric design. *Computing*, 19(1):29–34, 1977.
20. W. Boehm. Inserting new knots into B-spline curves. *Computer-Aided Design*, 12(4):199–201, 1980.
21. W. Boehm. Subdividing multivariate splines. *Computer-Aided Design*, 15(6):345–352, 1983.
22. W. Boehm. Curvature continuous curves and surfaces. *Computer-Aided Design*, 18(2):105–106, 1986.
23. K. Bolton. Biarc curves. *Computer-Aided Design*, 7(2):89–92, 1975.
24. H. Burchardt, J. Ayers, W. Frey, and N. Sapidis. Approximation with aesthetic constraints. In N. Sapidis, editor, *Designing Fair Curves and Surfaces*, pages 3–28. SIAM, Philadelphia, 1994.
25. W. Carlson. An algorithm and data structure for 3D object synthesis using surface patch intersections. *Computer Graphics*, 16(3):255–263, 1982.
26. E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 1978.
27. G. Chaikin. An algorithm for high speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.
28. H. Chiyokura. *Solid Modeling with DESIGNBASE*. Addison-Wesley, 1988.
29. H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. *Computer Graphics*, 17(3):289–298, 1983.
30. B. Choi, H. Sin, Y. Yoon, and J. Lee. Triangulation of scattered data in 3D-space. *Computer-Aided Design*, 20(5):239–248, 1988.

31. R. Clough and J. Tocher. Finite element stiffness matrices for analysis of plates in blending. In *Proceedings of Conference on Matrix Methods in Structural Analysis*, 1965.
32. E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-splines and subdivision techniques in computer aided geometric design and computer graphics. *Comp. Graphics and Image Process.*, 14(2):87–111, 1980.
33. L. Collatz. Approximation von Funktionen bei einer und bei mehreren unabhängigen Veränderlichen. *Z. Angew. Math. Mech.*, 36:198–211, 1956.
34. S. Coons. Graphical and analytical methods as applied to aircraft design. *J. of Eng. Education*, 37(10), 1947.
35. S. Coons. Surfaces for Computer-Aided Design. Technical report, MIT, 1964. Available as AD 663 504 from the National Technical Information service, Springfield, VA 22161.
36. S. Coons. Rational bicubic surface patches. Technical report, MIT, 1968. Project MAC.
37. M. Cox. The numerical evaluation of B-splines. *J Inst. Maths. Applics.*, 10:134–149, 1972.
38. H. Curry. Review. *Math. Tables Aids Comput.*, 2:167–169, 211–213, 1947.
39. C. de Boor. Bicubic spline interpolation. *J. Math. Phys.*, 41:212–218, 1962.
40. C. de Boor. On calculating with B-splines. *J Approx. Theory*, 6(1):50–62, 1972.
41. C. de Boor. Splines as linear combinations of B-splines – a survey. In G. Lorentz, C. Chui, and L. Schumaker, editors, *Approximation Theory II*, pages 1–47. Academic Press, New York, 1976.
42. C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, 1978.
43. C. de Boor. Cutting corners always works. *Computer Aided Geometric Design*, 4(1-2):125–131, 1987.
44. P. de Casteljau. Courbes et surfaces à pôles. Technical report, A. Citroën, Paris, 1963.
45. P. de Casteljau. *Formes à Pôles*. Hermès, Paris, 1985.
46. P. de Casteljau. *Shape Mathematics and CAD*. Kogan Page, London, 1986.
47. G. de Rham. Un peu de mathématique à propos d’une courbe plane. *Elem. Math.*, 2:73–76; 89–97, 1947. Also in *Collected Works*, 678–689.
48. G. de Rham. Sur une courbe plane. *J Math. Pures Appl.*, 35:25–42, 1956. Also in *Collected Works*, 696–713.
49. T. Dokken. Finding intersections of B-spline represented geometries using recursive subdivision techniques. *Computer Aided Geometric Design*, 2(1-3):189–195, 1985.
50. D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, 1978.
51. H.L. Duhamel du Monceau. *Éléments de l’Architecture Navale ou Traité Pratique de la Construction des Vaisseaux*. 1752. Paris.
52. J. Duchon. Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In W. Schempp and K. Zeller, editors, *Constructive Theory of Functions of Several Variables*, pages 85–100. Springer-Verlag, 1977.
53. N. Dyn, J. Gregory, and D. Levin. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9:160–169, 1990.

54. N. Dyn, D. Levin, and J. Gregory. A 4-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, 4(4):257–268, 1987.
55. G. Farin. *Subsplines ueber Dreiecken*. Ph.D. Thesis, Technical University Braunschweig, Germany, 1979.
56. G. Farin. Bézier polynomials over triangles and the construction of piecewise C^r polynomials. Technical Report TR/91, Brunel University, Uxbridge, England, 1980.
57. G. Farin. Algorithms for rational Bézier curves. *Computer-Aided Design*, 15(2):73–77, 1983.
58. G. Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3(2):83–128, 1986.
59. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Morgan-Kaufmann, 2001. fifth edition.
60. G. Farin, G. Rein, N. Sapidis, and A. Worsey. Fairing cubic B-spline curves. *Computer Aided Geometric Design*, 4(1-2):91–104, 1987.
61. R. Farouki. Exact offset procedures for simple solids. *Computer Aided Geometric Design*, 2(4):257–279, 1985.
62. R. Farouki and V. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, 1987.
63. I. Faux and M. Pratt. *Computational Geometry for Design and Manufacture*. Ellis Horwood, 1979.
64. J. Ferguson. Multivariable curve interpolation. *J ACM*, 11(2):221–228, 1964.
65. A.R. Forrest. *Curves and surfaces for computer-aided design*. Ph.D. Thesis, Cambridge, 1968.
66. A.R. Forrest. Interactive interpolation and approximation by Bézier polynomials. *The Computer J*, 15(1):71–79, 1972. reprinted in *Computer-Aided Design* 22(9):527–537, 1990.
67. A.R. Forrest. On the rendering of surfaces. *Computer Graphics*, 13(2):253–259, 1979.
68. A. Fowler and C. Wilson. Cubic spline, a curve fitting routine. Technical Report Y-1400, Union Carbide, Oak Ridge, 1963.
69. L. Frederickson. Triangular spline interpolation / generalized triangular splines. Technical Report no. 6/70 and 7/71, Dept. of Math., Lakehead University, Canada, 1971.
70. G. Geise. Über berührende Kegelschnitte ebener Kurven. *ZAMM*, 42:297–304, 1962.
71. W. Gordon. Free-form surface interpolation through curve networks. Technical Report GMR-921, General Motors Research Laboratories, 1969.
72. W. Gordon. Spline-blended surface interpolation through curve networks. *J of Math. and Mechanics*, 18(10):931–952, 1969.
73. W. Gordon and R. Riesenfeld. B-spline curves and surfaces. In R. E. Barnhill and R. F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 95–126. Academic Press, 1974.
74. W. Gordon and J. Wixom. Shepard’s method of “metric interpolation” to bivariate and multivariate interpolation. *Math. of Computation*, 32(141):253–264, 1978.
75. H. Gouraud. Computer display of curved surfaces. *IEEE Trans. Computers*, C-20(6):623–629, 1971.
76. P. Green and R. Sibson. Computing Dirichlet tessellations in the plane. *The Com-*

- puter J*, 21(2):168–173, 1978.
77. J. Gregory. Smooth interpolation without twist constraints. In R. E. Barnhill and R. F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 71–88. Academic Press, 1974.
 78. J. Gregory and R. Qu. Nonuniform corner cutting. *Computer Aided Geometric Design*, 13(8):763–772, 1996.
 79. H. Hagen. Geometric spline curves. *Computer Aided Geometric Design*, 2(1-3):223–228, 1985.
 80. R. Hardy. Multiquadratic equations of topography and other irregular surfaces. *J Geophysical Research*, 76:1905–1915, 1971.
 81. J. Hayes and J. Holladay. The least-squares fitting of cubic splines to general data sets. *J. Inst. Maths. Applics.*, 14:89–103, 1974.
 82. B. Horn. The curve of least energy. *ACM Trans. on Math. Software*, 9(4), 1983.
 83. M. Hosaka and F. Kimura. A theory and methods for three dimensional free-form shape construction. *J of Info. Process.*, 3(3), 1980.
 84. J. Hoschek. Dual Bézier curves and surfaces. In R. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 147–156. North-Holland, 1983.
 85. J. Hoschek. Offset curves in the plane. *Computer-Aided Design*, 17(2):77–82, 1985.
 86. J. Hoschek. Smoothing of curves and surfaces. *Computer Aided Geometric Design*, 2(1-3):97–105, 1985.
 87. J. Neder. Interpolationsformeln für Funktionen mehrerer Argumente. *Scand. Actuaritidskrift*, 9:59–69, 1926.
 88. O. Kellogg. On bounded polynomials in several variables. *Math. Z.*, 27:55–64, 1928.
 89. J. Kjellander. Smoothing of cubic parametric splines. *Computer-Aided Design*, 15(3):175–179, 1983.
 90. R. Klass. Correction of local surface irregularities using reflection lines. *Computer-Aided Design*, 12(2):73–77, 1980.
 91. R. Klass. An offset spline approximation for plane cubics. *Computer-Aided Design*, 15(5):296–299, 1983.
 92. V. Kolar, J. Kratochvil, A. Zenisek, and M. Zlamal. Technical, physical, and mathematical principles of the finite element method. Academia, Czech. Academy of Sciences, Prague, Czech., 1971.
 93. J. Krautter and S. Parizot. Système d’aide à la définition et à l’usinage des surfaces de carrosserie. *Journal de la SIA*, 44:581–586, 1971. Special Issue: La commande numérique, P. Bézier, guest editor.
 94. C. Lawson. Transforming triangulations. *Discrete Mathematics*, 3:365–372, 1971.
 95. R. Liming. *Practical analytical geometry with applications to aircraft*. Macmillan, 1944.
 96. C. Loop. Smooth subdivision surfaces based on triangles. Master’s thesis, University of Utah, 1987.
 97. D. MacLaren. Formulas for fitting a spline curve through a set of points. *Boeing Appl. Math. Rpt.*, 2, 1958.
 98. J. Manning. Continuity conditions for spline curves. *The Computer J*, 17(2):181–186, 1974.
 99. E. Mehlum. A curve-fitting method based on a variational criterion for smoothness.

- BIT*, 4:213–223, 1964.
100. E. Mehlum and P. Sorenson. Example of an existing system in the shipbuilding industry: the AUTOKON system. *Proc. Roy. Soc. London*, A.321:219–233, 1971.
 101. G. Nielson. Some piecewise polynomial alternatives to splines under tension. In R. E. Barnhill and R. F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 209–235. Academic Press, 1974.
 102. G. Nielson. The side-vertex method for interpolation in triangles. *J of Approx. Theory*, 25:318–336, 1979.
 103. H. Nowacki. Splines in shipbuilding. *Proc. 21st Duisburg colloquium on marine technology*, May 2000.
 104. A. Nutbourne, P. McLellan, and R. Kensit. Curvature profiles for plane curves. *Computer-Aided Design*, 4(4):176–184, 1972.
 105. B. Phong. Illumination for computer-generated images. *Comm. ACM*, 18(6):311–317, 1975.
 106. M. Powell. The local dependence of least squares cubic splines. *SIAM J. Numer. Anal.*, 6:398–413, 1969.
 107. M. Powell. Curve fitting by splines in one variable. In G. Hayes, editor, *Numerical Approximation to Functions and Data*, pages 65–83. Athlone Press (London), 1970.
 108. H. Prautzsch. *Unterteilungsalgorithmen für multivariate Splines Ein geometrischer Zugang*. Ph.D. Thesis, TU Braunschweig, 1984.
 109. F. Preparata and M. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, 1985.
 110. L. Ramshaw. Blossoming: a connect-the-dots approach to splines. Technical report, Digital Systems Research Center, Palo Alto, Ca, 1987.
 111. C. Reinsch. Smoothing by spline functions. *Numer. Math*, 10:177–183, 1967.
 112. R. Riesenfeld. *Applications of B-spline approximation to geometric problems of computer-aided design*. Ph.D. Thesis, Dept. of Computer Science, Syracuse U., 1973.
 113. R. Riesenfeld. On Chaikin's algorithm. *Computer Graphics and Image Processing*, 4(3):304–310, 1975.
 114. R. Riesenfeld. Homogeneous coordinates and projective planes in computer graphics. *IEEE Computer Graphics and Applications*, 1:50–55, 1981.
 115. D. Rogers and S. Satterfield. B-spline surfaces for ship hull design. *Computer Graphics (Proc. Siggraph 80)*, 14(3):211–217, 1980.
 116. M. Sabin. Conditions for continuity of surface normals between adjacent parametric surfaces. Technical Report VTO/MS/151, British Aircraft Corporation, 1968.
 117. M. Sabin. General interrogations of parametric surfaces. Technical Report VTO/MS/150, British Aircraft Corporation, 1968.
 118. M. Sabin. Offset parametric surfaces. Technical Report VTO/MS/149, British Aircraft Corporation, 1968.
 119. M. Sabin. Parametric splines in tension. Technical Report VTO/MS/160, British Aircraft Corporation, 1970.
 120. M. Sabin. Trinomial basis functions for interpolation in triangular regions (Bézier triangles). Technical report, British Aircraft Corporation, 1971.
 121. M. Sabin. *The use of piecewise forms for the numerical representation of shape*. Ph.D. Thesis, Hungarian Academy of Sciences, Budapest, Hungary, 1976.

122. H. Salzer. Note on interpolation for a function of several variables. *Bull. AMS*, 51:279–280, 1945.
123. I. Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. *Quart. Appl. Math.*, 4:45–99, 1946.
124. I. Schoenberg. Spline functions and the problem of graduation. *Proc. Nat. Acad. Sci*, 52:947–950, 1964.
125. D. Schweikert. An interpolation curve using a spline in tension. *J Math. Phys.*, 45:312–317, 1966.
126. T. Sederberg and S. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4):151–160, 1986. SIGGRAPH proceedings.
127. T. Sharrock. Biarcs in three dimensions. In R. Martin, editor, *The Mathematics of Surfaces II*, pages 395–412. Oxford University Press, 1987.
128. D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proc. ACM National Conference*, pages 517–524, 1968.
129. R. Sibson. A vector identity for the Dirichlet tessellation. *Math. Proc. of the Cambridge Philosophical Society*, 87:151–155, 1980.
130. R. Sibson. A brief description of the natural neighbour interpolant. In V. Barnett, editor, *Interpreting Multivariate Data*. John Wiley & Sons, 1981.
131. E. Staerk. *Mehrfach differenzierbare Bézierkurven und Bézierflächen*. Ph.D. Thesis, T. U. Braunschweig, 1976.
132. D. Stancu. De l'approximation par des polynômes du type bernstein des fonctions de deux variables. *Comm. Akad. R. P. Romaine*, 9:773–777, 1959.
133. D. Stancu. Some Bernstein polynomials in two variables and their applications. *Soviet Mathematics*, 1:1025–1028, 1960.
134. I. Sutherland. *Sketchpad, a Man-Machine Graphical Communication System*. Ph.D. Thesis, MIT, Dept. of Electrical Engineering, 1963.
135. H. Thacher. Derivation of interpolation formulas in several independent variables. *Annals New York Acad. Sc.*, 86:758–775, 1960.
136. F. Theilheimer and W. Starkweather. The fairing of ship lines on a high speed computer. *Numerical Tables Aids Computation*, 15:338–355, 1961.
137. W. Tiller and E. Hanson. Offsets of two-dimensional profiles. *IEEE Computer Graphics and Applications*, 4:36–46, 1984.
138. D. Vernet. Expression mathématique des formes. *Ingenieurs de l'Automobile*, 10:509–520, 1971.
139. K. Versprille. *Computer Aided Design Applications of the Rational B-spline Approximation Form*. Ph.D. Thesis, Syracuse U., 1975.
140. D. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer J*, 24(2):167–172, 1981.
141. T. Whitted. An improved illumination model for shaded display. *CACM*, 23:343–349, 1980.
142. H. Wilf. The stability of smoothing by least squares. *Proc. Amer. Math. Soc.*, 15:933–937, 1964.

Chapter 2

Geometric Fundamentals

Wolfgang Boehm and Hartmut Prautzsch

The following recalls the facts and terminology mostly used in Geometry. It may serve also as a first introduction to geometric tools, for more detailed and founded descriptions see the list of references, in particular [7]. Without additional effort most of the discussed topics can be presented in spaces of any dimension.

2.1. AFFINE FUNDAMENTALS

Many properties of computational geometry and its applications do not need the distance of points but only the concepts of parallelism and ratio.

2.1.1. Points and vectors

In general a point in n -space is fixed by its coordinates with respect to some Cartesian system¹. Nevertheless, we start our observations with affine aspects.

Let $\mathbf{a} = [\alpha_1 \dots \alpha_n]^t$ and $\mathbf{b} = [\beta_1 \dots \beta_n]^t$ denote two points, its difference $\mathbf{v} = \mathbf{b} - \mathbf{a}$ is called a vector, and one has $\mathbf{b} = \mathbf{a} + \mathbf{v}$. In particular the column $\mathbf{o} = [0 \dots 0]^t = \mathbf{a} - \mathbf{a}$ denotes the null-vector.

Let $\mathbf{a}_0, \dots, \mathbf{a}_d$ denote $d + 1$ points in n -space, $d \leq n$. The d vectors $\mathbf{v}_i = \mathbf{a}_i - \mathbf{a}_0$, $i = 1, \dots, d$, are called linearly dependent if $\alpha_1 \mathbf{v}_1 + \dots + \alpha_d \mathbf{v}_d = \mathbf{o}$, with at least one non-zero α_i , otherwise these vectors are called linearly independent. If $\mathbf{v}_1, \dots, \mathbf{v}_d$ are linearly independent, then they span a linear space \mathbf{V}^d , and the points $\mathbf{a}_0, \dots, \mathbf{a}_d$ are called affinely independent and span an affine space \mathbf{A}^d , d is called their dimension.

2.1.2. Affine systems

A point \mathbf{a}_0 and n linearly independent vectors \mathbf{v}_i define an affine system $[\mathbf{a}_0, \mathbf{v}_1 \dots \mathbf{v}_n]$ of the \mathbf{A}^n . In this system every point $\mathbf{p} = [\eta_1 \dots \eta_n]^t$ can be written uniquely as

$$\mathbf{p} = \mathbf{a}_0 + \xi_1 \mathbf{v}_1 + \dots + \xi_n \mathbf{v}_n = \mathbf{a}_0 + \mathbf{A}\mathbf{x}, \quad (2.1)$$

¹Matrix notation is preferred. To simplify the notation, a point as well as its coordinate column will be denoted by the same bold letter. Note that this notation depends on the coordinate system.

where $\mathbf{A} = [\mathbf{v}_1 \dots \mathbf{v}_n]$ and $\mathbf{x} = [\xi_1 \dots \xi_n]^t$. The ξ_i are called the affine coordinates of \mathbf{p} with respect to $[\mathbf{a}_0, \mathbf{v}_1 \dots \mathbf{v}_n]$, \mathbf{a}_0 is called the origin of the affine system.

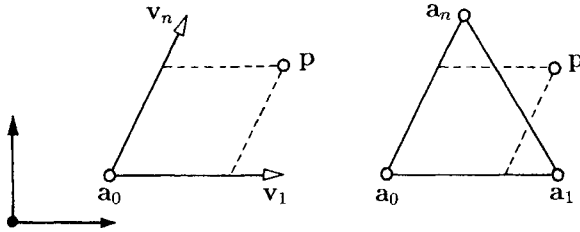


Figure 2.1. Affine and barycentric coordinates.

To distinguish between points and vectors described by elements \mathbf{a} of the \mathbb{R}^n one may add a further coordinate, ϵ , where

$$\epsilon = \begin{cases} 0 & \text{if } \mathbf{a} \text{ represents a vector,} \\ 1 & \text{if } \mathbf{a} \text{ represents a point.} \end{cases}$$

This convention can help to avoid mistakes in handling with points and vectors, see also Subsection 4.1 on homogeneous coordinates.

2.1.3. Barycentric coordinates

Let $\mathbf{a}_0, \dots, \mathbf{a}_n$ denote $n + 1$ affinely independent points of the \mathbf{A}^n and let $\mathbf{v}_i = \mathbf{a}_i - \mathbf{a}_0$. One may rewrite equation (1) as

$$\mathbf{p} = \xi_0 \mathbf{a}_0 + \xi_1 \mathbf{a}_1 + \dots + \xi_n \mathbf{a}_n, \quad \xi_0 = 1 - (\xi_1 + \dots + \xi_n).$$

The ξ_0, \dots, ξ_n are called barycentric coordinates of \mathbf{p} with respect to the frame $[\mathbf{a}_0 \dots \mathbf{a}_n]$. Note that $\xi_0 + \dots + \xi_n = 1$. Note also that any n of the $\xi_i, i \neq j$, represent affine coordinates of \mathbf{p} with respect to an affine system with origin \mathbf{a}_j .

It follows immediately that a vector $\mathbf{v} = \mathbf{b} - \mathbf{a}$ has barycentric coordinates ν_0, \dots, ν_n that sum to zero, $\nu_0 + \dots + \nu_n = 0$. Note that the sum of the coordinates ξ_i or ν_i corresponds to ϵ above.

2.1.4. Affine subspaces and parallelism

Let points $\mathbf{a}_0, \dots, \mathbf{a}_d \in \mathbf{A}^n$ be given. The point

$$\mathbf{p} = \xi_0 \mathbf{a}_0 + \xi_1 \mathbf{a}_1 + \dots + \xi_d \mathbf{a}_d, \quad 1 = \xi_0 + \xi_1 + \dots + \xi_d,$$

is called an affine combination of the points \mathbf{a}_i . Let $d \leq n$ and let the points \mathbf{a}_i be affinely independent. Then they span a d -dimensional subspace $\subset \mathbf{A}^n$. With barycentric coordinates its points are written as affine combinations or with affine coordinates as

$$\mathbf{p} = \mathbf{a}_0 + \xi_1 \mathbf{v}_1 + \dots + \xi_d \mathbf{v}_d, \tag{2.2}$$

where $\mathbf{v}_i = \mathbf{a}_i - \mathbf{a}_0$. This subspace is called a line, plane or hyperplane if $d = 1, 2$ or $n - 1$, respectively.

For any given \mathbf{p} and given $\mathbf{a}_0, \mathbf{v}_1, \dots, \mathbf{v}_d$ as elements of \mathbb{R}^n , the linear combination (2) represents a system of n linear equations for the ξ_i . It is solvable only if \mathbf{p} lies in the subspace.

Conversely, for varying ξ_i the presentation (2) can be viewed as the solution of some linear system of $n - d$ equations for an unknown $\mathbf{p} = [\eta_1 \dots \eta_n]^t$. In particular, if $d = n - 1$, this system consists of one linear equation, say

$$u_0 + \mathbf{u}^t \mathbf{p} = u_0 + u_1 \eta_1 + \dots + u_n \eta_n = 0,$$

or with barycentric coordinates of \mathbf{p}

$$u_0 \eta_0 + (u_1 + u_0) \eta_1 + \dots + (u_n + u_0) \eta_n = 0,$$

where additionally $\eta_0 + \dots + \eta_n = 1$.

Hyperplanes are called linearly independent if the rows $[u_0 \ u_1 \ \dots \ u_n]$ of their coefficients are linearly independent. Consequently, a subspace of dimension d of \mathbf{A}^n can be obtained as the intersection of $n - d$ linearly independent hyperplanes. In particular, a point can be obtained as the intersection of n hyperplanes.

Note that the points of an affine subspace solve an inhomogeneous system, while their differences, the vectors, solve the corresponding homogeneous system.

A line $\mathbf{p} = \mathbf{a} + \lambda \mathbf{v}$ is called parallel to a subspace $\mathbf{B} \subset \mathbf{A}^n$ if the coordinates of its points solve the homogeneous system corresponding to \mathbf{B} . Moreover, two affine subspaces \mathbf{A} and \mathbf{B} are parallel if all lines of \mathbf{A} are parallel to \mathbf{B} , or vice versa.

2.1.5. Affine maps and axonometric images

Equation (1) allows two interpretations. First, it expresses \mathbf{p} with respect to a new affine system $[\mathbf{a}_0, \mathbf{v}_1 \dots \mathbf{v}_n]$, where $\mathbf{x} = [\xi_1 \dots \xi_n]^t$ are the new coordinates of \mathbf{p} . For example, the equation $u_0 + \mathbf{u}^t \mathbf{p} = 0$ of a hyperplane reads in the new coordinates $q_0 + \mathbf{q}^t \mathbf{x} = 0$, where $q_0 = u_0 + \mathbf{u}^t \mathbf{a}_0$ and $\mathbf{q}^t = \mathbf{u}^t \mathbf{A}$.

Second, it represents an affine map $\phi : \mathbf{x} \rightarrow \mathbf{p}$, where \mathbf{x} and \mathbf{p} represent affine coordinates of two points. In particular, ϕ maps the origin $\mathbf{o} = [0 \dots 0]$ and the unit vectors² $\mathbf{e}_i = [\delta_{i,1} \dots \delta_{i,n}]^t$ into \mathbf{a}_0 and \mathbf{v}_i , $i = 1, \dots, n$, respectively. This important property defines the map uniquely and allows a simple design and investigation of an affine map. Note that ϕ maps the points \mathbf{x} of the hyperplane $q_0 + \mathbf{q}^t \mathbf{x} = 0$ above into the points \mathbf{p} of the hyperplane $u_0 + \mathbf{u}^t \mathbf{p} = 0$.

If the barycentric coordinate columns of points are denoted by the corresponding hollow letters, ϕ is written as

$$\mathbb{p} = \mathbf{A} \mathbf{x}, \quad \text{where } \mathbf{A} = [\mathfrak{a}_0 \ \mathfrak{a}_1 \ \dots \ \mathfrak{a}_n].$$

Note that affine maps preserve affine combinations, i.e. one has

$$\phi[\xi_0 \mathbf{a}_0 + \dots + \xi_d \mathbf{a}_d] = \xi_0 [\phi \mathbf{a}_0] + \dots + \xi_d [\phi \mathbf{a}_d],$$

and also preserve parallelism and the ratio of parallel distances, i.e. $\mathbf{w} = \lambda \mathbf{v}$ is mapped into $[\phi \mathbf{w}] = \lambda [\phi \mathbf{v}]$.

² $\delta_{i,j}$ is the so-called Kronecker-delta, $\delta_{i,j} = 1$ if $i = j$ and $= 0$ else

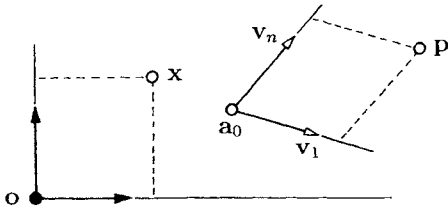


Figure 2.2. Affine map and new affine system.

If the v_i are linearly dependent, the map is degenerated. In particular, if $\eta_{d+1} = \dots = \eta_n = 0$ for all x , the map creates an axonometric image as used in descriptive geometry. Simple examples are the so-called cavalier and military projections, see [7].

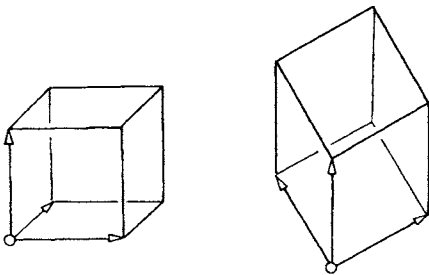


Figure 2.3. Cavalier and military projection.

2.1.6. Affine combinations and A-frame

Many algorithms in CAGD are based on repeated affine combinations. Consider two points, a_0 and a_1 . The affine combination

$$p = (1 - \alpha)a_0 + \alpha a_1$$

represents a point on the line spanned by a_0 and a_1 , and α represents an affine scale with $\alpha = 0$ corresponding to a_0 and $\alpha = 1$ corresponding to a_1 .

The term $r[p; a_0 a_1] = \alpha / (1 - \alpha)$ is called the ratio of p with respect to $a_0 a_1$.

Consider three points a_0, a_1, a_2 , and the affine combinations

$$b_0 = (1 - \alpha)a_0 + \alpha a_1 \quad \text{and} \quad b_1 = (1 - \alpha)a_1 + \alpha a_2,$$

both related by the same α , and the subsequent affine combination

$$\begin{aligned} \mathbf{p} &= (1 - \beta)\mathbf{b}_0 + \beta\mathbf{b}_1 \\ &= (1 - \alpha)(1 - \beta)\mathbf{a}_0 + (\alpha(1 - \beta) + (1 - \alpha)\beta)\mathbf{a}_1 + \alpha\beta\mathbf{a}_2. \end{aligned} \tag{2.3}$$

Obviously, the resulting point \mathbf{p} is symmetric in α and β . This means that α and β can be interchanged. This symmetry property is referred to as A-frame lemma and is a fundamental tool in de Casteljau's work [16].

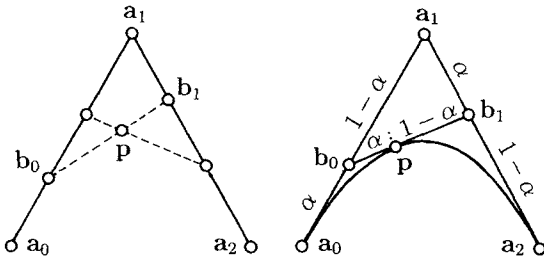


Figure 2.4. A-frame lemma and affine A-frame.

Let $\alpha = \beta$. Then (3) reduces to

$$\mathbf{p} = (1 - \alpha)^2\mathbf{a}_0 + 2\alpha(1 - \alpha)\mathbf{a}_1 + \alpha^2\mathbf{a}_2.$$

For fixed α the involved six points represent the so-called affine A-Frame, which is of great importance in Bernstein-Bézier methods. For varying α the point \mathbf{p} traces out a parabola, defined by \mathbf{a}_0 and \mathbf{a}_2 with tangents that intersect in \mathbf{a}_1 .

2.2. CONIC SECTIONS AND QUADRICS

The simplest figures in affine space besides lines and planes are conic sections, or more general, quadrics. They are studied conveniently by their quadratic equations, see [7].

2.2.1. Quadrics in affine space

In an affine space a quadric \mathbf{Q} consists of all points \mathbf{x} satisfying a quadratic equation

$$Q(\mathbf{x}, \mathbf{x}) = \mathbf{x}^t C \mathbf{x} + 2\mathbf{c}'\mathbf{x} + c = 0,$$

where $C = C^t$ is a symmetric non-zero matrix.

The intersection with a subspace is a quadric again. In particular, if the subspace is a line, one gets a pair of points. Note that these points can be real, coalescing or non-real.

A point \mathbf{m} is called a midpoint of \mathbf{Q} , if $Q(\mathbf{x}, \mathbf{x})$ is symmetric with respect to \mathbf{m} . This is the case for all solutions of

$$C \mathbf{m} + \mathbf{c} = \mathbf{o}.$$

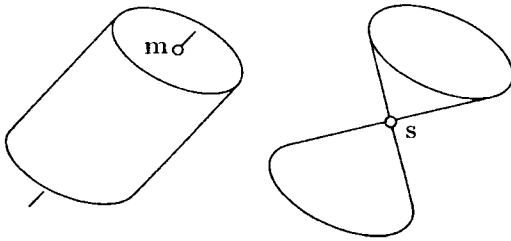


Figure 2.5. Midpoint and singular point.

Note that a solution may not exist. If a midpoint s lies on Q , it satisfies $Cs + c = 0$, and $c^t s + c = 0$, and is called a singular point, while Q degenerates to a cone.

2.2.2. Tangents and polar planes

A line L , given by $x = q + \lambda v$, where q is a point of Q , intersects Q in a second point. If both points coalesce, then L is a tangent of Q at q and satisfies $[Cq + c]^t v = 0$.

If additionally $v^t C v = 0$, then L lies completely on Q , and is called a generatrix of Q . Let $v = q - x$, then L is a tangent if

$$Q(q, x) = [Cq + c]^t x + c^t q + c = 0.$$

This equation for x represents a plane, the tangent plane of Q at q .

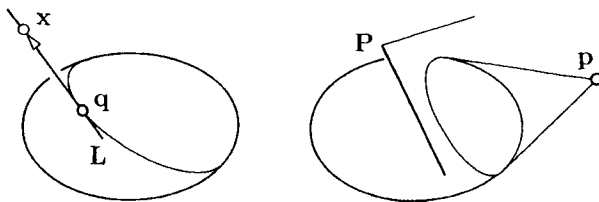


Figure 2.6. Tangent and polarity.

Replacing q by an arbitrary point p gives

$$Q(p, x) = p^t C x + c^t [x + p] + c = 0.$$

It represents the polar plane \mathbf{P} of the pole \mathbf{p} with respect to \mathbf{Q} . It intersects \mathbf{Q} in points \mathbf{q} with tangent planes through \mathbf{p} . Note that these points \mathbf{q} need not be real. Note also that $Q(\mathbf{p}, \mathbf{x})$ is symmetric in \mathbf{p} and \mathbf{x} .

A pair of points \mathbf{p}, \mathbf{x} is called conjugate with respect to \mathbf{Q} if $Q(\mathbf{p}, \mathbf{x}) = 0$. Hence the points of \mathbf{Q} are self-conjugate with respect to \mathbf{Q} . A pair of directions \mathbf{u}, \mathbf{v} is called conjugate with respect to \mathbf{Q} if $\mathbf{u}^t C \mathbf{v} = 0$. Conjugate elements play an important role in further investigations on quadrics in affine space.

Quadrics differ by the dimension of their midpoints or singularities, the dimension of their real generatrices and in affine space by the shape of their extensions to infinity.

2.2.3. Pascal's and Brianchon's theorems

From the past there is a lot of knowledge on conic sections. Of particular interest are the following two theorems on conic sections in the plane:

The three pairs of opposite sides of a hexagon inscribed to a conic section meet in three points of a line (*Pascal's theorem*).

The three connections of opposite points of a hexilateral circumscribed to a conic section intersects in one point (*Brianchon's theorem*).

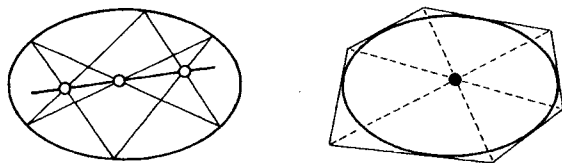


Figure 2.7. Pascal's and Brianchon's theorems.

As a consequence of these theorems a conic section is uniquely determined by five points or five tangents in the plane.

Of particular interest are the theorems if pairs of consecutive points or tangents coalesce. E.g., let $\mathbf{a}_0, \mathbf{a}_2$ denote two points of a conic section with tangents meeting at a point \mathbf{a}_1 . Let the points

$$\mathbf{b}_0 = (1 - \alpha)\mathbf{a}_0 + \alpha\mathbf{a}_1, \quad \text{and} \quad \mathbf{b}_1 = \beta\mathbf{a}_1 + (1 - \beta)\mathbf{a}_2. \quad (2.4)$$

span a third tangent. Its point of contact \mathbf{p} is easily obtained from Brianchon's theorem,

$$\mathbf{p} = [\beta\mathbf{b}_0 + \alpha\mathbf{b}_1]/(\alpha + \beta),$$

where α and β as in (4), see also [6].

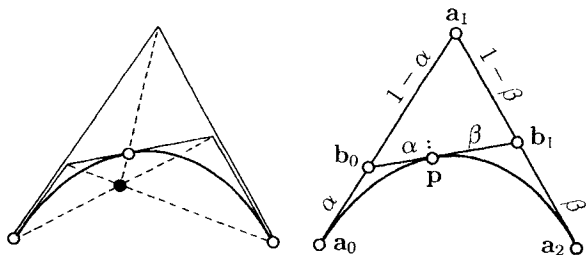


Figure 2.8. The projective A-frame and its affine representation.

2.3. THE EUCLIDEAN SPACE

The affine space \mathbf{A}^n is a Euclidean space denoted by \mathbf{E}^n and the corresponding vector space \mathbf{V}^n a Euclidean vector space if a dot product $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^t \mathbf{b}$ is given.

2.3.1. Cartesian coordinates

An affine system $[\mathbf{a}_0, \mathbf{v}_1 \dots \mathbf{v}_n]$ of \mathbf{E}^n is called Cartesian if³ $\mathbf{v}_i \cdot \mathbf{v}_j = \delta_{i,j}$ and it is positively oriented if $\det[\mathbf{v}_1 \dots \mathbf{v}_n] > 0$.

In Cartesian coordinates the distance of two points \mathbf{p} and \mathbf{q} is given by the length $\|\mathbf{v}\|$ of the vector $\mathbf{v} = \mathbf{q} - \mathbf{p}$,

$$\text{dist}(\mathbf{p} \mathbf{q}) = \|\mathbf{v}\| = \sqrt{\mathbf{v}^t \mathbf{v}},$$

and the angle φ of two vectors \mathbf{u} and \mathbf{v} is given by

$$\mathbf{u}^t \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \varphi.$$

In particular, both vectors are called orthogonal if $\cos \varphi = 0$, i.e. if $\mathbf{u}^t \mathbf{v} = 0$.

2.3.2. Gram-Schmidt orthogonalization

A Cartesian system $[\mathbf{a}_0, \mathbf{b}_1 \dots \mathbf{b}_d]$ of a subspace or the Euclidean space itself can easily be constructed from an affine system $[\mathbf{a}_0, \mathbf{v}_1 \dots \mathbf{v}_d]$ in \mathbf{E}^n with Gram-Schmidt's orthogonalization by alternating computation of the coefficients $\lambda_{i,j}$ and μ_i as follows:

Set $\mathbf{b}_1 = \mu_1 \mathbf{v}_1$ such that $\|\mathbf{b}_1\| = 1$.

Set $\mathbf{b}_2 = \mu_2 (\mathbf{v}_2 + \lambda_{2,1} \mathbf{b}_1)$ such that \mathbf{b}_2 is orthogonal to \mathbf{b}_1 and $\|\mathbf{b}_2\| = 1$.

...

Set $\mathbf{b}_d = \mu_d (\mathbf{v}_d + \lambda_{d,1} \mathbf{b}_1 + \dots + \lambda_{d,d-1} \mathbf{b}_{d-1})$,
such that \mathbf{b}_d is orthogonal to $\mathbf{b}_1, \dots, \mathbf{b}_{d-1}$ and $\|\mathbf{b}_d\| = 1$.

Note that in a Cartesian system the dot product is written as $\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^t \mathbf{v}$.

³see footnote 2

2.3.3. Euclidean motions and orthogonal projections

If the frame $[a_0, v_1 \dots v_n]$ is Cartesian, then (1) represents a Cartesian coordinate transformation or a Euclidean motion. Simple examples of motions in 3-space are the translation by v and the rotation around the 3-axis by an angle ζ , in matrices written as

$$p = v + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} x \quad \text{and} \quad p = o + \begin{bmatrix} \cos \zeta & -\sin \zeta & 0 \\ \sin \zeta & \cos \zeta & 0 \\ 0 & 0 & 1 \end{bmatrix} x,$$

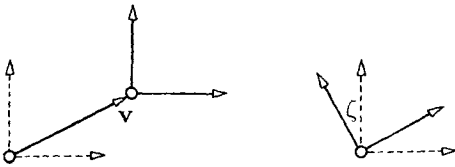


Figure 2.9. Translation and rotation.

respectively. In particular, let $p = B_i(\varphi) x$ describe the rotation around the i -axis by some angle φ . Any motion in 3-space can be written as

$$p = v + B_3(\gamma)B_1(\beta)B_3(\alpha) x,$$

where α, β, γ are the so-called Eulerian angles.

Any Euclidean motion followed by a map setting the coordinates $\eta_{d+1}, \dots, \eta_m$ of the image p to zero results in an orthogonal projection onto some d -dimensional subspace .

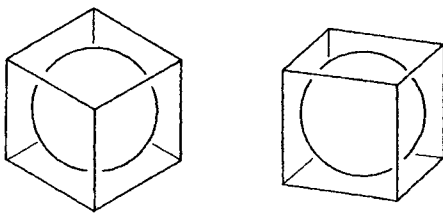


Figure 2.10. Isometric and dimetric orthogonal projection.

It should be mentioned that orthogonal projections are more informative than simple parallel projections and much more informative than perspectivities. They are the

only projections that map spheres to circles. Therefore orthogonal projections should be preferred in presenting technical objects.

2.3.4. Quadrics in Euclidean space

If the vectors \mathbf{v}_i of the Cartesian system $[\mathbf{a}_0, \mathbf{v}_1 \dots \mathbf{v}_n]$ are pairwise conjugate with respect to a quadric \mathbf{Q} , then the \mathbf{v}_i are principal axis directions of \mathbf{Q} and C is a diagonal matrix.

One easily checks that for a conic section given by its equation a rotation by an angle α where

$$\tan 2\alpha = 2c_{12}/(c_{11} - c_{22})$$

turns the coordinate axes into the axis directions of \mathbf{Q} and transforms C into diagonal form.



Figure 2.11. Principal axis transformation.

2.4. PROJECTIVE FUNDAMENTALS

Introducing points at infinity leads to the projective space and allows a unified and most elegant treatment of geometry⁴.

2.4.1. Homogeneous coordinates

Let ξ_1, \dots, ξ_n be affine coordinates of a point in \mathbf{A}^n with respect to an affine frame $[\mathbf{a}_0, \mathbf{v}_1 \dots \mathbf{v}_n]$ as above. Set $\xi_i = \beta_i/\beta_0$ with some $\beta_0 \neq 0$. Then the $\beta_0, \beta_1, \dots, \beta_n$ are homogeneous coordinates with respect to the given affine frame. Note that any non-zero multiple of the homogeneous coordinate column $\mathbf{b} = [\beta_0 \beta_1 \dots \beta_n]^t$ represents the same point. Note also that a point $\mathfrak{p} = \mathfrak{o}$ is undefined. It represents the so-called forbidden point.

As before $\beta_i = 0, i \neq 0$ represents the coordinate hyperplane $\xi_i = 0$. Further, $\beta_0 = 0$ represents points at infinity lying in the infinite or ideal hyperplane $\beta_0 = 0$. An affine space \mathbf{A}^n together with its ideal hyperplane forms a projective space \mathbf{P}^n , the projective extension of \mathbf{A}^n .

The advantage of this extension is the symmetry of homogeneous coordinates. Points at infinity are handled as points in any other plane. In particular, ideal points allow to

⁴“All geometry is projective geometry” [Arthur Cayley 1821-1895]

intersect parallel lines and subspaces - at infinity. Note that any non-zero multiple of a vector represents the same point at infinity.

Note also that $\beta_0 = 0$ and $\beta_0 \neq 0$ correspond to $\epsilon = 0$ and $\epsilon = 1$ in 1.2 above.

2.4.2. Projective coordinates

Let $\mathfrak{a}_0, \dots, \mathfrak{a}_d$ be linearly independent columns of homogeneous coordinates of $d + 1$ points in \mathbf{P}^n with integrated factors such that the sum $\mathfrak{a} = \mathfrak{a}_0 + \dots + \mathfrak{a}_d$ represents a given further point \mathfrak{a} called the unit point. These $d + 2$ points determine a projective frame $[\mathfrak{a}_0, \dots, \mathfrak{a}_d; \mathfrak{a}]$ of some projective subspace \mathbf{S} spanned by the points $\mathfrak{a}_0, \dots, \mathfrak{a}_d$. Any point \mathfrak{p} of this subspace can be represented by homogeneous projective coordinates $\mathfrak{x} = [\xi_0 \dots \xi_d]^t$ as

$$\rho \mathfrak{p} = \xi_0 \mathfrak{a}_0 + \xi_1 \mathfrak{a}_1 + \dots + \xi_d \mathfrak{a}_d, \quad \rho \neq 0. \tag{2.5}$$

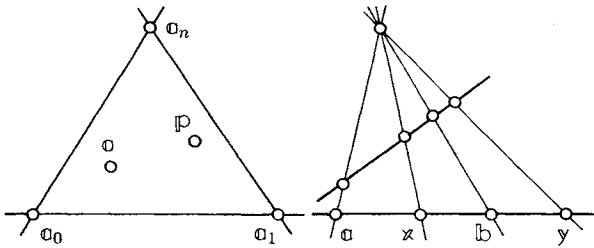


Figure 2.12. Projective system and crossratio.

In particular, if $\mathfrak{a}_i = [1, \mathbf{a}_i^t]^t$, and $\mathfrak{a} = [1, \mathbf{a}^t]^t$ then \mathbf{a} is the center $[\mathbf{a}_0 + \dots + \mathbf{a}_d]/d$ of the \mathbf{a}_i and the ξ_i are a multiple of the barycentric coordinates of \mathfrak{p} with respect to the affine frame $[\mathbf{a}_0 \dots \mathbf{a}_d]$.

2.4.3. Projective maps

The representation (5), with matrices written as $\rho \mathfrak{p} = \mathbf{A} \mathfrak{x}$, allows two interpretations. First it represents the point $\mathfrak{p} \in \mathbf{S}$ by new homogeneous coordinates \mathfrak{x} . Second it represents a projective map $\psi : \mathfrak{x} \rightarrow \mathfrak{p}$ of \mathbf{S} into \mathbf{P}^n .

In particular, ψ maps the fundamental points $\mathfrak{x}_i = [\delta_{0,i} \dots \delta_{d,i}]^t$ into $\mathfrak{a}_i, i = 0, \dots, d$ and the unit point $[1 \dots 1]^t$ into \mathfrak{a} . This determines the projective map uniquely - and \mathbf{A} except for a common factor ρ .

Note that a projective map does not preserve parallelism and ratio in general, but it preserves the cross ratio

$$cr[\mathfrak{x}\mathfrak{y}; \mathfrak{a}\mathfrak{b}] = r[\mathfrak{x}; \mathfrak{a}\mathfrak{b}]/r[\mathfrak{y}; \mathfrak{a}\mathfrak{b}].$$

In particular, if $cr[xy; \circ b] = -1$ then both pairs of points, xy and $\circ b$, lie in harmonic position. For example, let α be an affine scale, see 1.6, the pairs of points corresponding to $-1, +1$ and $0, \infty$ lie in harmonic position.

2.4.4. The procedure of inhomogeneizing

Any homogeneous equation in projective coordinates can easily be inhomogeneized by setting the homogeneizing coordinate to one. Any point $x = [\xi_0, \xi_0 x^t]^t$ or $v = [0, v^t]^t$ is simply inhomogeneized to x or v , respectively.

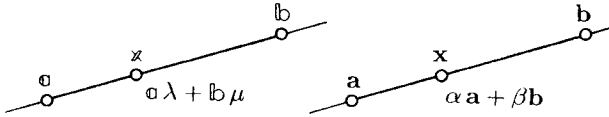


Figure 2.13. Inhomogeneizing the point of a line.

Of particular interest is the application of this procedure to the point x of a projective line given by

$$\rho x = \lambda a + \mu b.$$

Let $\rho = 1$ and let $a = [\alpha_0, \alpha_0 a^t]^t$ and $b = [\beta_0, \beta_0 b^t]^t$. Then inhomogeneizing $x = [\xi_0, \xi_0 x^t]^t$ results in the affine combination

$$x = \alpha a + \beta b, \text{ where } \alpha = \lambda \alpha_0 / \xi_0 \text{ and } \beta = \mu \beta_0 / \xi_0. \tag{2.6}$$

Similar results one gets by inhomogeneizing the points of a projective subspace (5) of higher dimension.

2.4.5. Repeated projective combinations

Repeated affine combinations and A-frames are often used in CAGD to compute polynomial curves and surfaces and can also be applied to the homogeneous coordinates of rational curves and surfaces. It is useful to inhomogeneize the resulting projective combinations by the procedure demonstrated above.

Moreover, after a first inhomogeneizing one can continue with the affine representation of the projective A-frame presented in Subsection 2.3. For more details on this procedure and its applications see [6].

2.4.6. Quadrics in projective space

In homogeneous or projective coordinates x the equation of a quadric Q simplifies to

$$Q(x) = x^t C x = 0, \text{ where } C^t = C,$$

and the polarity is written as

$$Q(p x) = p^t C x = 0.$$

Note that in homogeneous coordinates the midpoint of Q is the pole of the infinite plane. Note also that Q is a cylinder, if it has a singular point at infinity, etc.

2.4.7. Parametrizing a quadric and its equation

If a quadric Q is given by its equation $Q(x x) = 0$, and q represents a point of Q , i.e., $Q(q q) = 0$, then

$$x p = Q(p p) q - 2 Q(p q) p$$

is a parametrization of Q , which is quadratic in the coordinates of p .

Setting, e.g., $p = p_0 \zeta_0 + \dots + p_{n-1} \zeta_{n-1}$, where the p_i are suitably chosen, it is also quadratic in the homogeneous ζ_i . Note that one may use any other representation for p , e.g., polar coordinates with the center q .

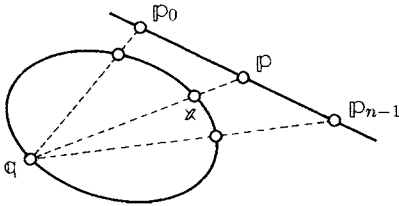


Figure 2.14. Parametrizing a quadric.

Conversely, the equation $Q(x x) = 0$ of a quadric in A^n or P^n depends on $r = (n + 1)(n + 2)/2$ homogeneous coefficients, the elements of C . Let $r - 1$ pairs of conjugate points, p_i and q_i , be given, and let x denote some arbitrary point of Q . Then the $r - 1$ conditions $Q(p_i q_i) = 0$ together with $Q(x x) = 0$ form a homogeneous linear system for the r unknown coefficients of C . Setting its determinant to zero results in the equation of Q .

2.5. DUALITY

In homogeneous or projective coordinates the equation of a hyperplane simplifies to

$$u^t x = u_0 \xi_0 + u_1 \xi_1 + \dots + u_n \xi_n.$$

The u_i are homogeneous coordinates of the hyperplane - as the ξ_i for x . Homogeneous coordinates can either represent a point or a hyperplane. Consequently any configuration of points and hyperplanes has a dual configuration of hyperplanes and points, where the

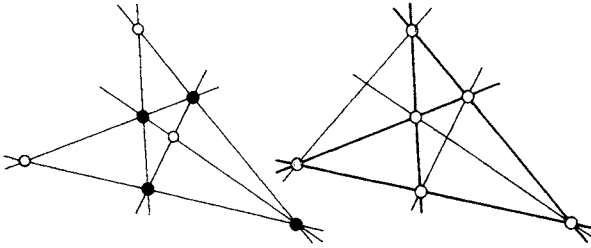


Figure 2.15. Quadrangle and dual quadrilateral.

dual of a point or hyperplane is a hyperplane or point represented by the same coordinates. More general, the hyperplanes meeting some points $\mathbb{b}_1, \dots, \mathbb{b}_d$ are dual to the points of intersection of the hyperplanes $\mathbb{b}_1, \dots, \mathbb{b}_d$, and vice versa.

Note that the duality depends on the dimension of the space. For example, Pascal's and Brianchon's configuration are dual in the plane, where points and tangents of a conic section are dual elements.

2.6. OSCULATING CURVES AND SURFACES

An important task in CAGD is to connect curves and surfaces smoothly.

2.6.1. Curve and surface

A curve $\mathbf{x}(t)$ in affine space \mathbf{A}^n is called regular at t_0 if $\dot{\mathbf{x}}(t_0) \neq \mathbf{o}$.

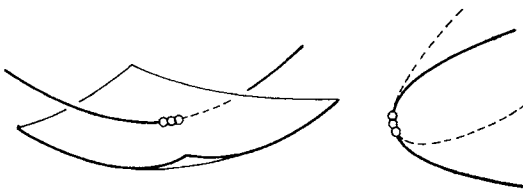


Figure 2.16. Contact of order two.

The curve $\mathbf{x}(t)$ is said to have a contact of order r at t_0 with a surface given by the equation $F(\mathbf{x}) = 0$, if it is regular at t_0 and if $F(\mathbf{x}(t))$ and its derivatives up to order r vanish at $t = t_0$. This means geometrically that the curve has $r + 1$ coalescing points in common with the surface at $t = t_0$. Note that this definition does not depend on the

parametrization of $\mathbf{x}(t)$. Note also that by its geometric meaning the contact of order r is projectively invariant.

2.6.2. Curve and curve

A second curve $\mathbf{y}(s)$ given by the intersection of a number of surfaces contacts $\mathbf{x}(t)$ at t_0 with order r if $\mathbf{x}(t)$ contacts all these surfaces at least with order r .

If the second curve $\mathbf{y}(s)$ is given parametrically, then both curves have contact of order r at t_0 if there exists a regular reparametrization $t = t(s)$, for $\mathbf{x}(t)$ such that the Taylor expansions of $\mathbf{x}(t(s))$ and $\mathbf{y}(s)$ agree at $t_0 = t(s_0)$ up to order r . This condition can be expressed by the chain rule as a system of $r + 1$ linear equations. Therefore contact of order r is referred to as chain rule continuity.

For example, a curve and its osculating circle at a point t_0 have contact of order 2.

2.6.3. Surface and surface

Two surfaces have contact of order r at \mathbf{p} if all regular curves that lie on one of them and meet \mathbf{p} have at least contact of order r with the other surface. This means that after a suitable reparametrization the Taylor expansions of both surfaces at \mathbf{p} agree up to order r .

2.6.4. Contour lines, reflection lines and isophotes

There are some important helpful curves to check the smoothness of surfaces visually.

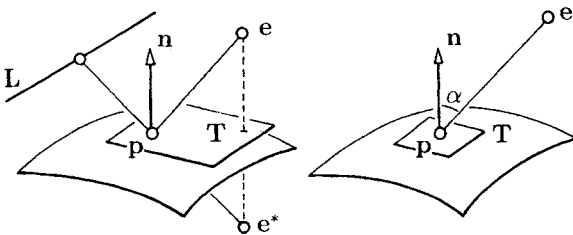


Figure 2.17. Reflection line and isophote.

A reflection line on a surface consists of all points \mathbf{p} whose connection with some fixed point \mathbf{e} , the eye, is reflected into a ray that meets a given fixed line \mathbf{L} .

An isophote on a surface consists of all points \mathbf{p} whose connection with the eye \mathbf{e} includes a fixed angle with the surface normal at \mathbf{p} .

Contour lines are special isophotes. They consist of all points \mathbf{p} , where the tangent plane meets \mathbf{e} .

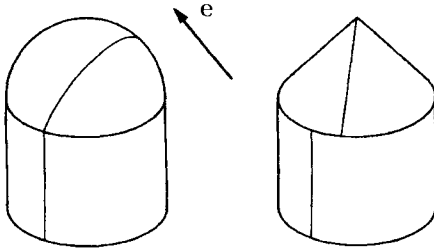


Figure 2.18. Contour lines.

In general, on composite surfaces contour lines, reflection lines and isophotes have a lower order of contact than the surfaces themselves.

Note that all three kinds of curves can decay in parts. Note also that the use of infinite elements \mathbf{e} and \mathbf{L} simplifies their computation.

2.7. DIFFERENTIAL FUNDAMENTALS

Arc length, curvature and torsion describe the local properties of a curve, the curvature of so-called principal normal sections describe the local properties of surfaces. The main tool for such investigations is a local frame.

2.7.1. Arc length and osculating plane

Let a curve in \mathbf{E}^3 be given parametrically as $\mathbf{x} = \mathbf{x}(t)$ and let

$$\mathbf{a}_0 = \mathbf{x}(t_0), \quad \mathbf{v}_1 = \dot{\mathbf{x}}(t_0), \quad \mathbf{v}_2 = \ddot{\mathbf{x}}(t_0),$$

denote its point and first two derivatives at some t_0 . If $\mathbf{v}_1 \neq \mathbf{0}$, its tangent at \mathbf{a}_0 is given by

$$\mathbf{p} = \mathbf{a}_0 + \xi_1 \mathbf{v}_1.$$

If $\mathbf{v}_1 \wedge \mathbf{v}_2 \neq \mathbf{0}$, its osculating plane at \mathbf{a}_0 is given by

$$\mathbf{p} = \mathbf{a}_0 + \xi_1 \mathbf{v}_1 + \xi_2 \mathbf{v}_2.$$

The differential term $ds = \|\dot{\mathbf{x}}(t)\|dt$ is called the arc element of $\mathbf{x}(t)$, and the integral

$$s(t) = \int_{t_0}^t \|\dot{\mathbf{x}}(t)\|dt$$

its arc length, beginning at t_0 . The arc length represents the natural parameter of the curve. In most cases it can only be computed approximately.

2.7.2. Curvature and torsion

The natural parameter s is very helpful to derive general curve properties. E.g., let $\alpha(s)$ and $\beta(s)$ denote the angles of the tangent and the osculating plane at s with the tangent and osculating plane at some s_0 , respectively, and let the prime ' denote differentiation with respect to the arc length. Then

$$\kappa = \alpha'(s_0) \quad \text{and} \quad \tau = \beta'(s_0)$$

are called the curvature and the torsion of $\mathbf{x}(t)$ at s_0 , respectively. Note that $\rho = 1/\kappa$ represents the radius of the osculating circle.

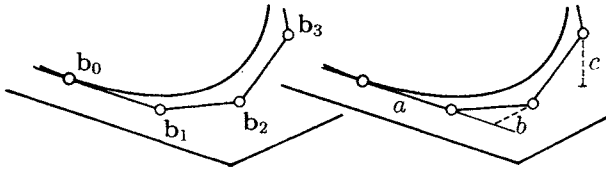


Figure 2.19. Curvature and torsion of a rational Bézier curve.

For example, a rational curve of degree n with Bézier points $\mathbf{b}_i = [\beta_i, \beta_i \mathbf{b}_i^t]^t$ has the span of $\mathbf{b}_0, \mathbf{b}_1$ as tangent at \mathbf{b}_0 , and the span of $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$ as osculating plane at \mathbf{b}_0 . At $t = 0$ one has

$$\kappa(0) = \frac{n-1}{n} \frac{\beta_0 \beta_2}{\beta_1^2} \frac{b}{a^2} \quad \text{and} \quad \tau(0) = \frac{n-2}{n} \frac{\beta_0 \beta_3}{\beta_1 \beta_2} \frac{c}{ab},$$

where a, b, c denote the distances of \mathbf{b}_1 from \mathbf{b}_0 , of \mathbf{b}_2 from the tangent at \mathbf{b}_0 , and of \mathbf{b}_3 from the osculating plane at \mathbf{b}_0 , respectively.

2.7.3. The Frenet frame

Schmidt's orthogonalization applied to $\dot{\mathbf{x}}, \ddot{\mathbf{x}}, \ddot{\mathbf{x}}$ at $\mathbf{x}(t_0)$ results in the so-called Frenet frame $[\mathbf{t} \ \mathbf{m} \ \mathbf{b}]$, which depends on t . One has

$$[\mathbf{t}' \ \mathbf{m}' \ \mathbf{b}'] = [\mathbf{t} \ \mathbf{m} \ \mathbf{b}] \begin{bmatrix} 0 & -\kappa & 0 \\ \kappa & 0 & -\tau \\ 0 & \tau & 0 \end{bmatrix},$$

which is an important tool for further investigations, see [3], [5] and [9].

2.7.4. Curves on surfaces

Let a surface be given parametrically as

$$\mathbf{x} = \mathbf{x}(u, v) = \mathbf{x}(\mathbf{u}).$$

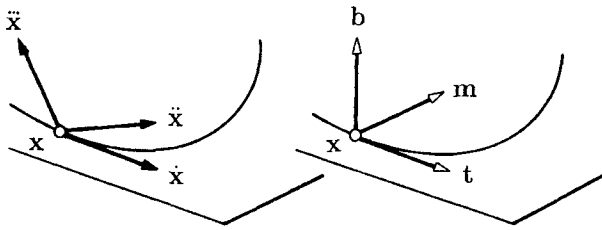


Figure 2.20. Curve with derivatives, osculating plane and Frenet-frame.

The lines $u = \text{fixed}$, $v = \text{fixed}$ are called iso-lines. If the partial derivatives \mathbf{x}_u and \mathbf{x}_v are linearly independent, the surface normal is defined by

$$\mathbf{n} = [\mathbf{x}_u \wedge \mathbf{x}_v] / \|\mathbf{x}_u \wedge \mathbf{x}_v\|.$$

Let $\mathbf{u} = \mathbf{u}(t)$ denote some curve in the \mathbf{u} -plane then, in general, $\mathbf{x} = \mathbf{x}(\mathbf{u}(t))$ represents a curve on the surface. The arc element ds of this curve is given by its square

$$ds^2 = (E \dot{u}^2 + 2F \dot{u} \dot{v} + G \dot{v}^2) dt^2,$$

where $E = \mathbf{x}_u^t \mathbf{x}_u$, $F = \mathbf{x}_u^t \mathbf{x}_v$, and $G = \mathbf{x}_v^t \mathbf{x}_v$ are well-known as Gaussian first fundamental quantities. Note that $\|\mathbf{x}_u \wedge \mathbf{x}_v\|^2 = EG - F^2$.

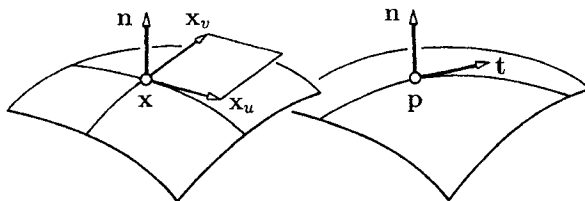


Figure 2.21. Local frames on a surface.

2.7.5. Meusnier's sphere and Dupin's indicatrix

Consider all curves on a surface meeting a given point \mathbf{p} with a given tangent \mathbf{t} there. One has :

The osculating circles of these curves lie on a sphere (*Meusnier's sphere*).

It follows that the radius of the osculating circle of a surface curve is given by $\rho = \rho_0 \cos \delta$, where δ denotes the angle between the surface normal and the osculating plane and ρ_0 is the radius of Meusnier's sphere. The inverse $\kappa_0 = 1/\rho_0$ is called the normal curvature of the surface at \mathbf{p} in direction of \mathbf{t} . Hence it is sufficient to know the curvature of one of these curves and the angle of its osculating plane with \mathbf{n} to compute all others.

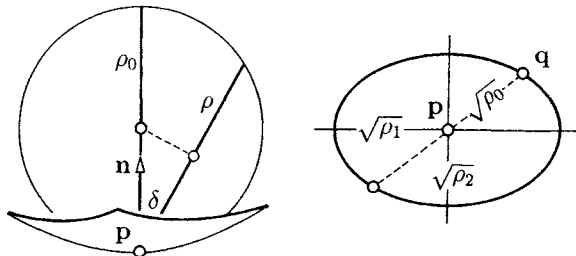


Figure 2.22. Meusnier's sphere and Dupin's indicatrix.

In general, the normal curvature κ_0 differs for different \mathbf{t} . For all tangent directions \mathbf{t} at \mathbf{p} consider the points

$$\mathbf{q} = \mathbf{p} + \sqrt{\rho_0} \mathbf{t}$$

of the tangent plane with distance $\sqrt{\rho_0}$ from \mathbf{p} . One has :

The points \mathbf{q} lie on a conic section with center \mathbf{p} (*Euler's theorem*).

This conic section is also known as Dupin's indicatrix. Its axis directions are called the principal directions, and the corresponding values of $\kappa = 1/\rho_0$ are called the principal curvatures of the surface at \mathbf{p} , mostly denoted by $\kappa_1 = 1/\rho_1$ and $\kappa_2 = 1/\rho_2$.

Note that Dupin's indicatrix can be an ellipse, a pair of parallel lines or a hyperbola. In case of a hyperbola it has two real asymptotic directions . The normal curvature is zero there and ρ_0 is infinite.

If $\kappa_1 = \kappa_2$, then Dupin's indicatrix is a circle and \mathbf{p} is called an umbilical or spherical point.

2.7.6. The curvatures of a surface

Because of its geometric meaning Dupin's indicatrix and consequently the principal curvatures κ_1 and κ_2 at a point \mathbf{p} do not depend on the parametric representation of the surface.

The expressions

$$K = \kappa_1 \kappa_2 \quad \text{and} \quad H = (\kappa_1 + \kappa_2)/2$$

are called the Gaussian curvature and the mean curvature of the surface at \mathbf{p} , respectively. Both give important information about the smoothness of a surface. Moreover, Gauss has shown that K depends on E , F , and G and their derivatives only. This means that K depends on the inner measurements on the surface only and is invariant under deformations of the surface that do not distort the measurement of lengths on the surface.

REFERENCES

1. C. Adler. *Modern Geometry*. McGraw Hill, New York, 1967.
2. M. Berger. *Geometry 1 & 2*. Springer, Berlin, 1987.
3. W. Boehm. Rational geometric splines. *Computer Aided Geometric Design*, 4:67-77, 1987.
4. W. Boehm and H. Prautzsch. *Numerical Methods*. A.K. Peters, Wellesley, 1992.
5. W. Boehm. *Differential Geometry I & II*. in [11], 1993.
6. W. Boehm. An affine representation of de Casteljau's and de Boor's rational algorithms. *Computer Aided Geometric Design*, 10:175-180, 1993.
7. W. Boehm and H. Prautzsch. *Geometric Concepts for Geometric Design*. A.K. Peters, Wellesley, 1994.
8. W. Boehm. Circles of curvature for curves in space. *Computer Aided Geometric Design*, 16:633-638, 1999.
9. M.P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, Englewood NJ, 1976.
10. H.S.M. Coxeter. *Introduction to Geometry*. John Wiley & Son, New York, 1969.
11. G. Farin. *Curves and Surfaces for CAGD: a Practical Guide*, 3rd Edition. Academic Press, Boston, 1993.
12. G. Farin and D. Hansford. *The Geometry Toolbox for Graphics and Modelling*. AK Peters, Natick MA, 1998.
13. G. Farin. *The Essentials of CAGD*. AK Peters, Natick MA, 2000.
14. D. Hilbert and S. Cohn-Vossen. *Geometry and the Imagination*. Chelsea Publishing Co., New York, 1990.
15. L. Ding-yuan. *Computational Geometry - Curve and Surface Modeling*. Academic Press Boston, 1992.
16. H. Prautzsch, W. Boehm, and M. Paluszny. *Bézier and Spline Techniques*. Springer, Berlin/New York, to appear, 2002.
17. C. Wylie. *Introduction to Projective Geometry*. McGraw Hill, New York, 1970.

Chapter 3

Geometries for CAGD

Helmut Pottmann and Stefan Leopoldseder

Chapter 2 describes the fundamental geometric setting for 3D modeling and addresses Euclidean, affine and projective geometry, as well as differential geometry. In the present chapter, the discussions will be continued with a focus on geometric concepts which are less widely known. These are *projective differential geometric methods*, *sphere geometries*, *line geometry*, and *non-Euclidean geometries*. In all cases, we outline and illustrate applications of the respective geometries in geometric modeling.

Special emphasis is put on a general important principle, namely the simplification of a geometric problem by application of an appropriate geometric transformation. For example, we show how to apply curve algorithms for computing with special surfaces such as developable surfaces, canal surfaces and ruled surfaces. As another example, it is shown that an appropriate geometric transformation can map an arbitrary rational surface onto a rational surface all whose offsets are also rational.

For the use of algebraic geometry in geometric design, the reader is referred to Chapter 15 on implicit surfaces. We also skip *difference geometry* [99], which studies discrete counterparts to differential geometric properties and invariants and is thus useful in geometric computing. This holds especially for subdivision curves and surfaces (Chapter 12) and multiresolution techniques (Chapter 14), where discrete models of curves and surfaces play a fundamental role.

Naturally, when describing applications, we reach into many other chapters of this handbook. Thus, our references concerning applications are examples, and partially far from being complete. A much more complete picture is achieved in connection with the references in those chapters we are referring to. The addressed geometric concepts cannot be discussed in sufficient detail within the present frame. For a careful and detailed study of most of the material in this chapter we refer to the monograph by Pottmann and Wallner [94], which focusses on line geometry and its applications in geometric computing. However, it also provides the necessary classical background of related areas such as projective geometry, differential geometry, and algebraic geometry.

3.1. CURVES AND SURFACES IN PROJECTIVE GEOMETRY

Differential geometry in projective spaces requires some modifications over Euclidean differential geometry. In n -dimensional real projective space P^n , a point X is represented by a one-dimensional subspace of \mathbb{R}^{n+1} . Any basis vector $\mathbf{x} = (x_0, \dots, x_n)$ in this subspace delivers the homogeneous coordinates (x_0, \dots, x_n) . The latter are just defined up to a scalar multiple, and thus we write $X = \mathbf{x}\mathbb{R}$. A parameterization of a curve $c \in P^n$ is given in the form

$$c(t) = (x_0(t), \dots, x_n(t)). \quad (3.1)$$

By homogeneity, any function $\lambda(t)c(t)$ with a real scalar-valued function $\lambda(t) \neq 0$ represents the same curve. The transition from the parameterization $c(t)$ to $\lambda(t)c(t)$ is called a *renormalization*. Like a reparameterization, a renormalization does not change the curve as a point set. Analogously, we have to treat parameterizations of m -dimensional surfaces in P^n .

Projective differential geometry is based on properties of curves or surfaces which are invariant under reparameterization, renormalization and projective mappings. It is a very well studied classical subject [10] and turned out to be useful for various applications in geometric modeling [19]. Those include *geometric continuity* and local approximation with the concept of higher order contact (see [19] and Chapters 2 and 8). Other applications, which involve duality, line and sphere geometries, are outlined in the following.

As an example of a concept of projective differential geometry, we mention *osculating spaces*. The osculating space $\Gamma^k(t_0)$ of dimension k at a curve point $c(t_0)\mathbb{R}$ is spanned by this point and the first k derivative points,

$$\Gamma^k(t_0) = c(t_0)\mathbb{R} \vee \dot{c}(t_0)\mathbb{R} \vee \dots \vee c^{(k)}(t_0)\mathbb{R}. \quad (3.2)$$

In case that these points are not linearly dependent, one adds higher derivative points until dimension k of the spanning set is reached. Although the derivative points change both under reparameterization and renormalization, their span does not change, and thus is an example of an invariant object of projective differential geometry.

3.1.1. Bézier curves and surfaces as images of normal curves and surfaces

Rational Bézier curves are fundamental for geometric modeling. It is widely known that rational Bézier curves of degree two are conics. In fact, since polynomial Bézier curves of degree two are just parabolae, the desire to represent all types of conics, quadrics, and other important shapes such as tori exactly in a CAD system, has been one of the motivations for the introduction of the full class of rational curves and surfaces into CAGD.

The most basic algorithm for Bézier curves, de Casteljau's algorithm, is for degree 2 equivalent to Steiner's generation of a conic with help of two projective lines, or more precisely, ranges of points (see [27,28,41]). However, not only quadratic Bézier curves are deeply rooted in projective geometry. The same holds for the full class of rational Bézier curves [18]. The corresponding concept in projective geometry is that of *rational normal curves* [6]. These are rational curves c^n of degree n which span n -dimensional projective space. Their set of osculating hyperplanes is generated by connecting associated points

in n projective ranges of points. For any two different points A, B on a normal curve c^n we may construct the so-called *osculating simplex* or *fundamental simplex* with vertices B_0, \dots, B_n as follows: Point B_i is the intersection of the osculating i -space at A with the osculating $(n-i)$ -space at B . In particular this implies $B_0 = A, B_n = B$. The tangent at A is spanned by B_0 and B_1 , the osculating plane at A is spanned by B_0, B_1, B_2 , and so on. Readers familiar with Bézier curves will immediately recognize the vertices of the osculating simplex as *Bézier points* of the curve segment defined by A and B . In fact, the segment is not yet fully defined, since the normal curve c^n is (like any straight line) a closed curve in P^n . The segment is defined, if one picks an additional curve point F on the two segments defined by A and B . It is common to intersect the osculating hyperplane at F with the lines $B_i \vee B_{i+1}$ and call them *frame points* F_i , $i = 0, \dots, n-1$. It can be shown that a homogeneous parameterization $c^n(t)\mathbb{R}$ of c^n has the form

$$c^n(t) = \sum_{i=0}^n B_i^n(t) \mathbf{b}_i, \quad B_i^n(t) := \binom{n}{i} t^i (1-t)^{n-i}. \quad (3.3)$$

Here, \mathbf{b}_i represent the points B_i , and the homogeneous coordinate vectors \mathbf{b}_i are chosen such that $\mathbf{b}_i + \mathbf{b}_{i+1}$ represents the frame point F_i . The parameter interval for the chosen segment is $[0, 1]$, in particular we have $c^n(0) = \mathbf{b}_0$, $c^n(1) = \mathbf{b}_n$, $c^n(0.5)\mathbb{R} = F$. Also just by intersecting osculating spaces, the so-called *blossom* can be defined and its properties may be seen as special cases of results on normal curves (see Chapter 4).

A projective map in P^n is defined if we know how it acts on the points of a simplex (say B_0, \dots, B_n) and a further point F which is in general position with respect to the points B_i . Thus, representation (3.3) also reveals the remarkable property that any two normal curves, in fact, even *any two segments of normal curves in P^n are projectively equivalent*. This “standard” curve segment has no singularities, inflections, or other degeneracies in the sequence of osculating spaces.

So far we have discussed normal curves, i.e., degree n curves which span P^n . Rational Bézier curves of degree n in lower dimensional spaces P^d ($d < n$) are obtained by applying projections of normal curves into P^d . This is illustrated for the cubic case in Figure 3.1. In fact, there we have an affine special case. A cubic polynomial normal curve c^3 (normal curve with the ideal hyperplane as an osculating hyperplane) with Bézier points B_0, B_1, B_2, B_3' is projected via a parallel projection onto the planar Bézier cubic c with control points B_0, \dots, B_3 . This geometric relation between planar and space cubics can be used for a *shape classification* of cubics in the plane. The questions are: Given B_0, B_1, B_2 , where to choose B_3 such that the curve segment has an inflection, a cusp, a loop, and so on [105]. Since the space cubic is a normal curve, it does not have such characteristics at all. Those are results of the projection and can easily be discussed with help of it (see [80,82], where the shape analysis is extended to rational cubics and also to quartics).

A projective basis for an analogous study of triangular Bézier surfaces are the so-called *Veronese manifolds* [6]. As an example for the application in CAGD, W. Degen [20] discusses the types of Bézier triangles, especially those of degree two. His characterization of quadrics is a basis for further work on quadric patches by G. Albrecht [2].

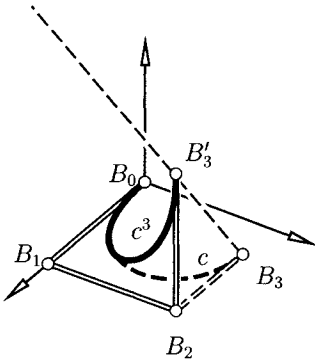


Figure 3.1. Planar cubic Bézier curve via projection of a cubic normal curve

3.1.2. NURBS curves and surfaces in projective geometry

As we have seen, the notion of a *frame point*, which goes back to G. Farin [26], is important for a geometric input of a rational Bézier curve. The so-called *weights* (the homogeneizing coordinates x_0 of the control points, see Chapter 5) have the disadvantage of not being projectively invariant. There is another advantage of frame points. With help of them, we may form a *geometric control polygon* of a rational Bézier or B-spline curve in projective space as follows: On each straight line $B_i B_{i+1}$ connecting consecutive control points take that segment as member of the geometric control polygon, which contains the frame point F_i (see Figure 3.2).

Frame points are tied to the curve in a projectively invariant way: Assume a rational Bézier curve $c(t)$ with geometric control polygon $B_0, F_0, B_1, F_1, \dots, B_n$. A projective transformation $\varkappa : x\mathbb{R} \mapsto (A \cdot x)\mathbb{R}$ maps $c(t)$ to a rational Bézier curve $c'(t)$ whose geometric control polygon is $\varkappa(B_0), \varkappa(F_0), \dots, \varkappa(B_n)$. An analogous property holds for rational B-spline curves.

An advantage of the use of the projective control polygon is that we do not have to confine ourselves to positive weights when formulating the most fundamental shape property, namely the *variation diminishing property*. In the projective setting, it reads as follows: *A hyperplane H intersects a NURBS curve $c(t)$ (not contained in H) in at most as many points as it intersects the geometric control polygon of this curve, if no vertex has zero weight.*

Frame points (also referred to as Farin points) for rational Bézier triangles have been introduced by G. Albrecht [1].

Projective geometry enters many algorithms for rational curves and surfaces, such as reparameterization, degree elevation and shape modification. For those topics, the reader is referred to Chapter 5 of this handbook and to [27,28,41,77] and the references therein.

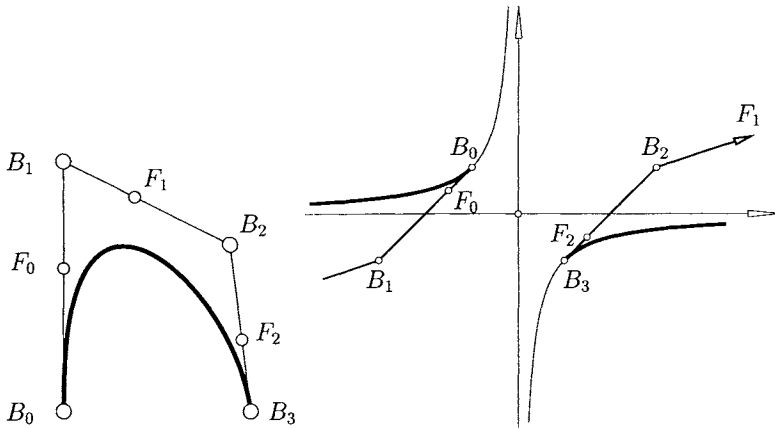


Figure 3.2. Rational Bézier curves with geometric control polygon.

3.1.3. Duality and dual representation

The Bézier representation of a rational curve expresses the polynomial homogeneous parametrization $c(t)\mathbb{R}$ in terms of the Bernstein polynomials. Then the coefficients have the remarkable geometric meaning of control points with a variety of important and practically useful properties.

The *tangent* of a planar rational curve $c(t) = c(t)\mathbb{R}$ at $t = t_0$ is computed as the line which connects $c(t_0)$ with its first derivative point $c^1(t_0) = \dot{c}(t_0)\mathbb{R}$. It has the homogeneous line coordinate vector $\mathbb{R}u(t) = \mathbb{R}(c(t) \wedge \dot{c}(t))$. Thus the family of tangents has again a polynomial parametrization, which can be expressed in the Bernstein basis. This leads to a *dual Bézier curve*

$$U(t) = \mathbb{R}u(t) = \mathbb{R} \left(\sum_{i=0}^m B_i^m(t) u_i \right), \tag{3.4}$$

which can be seen as a family of lines in the (ordinary) projective plane, or as a family of (ordinary) points of its dual plane. For the concept of projective duality, see Section 2.5 of Chapter 2.

The family of tangents of a planar rational Bézier curve is a dual Bézier curve, and vice versa.

When speaking of a Bézier curve we often mean a curve segment. In the form we have written the Bernstein polynomials, the curve segment is parametrized over the interval $[0, 1]$. For any $t \in [0, 1]$, Equation (3.4) yields a line $U(t) = \mathbb{R}u(t)$. The original curve segment is the envelope of the lines $U(t)$, where t ranges in $[0, 1]$.

As an example of dualization, let us discuss the dual control structure of a Bézier curve c (see Figure 3.3): There are the *Bézier lines* $U_i = \mathbb{R}u_i$, $i = 0, \dots, m$, and the *frame lines* F_i , whose line coordinate vectors are given by

$$f_i = u_i + u_{i+1}, \quad i = 0, \dots, m - 1. \tag{3.5}$$

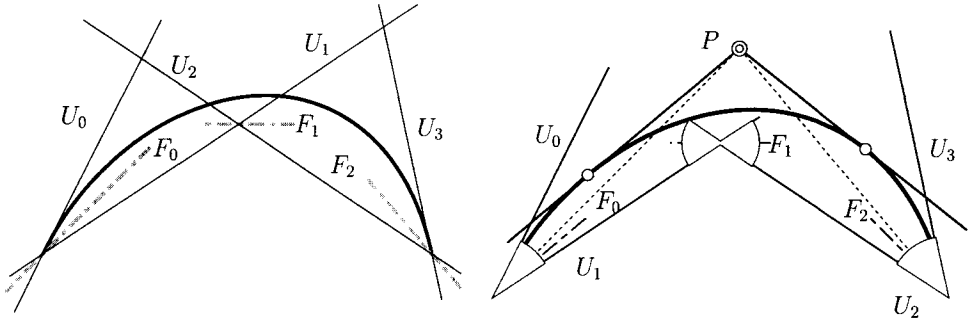


Figure 3.3. Left: Dual Bézier curve. Right: Complete dual control structure and variation diminishing property.

Frame line F_i is concurrent with the Bézier lines U_i and U_{i+1} . This is dual to the collinearity of a frame point with its two adjacent Bézier points.

We could also use weights instead of frame lines, just as we could have used weights instead of frame points. Because weights are no projective invariants, it is preferable to use frame lines and frame points. An invariant statement of theorems is also important for their dualization.

For a Bézier curve, the control points B_0 and B_m are the *end points* of the curve segment, and the lines $B_0 \vee B_1$ and $B_{m-1} \vee B_m$ are the *tangents* there. Dual to this, the *end tangents* of a dual Bézier curve are U_0 and U_m , and their *points of contact* are given by $U_0 \cap U_1$ and $U_{m-1} \cap U_m$, respectively.

We dualize the geometric control polygon: The line pencil spanned by lines U_i and U_{i+1} is divided into two subsets, bounded by U_i and U_{i+1} . The one which contains the frame line is part of the *complete dual control structure* (see Figure 3.3).

Dual to the variation diminishing property of a rational Bézier curve with respect to its projective control polygon we can state the following result: *If c is a planar rational Bézier curve, the number of c 's tangents incident with a given point P does not exceed the number of lines of the complete dual control structure which are incident with P (if no control line has zero weight).*

This result easily implies a sufficient condition for *convexity* of a dual Bézier curve. By a *convex* curve we understand part of the boundary of a convex domain. A support line L of a convex domain \mathcal{D} is a line through a point of the boundary of \mathcal{D} such that \mathcal{D} lies entirely on one side of L . Now the convexity condition reads: *If the Bézier lines U_i and the frame lines F_i of a dual Bézier curve c are among the edges and support lines of a convex domain \mathcal{D} , and the points $U_i \cap U_{i+1}$ are among \mathcal{D} 's vertices, then c is convex and lies completely outside \mathcal{D} .*

A planar rational curve segment, or more precisely, a rational parameterization of it, possesses two Bézier representations: the usual, point-based form, and the dual line-based representation. There are simple formulae for conversion between the two forms (see, e.g., [79,83,94]). However, their behavior when used for design purposes is different. By

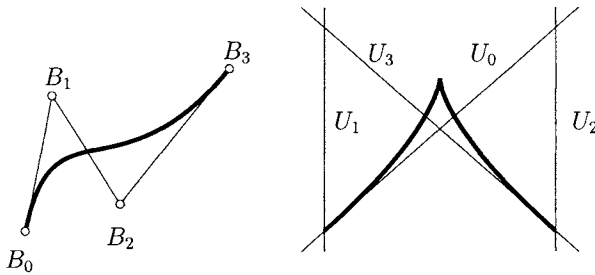


Figure 3.4. Standard control structure tends to generate inflections, whereas the dual control structure tends to introduce cusps.

using the standard representation it is difficult to design cusps but quite easy to achieve inflections of the curve segment. In the dual representation, very special conditions on the control structure must be met to design an inflection, but it is easy to get cusps. This is illustrated by Figure 3.4. For many applications, cusps are not desirable and therefore the convexity condition plays an important role. To achieve inflections, it is best to locate them at end points of the Bézier curve segments (see [79]). Cusps and inflections are dual to each other, but cusps are sometimes easier to detect than inflections. This has been the motivation for J. Hoschek [39,40] to introduce duality and dual Bézier curves and surfaces to CAGD.

The dual representation also provides an advantage in the construction of rational curves and surfaces with rational offsets, which will be outlined below in connection with the use of Laguerre sphere geometry.

3.1.4. Developable surfaces as dual curves

Dualizing the point set of a curve in 3-space, we obtain a family of planes, whose envelope is a *developable surface*. A developable surface is characterized by the property that it can be mapped isometrically into the plane. Because such surfaces can be unfolded into a planar surface without stretching or tearing, they play an important role in various applications, e.g., in sheet-metal and plate-metal based industries.

Dual to the tangents of a curve, a developable surface carries a one-parameter family of lines (rulings), and thus it is a ruled surface. The rulings may pass through a fixed finite or ideal point; this characterizes general *cones* or *cylinder surfaces*, respectively. The rulings may also be the tangents of a space curve c . On such a *tangent surface*, the curve c itself is singular and called *curve of regression*. More general developable surfaces are composed of segments of the mentioned basic types.

It turned out that for the design of developable NURBS surfaces the use of the dual representation has an advantage over treating them as ruled surfaces. This is so, since a ruled surface, represented as a tensor product Bézier or B-spline surface of bidegree $(1, n)$ has to fulfil a very special condition in order to be developable: the tangent plane has to be constant along any of its rulings. This results in a nonlinear system for the control

points, whose general solution is difficult to obtain [3,52].

To construct developable Bézier or general NURBS surfaces, one applies duality in P^3 to Bézier or NURBS curves, respectively. Hence, we obtain a dual control structure consisting of control planes and frame planes, whose major properties follow by duality, just as in the case of dual Bézier curves in the plane. Conversion of a NURBS developable surface from the dual form to its standard representation as a tensor product surface, interpolation and approximation algorithms (see also section 3.4.2 and Figure 3.14), the treatment of singularities, and other topics have been studied [9,13,15,42,43,56,60,83,93,94].

Developable surfaces with creases, e.g. models of crumpled paper, are discussed in [4,48]. The dual representation of developable surfaces also appears in the computation of envelopes [47,113,114]. Finally, even in certain algorithms for non-developable ruled surfaces, the dual representation may have some advantages [44].

3.2. SPHERE GEOMETRIES

In projective geometry the basic geometric elements are points and hyperplanes with incidence as their fundamental relation. Many geometric methods and properties involving (Euclidean) spheres are represented more elegantly, though, if one uses sphere geometries, i.e., spheres by themselves are the basic geometric elements. Classical sphere geometries include Laguerre geometry and Möbius geometry, both of which can be embedded in a larger concept, namely Lie geometry. For a detailed treatment of classical sphere geometries we refer to [5,8,17,12,67]. One of the most recent applications of sphere geometries can be found in biogeometric modeling [24], namely the concept of molecular skin surfaces [23].

3.2.1. Models of Laguerre geometry

The fundamental geometric elements of Laguerre geometry in Euclidean n -space E^n are *oriented hyperplanes* and *oriented hyperspheres*. Let \mathcal{H} denote the set of oriented hyperplanes \mathbf{H} of E^n and \mathcal{C} the set of hyperspheres \mathbf{C} including the points of E^n as (non-oriented) spheres with radius zero. The elements of \mathcal{C} are called *cycles*. The basic relation between oriented hyperplanes and cycles is that of *oriented contact*. An oriented hypersphere is said to be in oriented contact with an oriented hyperplane if they touch each other in a point and their normal vectors in this common point are oriented in the same direction. The oriented contact of a point (nullcycle) and a hyperplane is defined as incidence of point and hyperplane.

Laguerre geometry is the survey of properties that are invariant under the group of so-called *Laguerre transformations* $\alpha = (\alpha_{\mathcal{H}}, \alpha_{\mathcal{C}})$ which are defined by the two bijective maps

$$\alpha_{\mathcal{H}} : \mathcal{H} \rightarrow \mathcal{H}, \alpha_{\mathcal{C}} : \mathcal{C} \rightarrow \mathcal{C}, \quad (3.6)$$

which preserve oriented contact and non-contact between cycles and oriented hyperplanes.

Analytically, a hyperplane \mathbf{H} is determined by the equation $u_0 + u_1x_1 + \dots + u_nx_n = 0$ with normal vector (u_1, \dots, u_n) . The coefficients u_i are homogeneous plane coordinates (u_0, \dots, u_n) of \mathbf{H} in the projective extension P^n of E^n . Each scalar multiple $(\lambda u_0, \dots, \lambda u_n), \lambda \in \mathbb{R} \setminus \{0\}$ describes the same hyperplane. Thus it is possible to use

normalized homogeneous plane coordinates

$$\mathbf{H} = (u_0, \dots, u_n), \text{ with } u_1^2 + \dots + u_n^2 = 1,$$

which are appropriate for describing *oriented* hyperplanes. The unit vector (u_1, \dots, u_n) determines a unit normal and the orientation of the hyperplane.

An oriented hypersphere,

$$\mathbf{C} = (m_1, \dots, m_n; r),$$

is determined by its midpoint $\mathbf{m} = (m_1, \dots, m_n)$ and signed radius r . Positive sign of r indicates that the normal vectors are pointing towards the outside of the hypersphere, whereas in the case of negative sign of r they are pointing into the inside. Points of E^n are cycles characterized by $r = 0$.

The relation of oriented contact is given by

$$u_0 + u_1 m_1 + \dots + u_n m_n + r = 0. \tag{3.7}$$

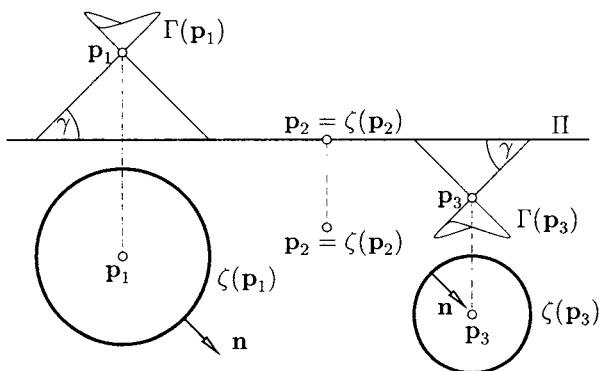


Figure 3.5. Cyclographic mapping, top and front view.

Another model of n -dimensional Euclidean Laguerre space can be constructed in $n + 1$ -dimensional affine space \mathbb{R}^{n+1} , by using the *cyclographic mapping* $\zeta : \mathbb{R}^{n+1} \rightarrow \mathcal{C}$. It maps points $\mathbf{x} = (m_1, \dots, m_n, r)$ to cycles $\mathbf{C} = \zeta(\mathbf{x})$ with midpoint $\mathbf{m} = (m_1, \dots, m_n)$ and oriented radius r . If $\mathbf{x} = (m_1, \dots, m_n, 0)$, $\zeta(\mathbf{x})$ gives the point (nullcycle) \mathbf{m} .

A geometric interpretation of the mapping ζ can be given as follows (see Figure 3.5 for dimension $n = 2$): We assume Euclidean n -space E^n to be embedded as the hyperplane $\Pi : x_{n+1} = 0$ in \mathbb{R}^{n+1} . Let $\Gamma(\mathbf{x})$ denote a hypercone of revolution with vertex \mathbf{x} , whose axis is parallel to the x_{n+1} -axis and whose generators enclose the angle $\gamma = \pi/4$ with the x_{n+1} -axis. Such cones will be called γ -cones, henceforth. Then the cycle $\zeta(\mathbf{x})$ is the intersection of Π with $\Gamma(\mathbf{x})$, where one has to add the correct orientation according to the sign of the $n + 1$ -th coordinate of \mathbf{x} .

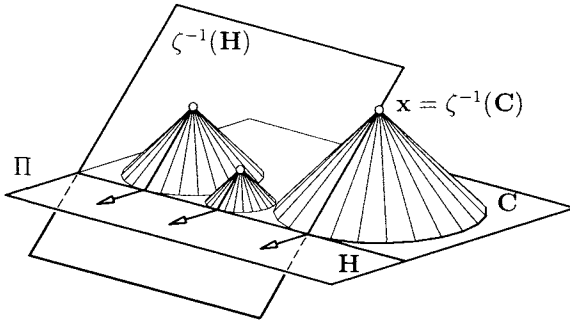


Figure 3.6. Cycles in oriented contact with an oriented line.

Now we focus on oriented contact of cycles and oriented hyperplanes (see Figure 3.6 for $n = 2$). The ζ -preimage of all cycles \mathbf{C} being in oriented contact with a fixed hyperplane $\mathbf{H} = (u_0, \dots, u_n)$ are the points \mathbf{x} of a hyperplane $\zeta^{-1}(\mathbf{H}) : u_0 + u_1x_1 + \dots + u_nx_n + x_{n+1} = 0$, according to (3.7). This hyperplane is incident with \mathbf{H} and encloses an angle of $\gamma = \pi/4$ with Π . It is called a γ -hyperplane. A γ -hyperplane touches the γ -cones $\Gamma(\mathbf{x})$ of its points \mathbf{x} along generators of $\Gamma(\mathbf{x})$, which will be denoted by γ -lines.

We summarize: *The cyclographic mapping ζ maps points of \mathbb{R}^{n+1} to cycles \mathbf{C} of Euclidean Laguerre n -space. Hyperplanes in \mathbb{R}^{n+1} with inclination angle $\pi/4$ to Π correspond to oriented hyperplanes \mathbf{H} . Incidence of point and γ -hyperplane in \mathbb{R}^{n+1} is equivalent to oriented contact of the corresponding cycle and oriented hyperplane.*

In the cyclographic model \mathbb{R}^{n+1} , Laguerre transformations (3.6) appear as transformations of \mathbb{R}^{n+1} which transform γ -lines to γ -lines. This is already sufficient to classify these transformations as special affine maps

$$\mathbb{R}^{n+1} \rightarrow \mathbb{R}^{n+1}, \quad \mathbf{x} \mapsto \lambda \mathbf{A} \cdot \mathbf{x} + \mathbf{c}, \quad \lambda \in \mathbb{R} \setminus \{0\}, \quad \mathbf{A}^T \cdot \mathbf{E}_{pe} \cdot \mathbf{A} = \mathbf{E}_{pe}, \quad (3.8)$$

where $\mathbf{E}_{pe} = \text{diag}(1, \dots, 1, -1)$. Formula (3.8) describes similarities in a *pseudo-Euclidean geometry* (also called *Minkowski geometry*). Its metric is based on the scalar product

$$\langle \mathbf{a}, \mathbf{b} \rangle_{pe} = a_1b_1 + \dots + a_nb_n - a_{n+1}b_{n+1} = \mathbf{a}^T \cdot \mathbf{E}_{pe} \cdot \mathbf{b}. \quad (3.9)$$

Points \mathbf{p} and \mathbf{q} with $\langle \mathbf{p}, \mathbf{q} \rangle_{pe} = 0$ correspond to cycles $\zeta(\mathbf{p}), \zeta(\mathbf{q})$ which are in oriented contact.

Besides the cyclographic model of Euclidean Laguerre space, which represents cycles by points, there are further geometric models, which give a point model for the set \mathcal{H} of oriented hyperplanes.

By dualizing the cyclographic model, γ -hyperplanes (representing the oriented hyperplanes of Euclidean Laguerre n -space) are mapped to points on a quadratic hypercone in \mathbb{R}^{n+1} , the so-called Blaschke hypercone Λ . Points of the cyclographic model (representing cycles) are mapped to hyperplanar intersections of Λ . The *Blaschke model* of Euclidean Laguerre space thus is just the dual of the cyclographic model.

A stereographic projection of the Blaschke cone Λ into a hyperplane \mathbb{R}^n yields the so-called *isotropic model* of Euclidean Laguerre n -space. Oriented hyperplanes $\mathbf{H} \in \mathcal{H}$ are represented by points in \mathbb{R}^n , cycles $\mathbf{C} \in \mathcal{C}$ are given as special quadrics in \mathbb{R}^n , which are spheres with respect to an isotropic metric in \mathbb{R}^n ; cf. section 3.5.3. A detailed discussion of the Blaschke model and the isotropic model, including their analytic treatment and applications to CAGD, can be found in [53,74,87].

3.2.2. Möbius geometry

Let E^n be real Euclidean n -space, \mathcal{P} its point set and \mathcal{M} the set of (non-oriented) hyperspheres and hyperplanes of E^n . We obtain the so-called *Euclidean conformal closure* E_M^n of E^n by extending the point set \mathcal{P} by an arbitrary element (ideal point) $\infty \notin \mathcal{P}$ to $\mathcal{P}_M = \mathcal{P} \cup \{\infty\}$. As an extension of the incidence relation we define that ∞ lies in all hyperplanes but in none of the hyperspheres. The elements of \mathcal{M} are called *Euclidean Möbius hyperspheres*.

Euclidean Möbius geometry is the study of properties that are invariant under *Euclidean Möbius transformations*. A Möbius transformation is a bijective map of \mathcal{P}_M , which maps Möbius hyperspheres to Möbius hyperspheres. A simple example is given by the inversion $\mathbf{x} \mapsto \frac{r^2}{\mathbf{x}^2} \mathbf{x}$ with respect to the sphere $\mathbf{x}^2 = r^2$ in \mathbb{R}^n . Another example is the reflection at a hyperplane, viewed as Möbius sphere. Any general Möbius transformation is a composition of inversions with respect to Möbius spheres.

Besides the standard model of Euclidean Möbius geometry, mentioned above, we obtain the *quadric model* of this geometry by embedding E^n in Euclidean $n + 1$ -space E^{n+1} as plane $x_{n+1} = 0$. Let $\sigma : \Sigma \setminus \{\mathbf{z}\} \rightarrow E^n$ be the stereographic projection of the unit hypersphere

$$\Sigma : x_1^2 + \dots + x_{n+1}^2 = 1 \tag{3.10}$$

onto E^n with projection center (or *north pole*) $\mathbf{z} = (0, \dots, 0, 1)$, see Figure 3.7.

Extending σ to $\bar{\sigma}$ with $\bar{\sigma} : \mathbf{z} \mapsto \infty$ gives the quadric model of Euclidean Möbius geometry which is related to the standard model via $\bar{\sigma}$. The point set is that of $\Sigma \subset E^{n+1}$ and the Möbius spheres are the hyperplanar intersections of Σ since σ is preserving hyperspheres.

For the analytic treatment of Euclidean Möbius geometry, let $\tilde{\mathbf{x}} = (\tilde{x}_1, \dots, \tilde{x}_n)$ denote a point in E^n , and $\mathbf{x} = \sigma^{-1}(\tilde{\mathbf{x}}) = (x_1, \dots, x_{n+1})$ the corresponding point of $\Sigma \subset E^{n+1}$. Let P^{n+1} denote the projective extension of E^{n+1} . In homogeneous coordinates we then have $\mathbf{x}\mathbb{R} = (x_0, x_1, \dots, x_{n+1})\mathbb{R}$ with $-x_0^2 + x_1^2 + \dots + x_{n+1}^2 = 0$ ($\mathbf{x}\mathbb{R} \in \Sigma$). The inverse stereographic projection $\sigma^{-1} : \mathcal{P} \rightarrow \Sigma \setminus \{\mathbf{z}\} \subset E^{n+1}$ is given by

$$\sigma^{-1}(\tilde{\mathbf{x}}) = \mathbf{x}\mathbb{R} = (\tilde{x}_1^2 + \dots + \tilde{x}_n^2 + 1, 2\tilde{x}_1, 2\tilde{x}_2, \dots, 2\tilde{x}_n, \tilde{x}_1^2 + \dots + \tilde{x}_n^2 - 1)\mathbb{R}. \tag{3.11}$$

The homogeneous coordinates $\mathbf{x} = (x_0, x_1, \dots, x_{n+1})$ are called *n-spherical coordinates* of a point $\tilde{\mathbf{x}} \in E^n$. These coordinates are appropriate to represent Möbius spheres as well: Via σ^{-1} a Möbius sphere $\mathbf{M} \in \mathcal{M}$ corresponds to a hyperplanar intersection of Σ , whose pole with respect to Σ shall be denoted by $\mathbf{c}\mathbb{R}$, see Figure 3.7. Its homogeneous coordinates

$$\mathbf{c} = (c_0, c_1, \dots, c_{n+1})$$

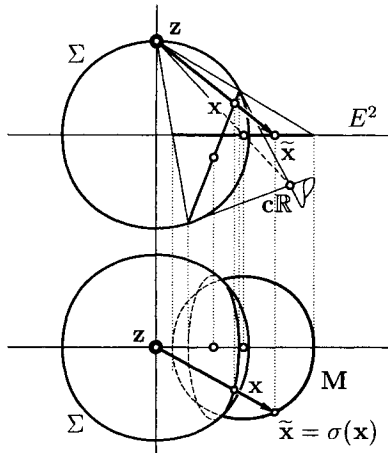


Figure 3.7. Stereographic projection, top and front view.

are called the n -spherical coordinates of \mathbf{M} . For $n = 2, 3$ these coordinates are usually denoted by *tetracyclic* and *pentaspherical coordinates*, respectively.

It can be easily verified that in case of $c_0 = c_{n+1}$ the Möbius sphere \mathbf{M} represents a hyperplane of the standard model with equation $-c_0 + c_1x_1 + \dots + c_{n+1}x_{n+1} = 0$. In case of $c_0 \neq c_{n+1}$ the Möbius sphere \mathbf{M} represents the hypersphere with midpoint $1/(c_0 - c_{n+1}) \cdot (c_1, \dots, c_n)$ and radius $(c_1^2 + \dots + c_n^2 - c_0^2)/(c_0 - c_{n+1})^2$. Let

$$\langle \mathbf{x}, \mathbf{y} \rangle_M = -x_0y_0 + x_1y_1 + \dots + x_{n+1}y_{n+1} = \mathbf{x}^T \cdot \mathbf{E}_M \cdot \mathbf{y}$$

with $\mathbf{E}_M = \text{diag}(-1, 1, \dots, 1)$ describe an indefinite scalar product. Then we are able to describe points by n -spherical coordinates \mathbf{x} with $\langle \mathbf{x}, \mathbf{x} \rangle_M = 0$ and Möbius spheres by n -spherical coordinates \mathbf{c} with $\langle \mathbf{c}, \mathbf{c} \rangle_M > 0$. Incidence of a point $\mathbf{x} \in \mathbb{R}$ and a Möbius sphere $\mathbf{c} \in \mathbb{R}$ is given by $\langle \mathbf{x}, \mathbf{c} \rangle_M = 0$.

It is a central theorem of Euclidean Möbius geometry that in the quadric model all Euclidean Möbius transformations are induced by linear maps $P^{n+1} \rightarrow P^{n+1}, \mathbf{x} \mapsto \mathbf{A} \cdot \mathbf{x}$ with $\mathbf{A}^T \cdot \mathbf{E}_M \cdot \mathbf{A} = \lambda \mathbf{E}_M$, where P^{n+1} again denotes the projective extension of E^{n+1} . These linear maps represent those projective maps of P^{n+1} that keep Σ fixed (as a whole).

3.2.3. Applications of the cyclographic image of a curve in 3-space

With help of the cyclographic mapping, the points of a curve \mathbf{p} in \mathbb{R}^3 are mapped to a family of cycles in the plane E^2 . The envelope of this family of cycles is called the cyclographic image $c(\mathbf{p})$ of the curve. Points of the envelope can be constructed with help of the tangents of \mathbf{p} as shown in Figure 3.8. Note that the orientation of cycles in *planar* Laguerre geometry can be visualized by a counterclockwise (positive) or clockwise (negative) orientation of the corresponding circle.

Consider a Bézier curve \mathbf{p} in \mathbb{R}^3 all of whose control points \mathbf{b}_i are contained in the upper half-space Π^+ which is defined by the equation $x_3 > 0$. The cyclographic image

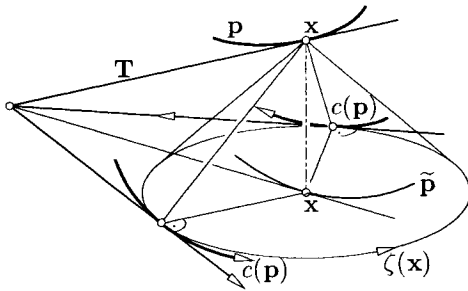


Figure 3.8. Cyclographic image of a curve in \mathbb{R}^3 .

points $\zeta(\mathbf{b}_i)$ are cycles with positive orientation. They determine disks D_i . We can see these disks as *tolerance regions* for imprecisely determined control points in the plane and ask the following question: if the control points vary in their respective tolerance regions D_i , which part of the plane is covered by the corresponding Bézier curves? We call this planar region the *tolerance region of the Bézier curve* (see Figure 3.9). It is not difficult to show that this tolerance region is essentially bounded by the cyclographic image of the Bézier curve $\mathbf{b}(t)$ with control points $\mathbf{b}_0, \dots, \mathbf{b}_n$. An example of this can be seen in Figure 3.9. Such *disk Bézier curves* have been studied by Lin and Rokne [57].

Generalizations to arbitrary convex tolerance regions for the input points are discussed in [35,85,108]. There, other problems of *geometric tolerancing* and error propagation in geometric constructions are addressed as well. Various applications of Laguerre geometry and the cyclographic mapping appear in connection with toleranced circles or spheres.

Further investigations of geometric tolerancing in the plane could make use of very recent work by Farouki et al. [29,30]. It concerns the geometry of sets in the plane, which can be represented in a simple way using complex numbers. Complex numbers are known as elegant tool for certain geometric investigations, for example in planar kinematics and Möbius geometry [102].

3.2.4. The medial axis transform in a sphere geometric approach

Let \mathcal{D} denote a planar domain with boundary $\partial\mathcal{D}$. The (ordinary, trimmed) *medial axis* $\tilde{\mathbf{c}}$ is the locus of centers of maximal disks that are contained in \mathcal{D} ; see Chapter 19 for a detailed discussion on this topic.

The construction of $\tilde{\mathbf{c}}$ allows a Laguerre geometric interpretation, after embedding the plane of \mathcal{D} into \mathbb{R}^3 as $\Pi : x_3 = 0$: We search for a space curve \mathbf{c} whose cyclographic image $\zeta(\mathbf{c})$ is $\partial\mathcal{D}$ (see Figure 3.10 and [37,87]). Let $\partial\mathcal{D}$ be oriented such that its curve normals are pointing outside. Then $\partial\mathcal{D}$ defines a γ -*developable* Γ passing through $\partial\mathcal{D}$, i.e., a developable surface whose generators are γ -lines. The set \mathbf{c} of all self-intersections of Γ is called the (untrimmed) *medial axis transform* of \mathcal{D} . The orthogonal projection of \mathbf{c} onto Π gives the (untrimmed or complete) medial axis $\tilde{\mathbf{c}}$. It is the locus of (oriented) circles that touch the boundary $\partial\mathcal{D}$ in at least two points, but are — because of the lack

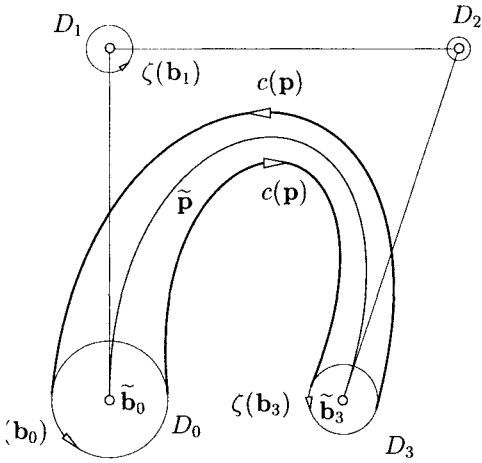


Figure 3.9. Tolerance region of a Bézier curve with disks D_i as tolerance regions for the control points.

of trimming — not necessarily contained in \mathcal{D} .

The above construction of the (untrimmed) medial axis transform allows the computation via surface/surface intersection algorithms, which are discussed in Chapter 25. Those parts of the intersection curve which belong to the *trimmed* medial axis can be easily detected by a visibility algorithm: The interesting part of c lies in the upper half space $x_3 \geq 0$ of \mathbb{R}^3 . If the surface Γ is thought as opaque, exactly the part of c which is visible from below corresponds to the trimmed medial axis.

The medial axis transform c uniquely determines the boundary of the domain \mathcal{D} via the cyclographic image of c . In general, $\partial\mathcal{D}$ will not be rational. The most general class of curves c whose cyclographic images are rational are so-called Minkowski Pythagorean-hodograph (MPH) curves (see Choi et al. [16] and Moon [66]). For a more detailed treatment see Chapters 17 and 19.

3.2.5. Canal surfaces (in Laguerre and Möbius geometry)

A *canal surface* Φ in Euclidean 3-space E^3 is defined as envelope surface of a one parameter family of spheres $\mathbf{S}(t) = (\mathbf{m}(t), r(t))$ (see Figure 3.11).

The sphere family may be written in dependency on the real parameter t ,

$$\mathbf{S}(t) : (\mathbf{x} - \mathbf{m}(t))^2 - r(t)^2 = 0.$$

To compute the envelope, one has to form the derivative with respect to t , which is a plane

$$\dot{\mathbf{S}}(t) : (\mathbf{x} - \mathbf{m}(t)) \cdot \dot{\mathbf{m}}(t) - r(t)\dot{r}(t) = 0.$$

For a parameter t_0 with $\dot{\mathbf{m}}(t_0)^2 - \dot{r}^2(t_0) \geq 0$, the intersecting circle $c(t_0) = \mathbf{S}(t_0) \cap \dot{\mathbf{S}}(t_0)$

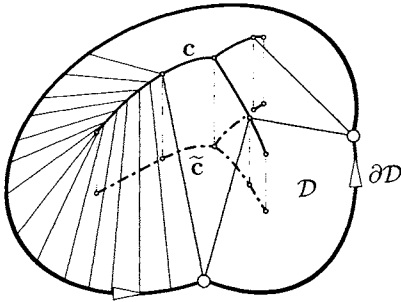


Figure 3.10. Medial axis transform of domain \mathcal{D}

is called *characteristic circle*. Along $c(t_0)$ the sphere $\mathbf{S}(t_0)$ is in smooth contact with the canal surface Φ .

For a Laguerre-geometric interpretation of Φ we allow oriented radii $r(t)$ of $\mathbf{S}(t)$. A canal surface can be obtained as cyclographic image $\zeta(\mathbf{p}(t))$ of the curve $\mathbf{p}(t) = (\mathbf{m}(t), r(t)) \in \mathbb{R}^4$, as described in section 3.2.1. If the tangent line $\mathbf{p}(t_0) + \lambda \dot{\mathbf{p}}(t_0)$ to parameter t_0 encloses an angle $\alpha \geq \pi/4$ with the 3-space $\Pi : x_4 = 0$, the characteristic circle on the corresponding (oriented) sphere $\mathbf{S}(t_0)$ is real.

Besides the Laguerre geometric interpretation of a canal surface as cyclographic image of a space curve, canal surfaces can also be seen from a Möbius geometric point of view: A real canal surface in \mathbb{R}^3 is determined by a curve $c\mathbb{R}(t)$ in the quadric model P^4 whose tangent lines do not intersect the Möbius quadric Σ (a tangent line intersecting Σ can be shown to be equivalent to the corresponding characteristic circle not being real).

We see that from the standpoint of both Laguerre and Möbius geometry, canal surfaces have a representation as curves in 4-dimensional space. Partially by using sphere geometric methods, it could be proved that rationality of these curves implies the existence of a rational parameterization of the corresponding canal surfaces [51,71,73,74].

Thus, these curve models are well suited for design. Approximation and interpolation schemes for curves can be used for approximation or blending schemes with canal surfaces [63,70,87]; see also section 3.4.1.

A very important family of canal surfaces in CAGD are the Dupin cyclides, see Chapter 23. In the cyclographic model of Laguerre geometry they are represented as pseudo-Euclidean circles in \mathbb{R}^4 , i.e., conics that are planar intersections of γ -hypercones. Thus, well-known biarc interpolation schemes can be used to construct G^1 -canal surfaces composed of smoothly joined cyclide patches [87]. Furthermore, the Bézier control points of Dupin cyclide patches and the connection of cyclide patches along cubic or quartic curves can be discussed based on Laguerre geometry [50,64,72].

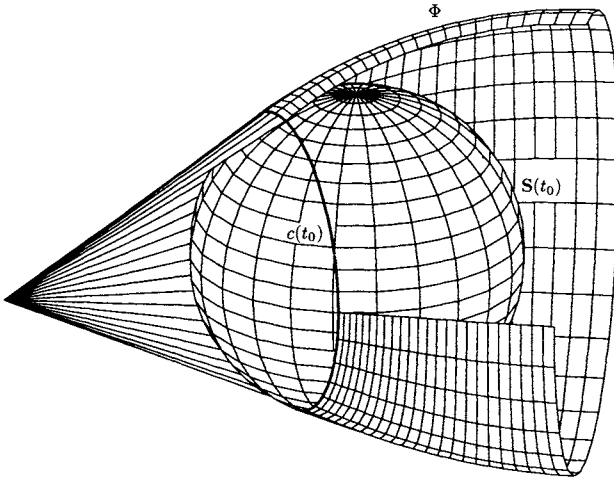


Figure 3.11. Canal surface

3.2.6. Rational curves and surfaces with rational offsets

An *offset* $c_d(t)$ of a given planar curve $c(t)$ lies in constant normal distance d to c . With help of a field of unit normal vectors $\mathbf{n}(t)$ of $c(t)$, the two ‘one-sided’ offsets are $c_d(t) = c(t) + d\mathbf{n}(t)$, where d may have a positive or negative sign. Analogously, we define the offsets of a surface in \mathbb{R}^3 . Offsets possess important applications, for example in NC machining. For the rich literature on this topic, we refer to the survey by T. Maekawa [59].

Given a rationally parameterized curve $c(t)$, the unit normal vectors are in general not rational in t , and thus the offsets of rational curves are in general not rational. However, CAD systems require piecewise rational representations and thus offsets need to be approximated. Another possibility is to use only those rational curves or surfaces which do have rational offsets.

Chapter 17 is exclusively devoted to polynomial and rational Pythagorean-hodograph (PH) curves and gives an extensive overview of the literature on this topic. In the plane, PH curves are polynomial curves whose offsets are rational curves. They can be defined as those polynomial curves whose hodograph $(x'(t), y'(t))$ satisfies the Pythagorean equation $x'^2(t) + y'^2(t) = \sigma^2(t)$ for some polynomial $\sigma(t)$. This property motivates the name ‘PH curve’ and is equivalent to the existence of a polynomial arc length function.

Here we will just skim the surface of the theory of *rational* curves with rational offsets, also referred to as *rational PH curves*. In particular we will stress the close relation of rational PH curves to certain rational developable surfaces via the cyclographic mapping introduced in section 3.2.1.

As outlined in section 3.2.4, an oriented planar curve $\mathbf{p} \subset \Pi$ defines a γ -developable surface Γ passing through \mathbf{p} . The planar intersections of Γ with horizontal planes $x_3 = d$, projected orthogonally onto the plane Π , give the (one-sided) offset \mathbf{p}_d to signed distance

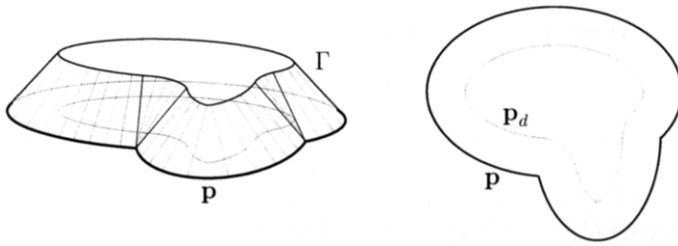


Figure 3.12. Connection between planar intersections of γ -developables to offset curves

d (see Figure 3.12).

Keeping this property in mind, one can classify rational PH curves as planar horizontal intersection curves of *rational* γ -developables Γ . In general, Γ is the tangent surface of a spatial curve c of constant slope $\gamma = \pi/4$, i.e., all of the curve tangents are γ -lines. Obviously Γ is rational if and only if the curve c is. Since c has constant slope $\pi/4$, its third coordinate function $x_3(t)$ equals, up to an additive constant, the total arc length of the top projection $c'(t) = (x_1(t), x_2(t), 0)$. All the offsets of the PH curves share a common evolute, which is the top projection c' of c onto Π . This can be used to obtain the following characterization: *Rational PH curves are exactly the involutes of rational curves with rational arc length function* [78].

Rational γ -developables are easily described in their dual form, and the same holds for rational PH curves. Explicit representations are found in [78].

The description of rational PH curves gets even simpler when one uses the dual Bezier control structure as described in section 3.1.3. A rational PH curve and its offsets have control and frame lines that are related to each other and to a certain dual rational representation of a circle segment by parallel translation. For a detailed treatment of this property, see [78,79,100]. In [79], special rational PH curves, namely cyclographic images of certain conics (studied first by W. Blaschke [7]), have been used to design curvature continuous rational curves with rational offsets.

Whereas the approach to PH curves taken by Farouki and Sakkalis [31] does not have a generalization to surfaces, the dual and Laguerre geometric approach to rational PH curves extend to surfaces [78,74,101]. These *Pythagorean-normal (PN) surfaces* possess rational offset surfaces. A remarkably simple characterization of rational curves and surfaces with rational offsets is within the isotropic model of Laguerre geometry. There, these curves (surfaces), viewed as envelopes of their oriented tangents (tangent planes), appear as arbitrary rational curves or surfaces. *The change between two models of Laguerre geometry transforms an arbitrary rational curve or surface into a rational PH curve or PN surface, respectively* [72,74]. The suitability of Laguerre geometry for studying curves and their offsets is not surprising in view of the fact that the mapping from a curve/surface to an offset of it can be performed with a special Laguerre transformation.

Special polynomial surfaces with rational offsets have been applied to surface design

by Jüttler and Sampoli [46]. The family of PN surfaces includes the following classes of rational surfaces: Regular quadrics [58,74], canal surfaces with rational spine curve $\mathbf{m}(t)$ and rational radius function $r(t)$ [51,71,73,74], and skew rational ruled surfaces [84]. Using Laguerre and Möbius geometry, PN surfaces which generalize Dupin cyclides in the sense that they also possess rational principal curvature lines, have been studied by Pottmann and Wagner [92].

Quadrics, canal surfaces as well as skew ruled surfaces are enveloped by a one parameter set of cones of revolution. Cones of revolution are the cyclographic images of lines in \mathbb{R}^4 which enclose an angle smaller than $\pi/4$ to the embedded 3-space Π . Using this property it is possible to show that any rational one parameter family of cones of revolution envelopes a PN surface [71]. M. Peternell [71] extended this result to other families of quadrics which possess a rationally parametrizable envelope.

Offsets of surfaces are of importance in NC milling [61] when using a *spherical* milling tool. As the milling tool is touching the surface the midpoint of the ball must be located on the offset surface to a distance equaling the radius of the cutting tool. Natural generalizations of offset surfaces occur if the milling tool — which is rotating around its axis — is not a spherical one but a general rotational surface [61,81]. The special case of a *cylindrical* milling tool (flat end mill) yields *circular offset surfaces*. A geometric interpretation via *Galilei sphere geometry* can be found in [107].

3.3. LINE GEOMETRY

Line geometry investigates the set of lines in three-space. There is rich literature on this classical topic of geometry including several monographs [25,36,38,68,94,112,116]. Line geometry possesses a close relation to *spatial kinematics* [11,45,103,106,112], see also Chapter 29. Line geometry enters problems in geometric computing in various ways. A detailed account of the use of line geometry in geometric modeling and related areas is given in a monograph by Pottmann and Wallner [94]. In the following, we briefly outline just a few basic principles and typical applications.

3.3.1. Basics of line geometry

A straight line \mathbf{L} in Euclidean 3-space E^3 can be determined by a point $\mathbf{p} \in \mathbf{L}$ and a normalized direction vector \mathbf{l} of \mathbf{L} , i.e. $\|\mathbf{l}\| = 1$. To obtain coordinates for \mathbf{L} , one forms the moment vector $\bar{\mathbf{l}} := \mathbf{p} \wedge \mathbf{l}$, with respect to the origin. $\bar{\mathbf{l}}$ is independent of the choice of $\mathbf{p} \in \mathbf{L}$. The six coordinates $(\mathbf{l}, \bar{\mathbf{l}})$ with

$$\mathbf{l} = (l_1, l_2, l_3), \quad \text{and} \quad \bar{\mathbf{l}} = (l_4, l_5, l_6)$$

are called *normalized Plücker coordinates* of \mathbf{L} . With normalized \mathbf{l} , the distance of the origin \mathbf{o} to the line \mathbf{L} simply equals $\|\bar{\mathbf{l}}\|$.

However, one may give up the normalization condition and interpret $(l_1, \dots, l_6)\mathbb{R}$ as a point in a 5-dimensional projective space P^5 . Note that \mathbf{l} and $\bar{\mathbf{l}}$ are orthogonal, thus

$$\mathbf{l} \cdot \bar{\mathbf{l}} = l_1 l_4 + l_2 l_5 + l_3 l_6 = 0 \tag{3.12}$$

holds. Equation (3.12) is the so-called *Plücker identity* and describes a hyperquadric M_2^4 in P^5 , the *Klein quadric*. M_2^4 is a four-dimensional manifold and each of its points

$\mathbf{L}\mathbb{R} = (\mathbf{l}, \bar{\mathbf{l}})\mathbb{R}$ with $\mathbf{l} \cdot \bar{\mathbf{l}} = 0$ describes a line \mathbf{L} in the projective extension P^3 of Euclidean 3-space E^3 . Lines \mathbf{L} at infinity are characterized by $\mathbf{l} = \mathbf{o}$.

Summarizing, the use of homogeneous Plücker coordinates for lines in P^3 and their interpretation as points in P^5 results in a point model for line space, which is called *Klein model*. Lines in P^3 correspond to points on the Klein quadric $M_2^4 \subset P^5$.

A line \mathbf{L} may be spanned by two points $\mathbf{x}\mathbb{R}$ and $\mathbf{y}\mathbb{R}$, possibly at infinity. In the following it will turn out convenient to write $\mathbf{x} = (x_0, \mathbf{x})$ with $x_0 \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^3$. Note that here \mathbf{x} does not denote the affine coordinates of $\mathbf{x}\mathbb{R}$, but a scalar multiple of them. The *homogeneous Plücker coordinates* of \mathbf{L} are found as

$$\mathbf{L} = (\mathbf{l}, \bar{\mathbf{l}}) = (x_0\mathbf{y} - y_0\mathbf{x}, \mathbf{x} \wedge \mathbf{y}) \in \mathbb{R}^6.$$

Basic geometric relations with lines, like intersecting a line with a plane or connecting a line with a point result in simple linear equations in homogeneous point, line, and plane coordinates. These formulae can be found in each book on line geometry. As an example we will just mention the intersection condition of two lines $\mathbf{G} = (\mathbf{g}, \bar{\mathbf{g}})\mathbb{R}$ and $\mathbf{H} = (\mathbf{h}, \bar{\mathbf{h}})\mathbb{R}$,

$$\mathbf{g} \cdot \bar{\mathbf{h}} + \bar{\mathbf{g}} \cdot \mathbf{h} = g_1h_4 + g_2h_5 + g_3h_6 + g_4h_1 + g_5h_2 + g_6h_3 = 0. \tag{3.13}$$

It characterizes \mathbf{G}, \mathbf{H} as two conjugate points with respect to the Klein quadric M_2^4 , i.e., they are lying in each others polar hyperplane with respect to M_2^4 .

3.3.2. Linear complexes in kinematics and reverse engineering

A 3-parameter set of lines $\mathbf{L} = (\mathbf{l}, \bar{\mathbf{l}})\mathbb{R}$ satisfying a linear equation in Plücker coordinates,

$$c_1l_4 + c_2l_5 + c_3l_6 + c_4l_1 + c_5l_2 + c_6l_3 = 0, \tag{3.14}$$

is called a *linear line complex* or *linear complex* \mathcal{C} . With $\mathcal{C} = (\mathbf{c}, \bar{\mathbf{c}}) = (c_1, c_2, c_3, c_4, c_5, c_6)$ we can rewrite (3.14) as $\bar{\mathbf{c}} \cdot \mathbf{l} + \mathbf{c} \cdot \bar{\mathbf{l}} = 0$, where $\mathbf{c} \cdot \bar{\mathbf{c}}$ not necessarily equals 0, i.e., $\mathcal{C}\mathbb{R}$ does not need to describe a line.

The connection of linear complexes to kinematics is given as follows. Let us consider a *continuous helical motion*, that is composed of a continuous rotation around a line \mathbf{A} and a continuous translation parallel to \mathbf{A} . In an appropriate coordinate system we have

$$\mathbf{x}(t) = \begin{pmatrix} 0 \\ 0 \\ pt \end{pmatrix} + \begin{pmatrix} \cos t & -\sin t & 0 \\ \sin t & \cos t & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{x}(0). \tag{3.15}$$

In an arbitrary coordinate system the (time independent) velocity vector field for such a motion is $\mathbf{v}(\mathbf{x}) = \bar{\mathbf{c}} + \mathbf{c} \wedge \mathbf{x}$ with constant vectors $\mathbf{c}, \bar{\mathbf{c}}$, see Bottema and Roth [11].

Lines through points \mathbf{x} normal to $\mathbf{v}(\mathbf{x})$ are normal to the trajectory of \mathbf{x} and are called *path normals*, see Figure 3.13. It is easy to show that the path normals \mathbf{L} of a helical motion satisfy $\bar{\mathbf{c}} \cdot \mathbf{l} + \mathbf{c} \cdot \bar{\mathbf{l}} = 0$, thus lie in a linear complex.

If the *pitch* p in (3.15) equals zero, we obtain a pure rotation. The vectors $\mathbf{c}, \bar{\mathbf{c}}$ then will fulfill $\mathbf{c} \cdot \bar{\mathbf{c}} = 0$ and determine the rotational axis \mathbf{A} which is intersected by all of the motion's path normals.

Linear complexes as simple 'linear manifolds' of lines play an important role in various applications. Subsequently, we will address two of them.

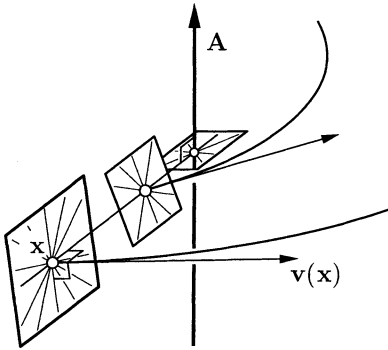


Figure 3.13. The path normals of a helical motion lie in a linear complex.

The first application is in *reverse engineering* of geometric objects (see Chapter 26), where we consider the following problem: Given a cloud of measurement points from a surface, decide whether this cloud can be fitted well by a helical or rotational surface, and if so, construct such an approximating surface.

A helical surface is swept out by a curve which undergoes a continuous helical motion. For vanishing pitch p of the motion, we obtain a rotational surface. It is easy to see that all surface normals of a helical surface lie in the path normal complex of the generating helical motion. Conversely, it can be shown that a surface all whose normals lie in a linear complex must be a helical surface, a rotational surface ($p = 0$) or a general cylinder surface (limit case with $p = \infty$).

Thus, the above reconstruction problem can be solved as follows. We estimate surface normals at the given data points. Those should lie, up to some small deviations, in a linear complex. After defining the deviation of a line L from a linear complex C this leads to an approximation problem in line space. It amounts to a general eigenvalue problem, whose eigenvalues also tell us about the presence of special cases (plane, sphere, right circular cylinder) [89,90].

Another application concerns the stability of a *six-legged parallel manipulator*. There, a moving system Σ is linked to a fixed base system Σ_0 via six legs, realized as hydraulic cylinders, which are attached to both systems via spherical joints. If these six legs (axes of the hydraulic cylinders) lie nearly in a linear complex, the position of the platform Σ gets instable [65,89]. Thus, the determination of instable positions amounts to fitting a linear complex to the axes of the parallel manipulator.

3.3.3. Ruled surfaces

Ruled surfaces are generated by moving a straight line in 3-space. In the Klein model of line space they appear as curves on the Klein quadric M_2^4 [25]. The point model may be advantageous, because for some applications it is easier to deal with curves, even in projective 5-space, than working with ruled surfaces. Approximation and Hermite interpolation algorithms for ruled surfaces amount to corresponding algorithms for curves

on the quadric M_2^4 (see Chapter 31 on quadrics, and [76,94]).

For example, Peternell et al. [76] have formulated algorithms for the approximation of ruled surfaces by low degree algebraic ruled surfaces (ruled quadrics, cubic and quintic ruled surfaces) and have presented a G^1 Hermite interpolation scheme resulting in piecewise quadratic ruled surfaces.

Line geometry applied to CAD has also been considered by Ravani et al. [33,34,96,104], where line geometric counterparts to subdivision algorithms for curves and surfaces, like de Casteljau's algorithm, are developed.

3.3.4. Other applications of line geometry in geometric computing

Line geometry is a basic entity in the formulation of the so-called *generalized stereographic projection* σ , also known as *Hopf mapping*. It maps points in projective 3-space P^3 onto points of the Euclidean sphere S^2 . The preimage of a point on S^2 under this mapping σ is a straight line in P^3 . All fibers of σ form a so-called elliptic linear line congruence in P^3 . It may be seen as intersection of two appropriate linear complexes, and the Klein image of the line congruence is an oval quadric in M_2^4 . Dietz, Hoschek, and Jüttler [22] have shown that the mapping σ is well-suited to construct rational curve and surface patches on the sphere. Applying a projective mapping, one can work on other oval quadrics as well. It can also be used for the definition of a B-spline like intrinsic control structure for NURBS curves on the sphere [80]. There are similar mappings for ruled quadrics and singular quadrics [21], whose fibers are line congruences (intersections of two linear complexes). Such mappings are useful for the design of curves and surface patches on quadrics (see also Chapter 31), and they can also be used to construct rational blending surfaces between quadrics [109].

A generalization of the mapping σ to the construction of rational curves and surface patches on Dupin cyclides has been studied by C. Mäurer [62].

Line geometry also appears in manufacturing, such as sculptured surface machining [91,111] and wire cut EDM [96]. For further applications and detailed discussions, we refer the reader to Pottmann and Wallner [94].

3.4. APPROXIMATION IN SPACES OF GEOMETRIC OBJECTS

For different geometric objects in E^3 there exist point models: Oriented spheres can be represented as points in the cyclographic model, see section 3.2.1. Planes are represented as points in dual projective space, see section 3.1.3. Lines are represented as points on a hyperquadric M_2^4 in P^5 , see section 3.3.1.

Approximation schemes in the spaces of spheres, planes, lines or other geometric objects require a point model and an appropriate distance defined for these geometric objects. After mapping the point model to an affine space one will define an appropriate Euclidean metric, which is motivated by a deviation measure between two objects. Here we will briefly mention the deviation measures in the spaces of spheres, planes and lines, and resulting approximation schemes for canal surfaces (section 3.4.1), developable surfaces (section 3.4.2) and ruled surfaces (section 3.4.3). For details, see Pottmann and Peternell [75,88].

3.4.1. Approximation in the space of spheres

In the cyclographic model of 3-dimensional Euclidean Laguerre geometry, oriented spheres \mathbf{S} are seen as points $\zeta^{-1}(\mathbf{S}) = (m_1, m_2, m_3, r)$. The distance of two oriented spheres $\mathbf{A} : (a_1, \dots, a_4)$ and $\mathbf{B} : (b_1, \dots, b_4)$ can be defined via the canonical Euclidean distance of their image points in \mathbb{R}^4 ,

$$d(\mathbf{A}, \mathbf{B})^2 = \sum_{i=1}^4 (a_i - b_i)^2. \quad (3.16)$$

A geometric interpretation of $d(\mathbf{A}, \mathbf{B})$ can be found in [88]. With help of the above metric in \mathbb{R}^4 one can use standard Bézier and B-spline techniques for curve design in \mathbb{R}^4 , and one obtains rational canal surfaces as the cyclographic images of the designed curves. Here, the geometric continuity (Chapter 8) is preserved: a G^k curve gives rise to a G^k canal surface.

3.4.2. Approximation in the space of planes

The set of planes in P^3 is a 3-dimensional projective space itself. The homogeneous coordinates $\mathbf{U} = (u_0, u_1, u_2, u_3)$ of a plane \mathbf{U} are the coefficients of the plane's equation $u_0 + u_1x + u_2y + u_3z = 0$, see section 3.1.3. If we work in Euclidean 3-space and restrict ourselves to planes which are not parallel to the z -axis of a Cartesian system, i.e., $u_3 \neq 0$, we can normalize the plane coordinates to $\mathbf{U} = (u_0, u_1, u_2, -1)$ and obtain affine coordinates $(u_0, u_1, u_2) \in A^3$ of \mathbf{U} . Note that one may choose an appropriate coordinate system to avoid that planes of interest are parallel to the z -axis.

The distance of two planes \mathbf{A}, \mathbf{B} within some region of interest may be defined by

$$d_\Gamma(\mathbf{A}, \mathbf{B})^2 = \int_\Gamma ((a_0 - b_0) + (a_1 - b_1)x + (a_2 - b_2)y)^2 dx dy.$$

which equals the squared z -differences of \mathbf{A} and \mathbf{B} , integrated over a fixed domain Γ of interest in the xy -plane, see Figure 3.14. The such defined d_Γ is a positive definite quadratic form in $a_i - b_i$, whose constant coefficients are certain integrals that can be easily computed. Thus, d_Γ introduces a Euclidean metric in affine 3-space A^3 .

One parameter sets of planes envelop developable surfaces which correspond to curves in A^3 . Again, standard Bézier and B-spline approximation techniques can be used, e.g., to approximate a discrete set of tangent planes with a NURBS developable surface, see Figure 3.14. Details and the important task of controlling the singularities are discussed in [42,93,94].

3.4.3. Approximation in line space

Consider two parallel planes Π_0, Π_1 in \mathbb{R}^3 and the set \mathcal{L}^0 of all lines which are not parallel to them. Then intersection of any line in \mathcal{L}^0 with Π_0, Π_1 gives a pair $\mathbf{x}_0 = (l_1, l_2)$, $\mathbf{x}_1 = (l_3, l_4)$ of points, which may be considered as point $\mathbf{L} = (l_1, l_2, l_3, l_4)$ in real affine 4-space \mathbb{R}^4 . This mapping from \mathcal{L}^0 onto \mathbb{R}^4 can be interpreted as stereographic projection of the Klein quadric M_2^4 .

The affine image space can be equipped with the Euclidean metric

$$d(\mathbf{A}, \mathbf{B})^2 = \sum_{i=1}^4 (a_i - b_i)^2 + (a_1 - b_1)(a_3 - b_3) + (a_2 - b_2)(a_4 - b_4).$$

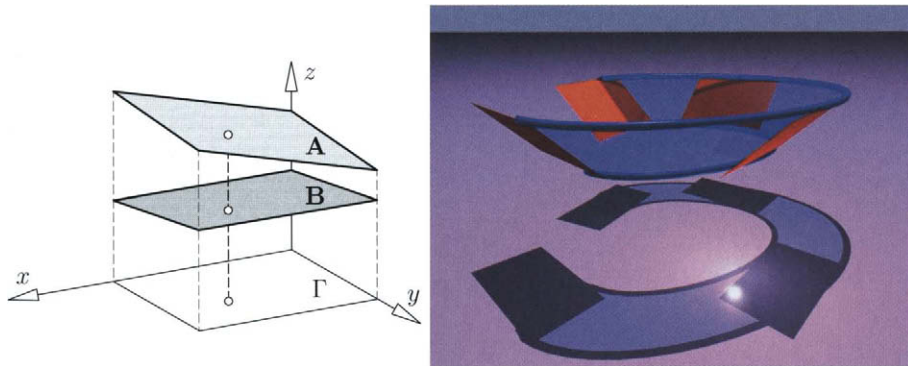


Figure 3.14. Left: To the definition of the deviation of two planes: Right: Developable surface approximating four planes.

It corresponds to a *distance of the two lines A, B* within the parallel strip bounded by planes Π_0, Π_1 (region of interest). It is obtained by integrating the squared distances between the lines, measured horizontally, see Figure 3.15.

The Euclidean metric $d(\mathbf{A}, \mathbf{B})$ defined above is useful for solving various approximation problems in line space [14,94]. It has also been used to compute the approximation of given lines L_i by a ruled surface in Figure 3.15, see [88] for details.

3.5. NON-EUCLIDEAN GEOMETRIES

3.5.1. Hyperbolic geometry and geometric topology

Although we are usually designing in Euclidean space, there are various examples for applications of non-Euclidean geometries in geometric modeling.

A remarkable application is the following. Consider the *hyperbolic plane* H^2 , a model of which can be realized as follows. Take a circular disk with bounding circle u . The points in the open disk are the points of the hyperbolic plane. Collinear points in hyperbolic geometry lie on circles (or straight lines) which intersect u orthogonally. Such hyperbolic straight line segments are seen in Figure 3.16, left. Hyperbolic congruences are seen in this special model as Möbius transformations which preserve u as a whole.

There are other models of the hyperbolic plane, which are more appropriate for computations. One of these is the projective model, where points and lines appear as points and line segments inside a circle u and congruence transformations are given by projective maps which preserve u as a whole.

In the hyperbolic plane, there exist remarkable discrete groups \mathcal{G} of congruences. They possess a domain \mathcal{F} bounded by $4g$ -gon (g being an integer ≥ 2) as fundamental domain. This means that application of the elements of the group \mathcal{G} to \mathcal{F} generates a tiling of the hyperbolic plane. Figure 3.16, left, shows such a tiling for $g = 2$. It illustrates a slightly more complicated fundamental domain, which is, however, equivalent to an octagon as

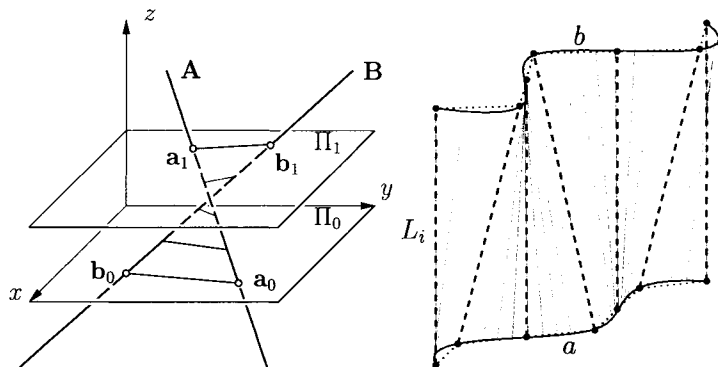


Figure 3.15. Left: Distance between lines measured horizontally; Right: Ruled surface approximating seven (dashed) lines L_i .

the group in the sense that its value $f(\mathbf{x})$ at a point $\mathbf{x} \in H^2$ and at all images of \mathbf{x} under the elements of \mathcal{G} are the same. Then, three such functions, evaluated at the fundamental domain \mathcal{F} , may be seen as coordinate functions of a parametric surface in 3-space. It is well-known that this surface is a closed orientable surface of *genus* g and that all closed orientable surfaces of *genus* $g \geq 2$ may be obtained via hyperbolic geometry in this way [95,115].

This hyperbolic approach to the design of closed surfaces of arbitrary *genus* and smoothness has first been taken by Ferguson and Rockwood [32]. [110] have further investigated this direction and shown, for example, how to design piecewise rational surfaces with arbitrarily high geometric continuity. Although theoretically very elegant, the practical use for complicated shapes seems to be limited. Most likely, subdivision based schemes will be preferred for applications.

3.5.2. Elliptic geometry and kinematics

The intrinsic geometry of the n -dimensional Euclidean sphere $S^n \subset E^{n+1}$, with identification of antipodal points, is called *elliptic geometry*. Three-dimensional elliptic geometry is very closely related to *spherical kinematics* and has important applications in the design and analysis of motions on the sphere and in Euclidean 3-space [69]. This relation as well as applications in computer animation and robot motion planning are discussed in Chapter 29.

3.5.3. Isotropic geometry and analysis of functions and images

In order to visualize the function $f : \mathcal{D} \subset \mathbb{R}^2 \rightarrow \mathbb{R}$, defined on a region \mathcal{D} of the Euclidean plane $E^2 = \mathbb{R}^2$, we usually embed this plane as (x_1, x_2) -plane into 3-space \mathbb{R}^3 and consider the *graph surface* $\Gamma(f) := \{(x_1, x_2, f(x_1, x_2)) \in \mathbb{R}^3 : (x_1, x_2) \in \mathcal{D}\}$. This natural procedure is sometimes followed by the seemingly natural assumption to interpret \mathbb{R}^3 as Euclidean space. However, it is much more appropriate for many applications to introduce a so-



Figure 3.16. Tesselation of the hyperbolic plane (left); a function which is invariant under the associated discrete group is suitable for parametrizing a closed orientable surface of genus two (right).

space. However, it is much more appropriate for many applications to introduce a so-called *isotropic metric* in \mathbb{R}^3 . In *isotropic geometry*, one investigates properties which are invariant under the following group of affine mappings,

$$\begin{aligned} x'_1 &= a_1 + x_1 \cos \varphi - x_2 \sin \varphi, \\ x'_2 &= a_2 + x_1 \sin \varphi + x_2 \cos \varphi, \\ x'_3 &= a_3 + a_4 x_1 + a_5 x_2 + x_3. \end{aligned} \tag{3.17}$$

Like the Euclidean motion group in \mathbb{R}^3 , this group of so-called isotropic motions depends on six real parameters φ, a_1, \dots, a_5 . As seen from the first two equations in (3.17), an isotropic motion appears as Euclidean motion in the projection onto the plane $x_3 = 0$. A careful study of isotropic geometry in two and three dimensions is found in the monographs by H. Sachs [97,98].

The application to the analysis and visualization of functions defined on Euclidean spaces is studied in [86]. For example, the standard thin plate spline functional in two dimensions,

$$J(f) := \int \left(\left(\frac{\partial^2 f}{\partial x_1^2} \right)^2 + 2 \left(\frac{\partial^2 f}{\partial x_1 \partial x_2} \right)^2 + \left(\frac{\partial^2 f}{\partial x_2^2} \right)^2 \right) dx,$$

has a purely geometric interpretation for the graph surface of f within isotropic geometry. It is the surface integral over the sum of squares of isotropic principal curvatures \varkappa_1, \varkappa_2 ,

$$J(f) = \int (\varkappa_1^2 + \varkappa_2^2) dx.$$

The use of isotropic geometry has been extended to functions defined on surfaces (Chapter 9) rather than flat Euclidean spaces [86]. Currently, it is investigated by J. Koenderink for understanding images of surfaces along the lines described in [49].

Isotropic geometry also appears in the context of Laguerre geometry, namely in the so-called isotropic model. For example, the oriented tangent planes of a right circular cone appear as an *isotropic circle* in the isotropic model. This is in general a conic, whose projection onto $x_3 = 0$ is a Euclidean circle. Smooth spline curves formed by such conic segments could be called “isotropic arc splines”. Their construction is completely analogous to arc splines in Euclidean 3-space. The transformation back to the standard model of Laguerre geometry gives developable surfaces, which consist of smoothly joined pieces of right circular cones [55]. Geometric computing with these *cone spline surfaces* rather than general developables has a variety of advantages: The computation of bending sequences and the planar development can be performed in an elementary way. The degree, namely two for both the implicit and parametric representation of the segments, is the lowest possible for generating smooth surfaces, and the offsets are of the same type [54,56].

REFERENCES

1. G. Albrecht. A note on Farin points for rational triangular Bézier patches. *Computer Aided Geometric Design*, 12:507–512, 1995.
2. G. Albrecht. *Rational Triangular Bézier Surfaces – Theory and Applications*. Shaker Verlag, Aachen, 1999.
3. G. Aumann. Interpolation with developable Bézier patches. *Computer Aided Geometric Design*, 8:409–420, 1991.
4. M. Ben Amar and Y. Pomeau. Crumpled paper. *Proc. Royal Soc. London*, A453:729–755, 1997.
5. W. Benz. *Geometrische Transformationen*. BI-Wiss. Verlag, Mannheim, 1992.
6. E. Bertini. *Einführung in die projektive Geometrie mehrdimensionaler Räume*. 2nd ed., Seidel & Sohn, Wien, 1924.
7. W. Blaschke. Untersuchungen über die Geometrie der Speere in der Euklidischen Ebene. *Monatshefte für Mathematik und Physik*, 21:3–60, 1910.
8. W. Blaschke. *Vorlesungen über Differentialgeometrie*. Vol. III. Springer, Berlin, 1929.
9. R.M.C. Bodduluri and B. Ravani. Design of developable surfaces using duality between plane and point geometries. *Computer-Aided Design*, 25:621–632, 1993.
10. G. Bol. *Projektive Differentialgeometrie I, II, III*. Vandenhoeck & Ruprecht, Göttingen, 1950, 1954, 1967.
11. O. Bottema and B. Roth. *Theoretical Kinematics*. Dover Publ., New York, 1990.
12. T.E. Cecil. *Lie Sphere Geometry*. Springer, Berlin, Heidelberg, New York, 1992.
13. J. Chalfant and T. Maekawa. Design for manufacturing using B-spline developable surfaces. *J. Ship Research*, 42:207–215, 1998.
14. H.Y. Chen and H. Pottmann. Approximation by ruled surfaces. *J. Comput. Applied Math.* 102:143–156, 1999.
15. H.Y. Chen, I.K. Lee, S. Leopoldseder, and H. Pottmann, T. Randrup, and J. Wallner. On surface approximation using developable surfaces. *Graphical Models and Image*

- Processing*, 61:110–124, 1999.
16. H.I. Choi, C.Y. Han, H.P. Moon, K.H. Roh, and N.S. Wee. Medial axis transform and offset curves by Minkowski Pythagorean hodograph curves. *Computer-Aided Design*, 31:59–72, 1999.
 17. J.L. Coolidge. *A Treatise on the Circle and the Sphere*. Clarendon Press, Oxford, 1916.
 18. W.L.F. Degen. Some remarks on Bézier curves. *Computer Aided Geometric Design*, 5:259–268, 1988.
 19. W.L.F. Degen. Projektive differential geometrie. In O. Giering and J. Hoschek, editors, *Geometrie und ihre Anwendungen*, Hanser, München, pages 319–374, 1994.
 20. W.L.F. Degen. The types of triangular Bézier surfaces. In G. Mullineux, editor, *The Mathematics of Surfaces VI*, Clarendon Press, Oxford, pages 153–170, 1996.
 21. R. Dietz. *Rationale Bézier-Kurven und Bézier-Flächenstücke auf Quadriken*. Diss., TH Darmstadt, 1995.
 22. R. Dietz, J. Hoschek, and B. Jüttler. An algebraic approach to curves and surfaces on the sphere and on other quadrics. *Computer Aided Geometric Design*, 10:211–229, 1993.
 23. H. Edelsbrunner. Deformable smooth surface design. *Discrete Comput. Geom.*, 21:87–115, 1999.
 24. H. Edelsbrunner. *Bio-Geometric Modeling*. Lecture Notes, Duke University, 2001.
 25. W.L. Edge. *The Theory of Ruled Surfaces*. Cambridge University Press, Cambridge, UK, 1931.
 26. G. Farin. Algorithms for rational Bézier curves. *Computer-Aided Design*, 15:73–77, 1983.
 27. G. Farin. *NURBS for Rational Curve and Surface Design*. AK Peters, Boston, 1994.
 28. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Fourth ed., Academic Press, San Diego, 1997.
 29. R.T. Farouki, H.P. Moon, and B. Ravani. Algorithms for Minkowski products and implicitly defined complex sets. *Advances in Comp. Math.*, 13:199–229, 2000.
 30. R.T. Farouki, H.P. Moon, and B. Ravani. Minkowski geometric algebra of complex sets. *Geometriae Dedicata*, 85:283–315, 2001.
 31. R.T. Farouki and T. Sakkalis. Pythagorean hodographs. *IBM J. Res. Develop.*, 34:736–752, 1990.
 32. H. Ferguson and A. Rockwood. Multiperiodic functions for surface design. *Computer Aided Geometric Design*, 10:315–328, 1993.
 33. Q.J. Ge and B. Ravani. On representation and interpolation of line segments for computer aided geometric design. *ASME Design Automation Conf.*, 96(1):191–198, 1994.
 34. Q.J. Ge and B. Ravani. Geometric design of rational Bézier line congruences and ruled surfaces using line geometry. *Computing*, 13:101–120, 1998.
 35. C.U. Hinze. *A Contribution to Optimal Tolerancing in 2-Dimensional Computer Aided Design*. Dissertation, Kepler University Linz, 1994.
 36. V. Hlavaty. *Differential Line Geometry*. P. Nordhoff Ltd., Groningen, 1953.
 37. C. Hoffmann. Computer vision, descriptive geometry and classical mechanics. In B. Falcidieno et al., editors, *Computer Graphics and Mathematics*, Springer Verlag,

- pages 229–243, Berlin, 1992.
38. J. Hoschek. Liniengeometrie. Bibliograph. Institut, Zürich, 1971.
 39. J. Hoschek. Dual Bézier curves and surfaces. In R.E. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, North Holland, pages 147–156, 1983.
 40. J. Hoschek. Detecting regions with undesirable curvature. *Computer Aided Geometric Design*, 1:183–192, 1984.
 41. J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. A.K. Peters, Wellesley, MA, 1993.
 42. J. Hoschek and H. Pottmann. Interpolation and approximation with developable B-spline surfaces. In M. Daehlen, T. Lyche, and L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, Vanderbilt University Press, Nashville (TN), pages 255–264, 1995.
 43. J. Hoschek and M. Schneider. Interpolation and approximation with developable surfaces. In A. Le Méhauté, C. Rabut, and L.L. Schumaker, editors, *Curves and Surfaces with Applications in CAGD*, Vanderbilt University Press, Nashville (TN), pages 185–202, 1997.
 44. J. Hoschek and U. Schwanecke. Interpolation and approximation with ruled surfaces. In R. Cripps, editor, *The Mathematics of Surfaces VIII*, Information Geometers, Winchester, pages 213–231, 1998.
 45. K.H. Hunt. *Kinematic Geometry of Mechanisms*. Clarendon Press, Oxford, 1978.
 46. B. Jüttler and M.L. Sampoli. Hermite interpolation by piecewise polynomial surfaces with rational offsets. *Computer Aided Geometric Design*, 17, 2000.
 47. B. Jüttler and M. Wagner. Rational motion-based surface generation. *Computer-Aided Design*, 31:203–213, 1999.
 48. Y.L. Kergosien, H. Gotoda, and T.L. Kunii. Bending and creasing virtual paper. *IEEE Computer Graphics & Applications*, 14:40–48, 1994.
 49. J. Koenderink, A. van Doorn, and A. Kappers. Surfaces in the mind’s eye. In R. Cipolla and R. Martin, editors, *The Mathematics of Surfaces IX*, Springer, pages 180–193, London, 2000.
 50. R. Krasauskas and C. Mäurer. Studying cyclides with Laguerre geometry. *Computer Aided Geometric Design*, 17:101–126, 2000.
 51. G. Landsmann, J. Schicho, F. Winkler, and E. Hillgarter. Symbolic parametrization of pipe and canal surfaces. *Proceedings ISSAC-2000*, ACM Press, pages 194–200, 2000.
 52. J. Lang and O. Röschel. Developable $(1, n)$ Bézier surfaces. *Computer Aided Geometric Design*, 9:291–298, 1992.
 53. S. Leopoldseder. *Cone Spline Surfaces and Spatial Arc Splines*. Dissertation, Vienna Univ. of Technology, 1998.
 54. S. Leopoldseder. Algorithm on cone spline surfaces and spatial osculating arc splines. *Computer Aided Geometric Design*, 18:505–530, 2001.
 55. S. Leopoldseder. Cone spline surfaces and spatial arc splines – a sphere geometric approach. *Advances in Computational Mathematics*, to appear, 2001.
 56. S. Leopoldseder and H. Pottmann. Approximation of developable surfaces with cone spline surfaces. *Computer-Aided Design*, 30:571–582, 1998.
 57. Q. Lin and J. Rokne. Disk Bézier curves. *Computer Aided Geometric Design*, 15:721–

- 737, 1998.
58. W. Lü. Rational parameterization of quadrics and their offsets. *Computing*, 57:135–147, 1996.
 59. T. Maekawa. An overview of offset curves and surfaces. *Computer-Aided Design*, 31:165–173, 1999.
 60. T. Maekawa and J.S. Chalfant. Computation of inflection lines and geodesics on developable surfaces. *Math. Engineering in Industry*, 7:251–267, 1998.
 61. K. Marciniak. *Geometric Modelling for Numerically Controlled Machining*. Oxford University Press, New York, 1991.
 62. C. Mäurer. *Rationale Bézier-Kurven und Bézier-Flächenstücke auf Dupinschen Zykloiden*. Diss., Darmstadt, 1997.
 63. C. Mäurer. Applications of sphere geometry in canal surface design. In P.J. Laurent et al., editors, *Curve and Surface Design: Saint Malo 1999*, Vanderbilt Univ. Press, Nashville, pages 267–276, 1999.
 64. C. Mäurer and R. Krasauskas. Joining cyclide patches along quartic boundary curves. In M. Dæhlen et al., editors, *Mathematical Methods for Curves and Surfaces II*, Vanderbilt Univ. Press, Nashville, pages 359–366, 1998.
 65. J.-P. Merlet. Singular configurations of parallel manipulators and grassmann geometry. *Int. J. Robotics Research*, 8:45–56, 1992.
 66. H.P. Moon. Minkowski Pythagorean hodographs. *Computer Aided Geometric Design*, 16:739–753, 1999.
 67. E. Müller and J. Krames. *Vorlesungen über Darstellende Geometrie II: Die Zyklographie*. Deuticke, Leipzig, Wien, 1929.
 68. E. Müller and J. Krames. *Vorlesungen über Darstellende Geometrie III: Konstruktive Behandlung der Regelflächen*. Deuticke, Leipzig, Wien, 1931.
 69. H.R. Müller. Sphärische Kinematik. *VEB Deutscher Verlag der Wissenschaften*, Berlin, 1962.
 70. M. Paluszny and W. Boehm. General cyclides. *Computer Aided Geometric Design*, 15:699–710, 1998.
 71. M. Peternell. *Rational Parameterizations for Envelopes of Quadric Families*. Dissertation, Vienna Univ. of Technology, 1997.
 72. M. Peternell and H. Pottmann. Designing rational surfaces with rational offsets. In F. Fontanella, K. Jetter, and P.J. Laurent, editors, *Advanced Topics in Multivariate Approximation*, World Scientific, pages 275–286, 1996.
 73. M. Peternell and H. Pottmann. Computing rational parametrizations of canal surfaces. *J. Symbolic Computation*, 23:255–266, 1997.
 74. M. Peternell and H. Pottmann. A Laguerre geometric approach to rational offsets. *Computer Aided Geometric Design*, 15:223–249, 1998.
 75. M. Peternell and H. Pottmann. Approximation in the space of planes – Applications to geometric modeling and reverse engineering. Technical Report 87, Institut für Geometrie, TU Wien, 2001.
 76. M. Peternell, H. Pottmann, and B. Ravani. On the computational geometry of ruled surfaces. *Computer-Aided Design*, 31:17–32, 1999.
 77. L. Piegl and W. Tiller. *The NURBS Book*. Springer-Verlag, New York, 1995.
 78. H. Pottmann. Rational curves and surfaces with rational offsets. *Computer Aided*

- Geometric Design*, 12:175–192, 1995.
79. H. Pottmann. Curve design with rational Pythagorean-hodograph curves. *Advances in Comp. Math.*, 3:147–170, 1995.
 80. H. Pottmann. Studying NURBS curves and surfaces with classical geometry. In M. Dæhlen, T. Lyche, and L.L. Schumaker, editors, *Mathematical Methods for Curve and Surface Design*, Vanderbilt Univ. Press, Nashville, pages 413–438, 1995.
 81. H. Pottmann. General offset surfaces. *Neural, Parallel and Scientific Computations*, 5:55–80, 1997.
 82. H. Pottmann and T.D. DeRose. Classification using normal curves. In *Curves and Surfaces in Computer Vision and Graphics II*, SPIE, 1610:217–228, 1991.
 83. H. Pottmann and G. Farin. Developable rational Bezier and B-spline surfaces. *Computer Aided Geometric Design*, 12:513–531, 1995.
 84. H. Pottmann, W. Lü, and B. Ravani. Rational ruled surfaces and their offsets. *Graphical Models and Image Processing*, 58:544–552, 1996.
 85. H. Pottmann, B. Odehnal, M. Peternell, J. Wallner, and R. Ait Haddou. On optimal tolerancing in computer-aided design. Proceedings *Geometric Modeling and Processing 2000*, IEEE, pages 347–363, Los Alamitos, CA.
 86. H. Pottmann and K. Opitz. Curvature analysis and visualization for functions defined on Euclidean spaces or surfaces. *Computer Aided Geometric Design*, 11:655–674, 1994.
 87. H. Pottmann and M. Peternell. Applications of laguerre geometry in CAGD. *Computer Aided Geometric Design*, 15:165–186, 1998.
 88. H. Pottmann and M. Peternell. On approximation in spaces of geometric objects. In R. Cipolla and R. Martin, editors, *The Mathematics of Surfaces IX*, Springer, pages 438–458, London, 2000.
 89. H. Pottmann, M. Peternell, and B. Ravani. An introduction to line geometry with applications. *Computer-Aided Design*, 31:3–16, 1999.
 90. H. Pottmann and T. Randrup. Rotational and helical surface reconstruction for reverse engineering. *Computing*, 60:307–322, 1998.
 91. H. Pottmann and B. Ravani. Singularities of motions constrained by contacting surfaces. *Mechanism and Machine Theory*, 35:963–984, 2000.
 92. H. Pottmann and M. Wagner. Principal surfaces. In T. Goodman and R. Martin, editors, *The Mathematics of Surfaces VII*, Information Geometers, Winchester, pages 337–362, 1997.
 93. H. Pottmann and J. Wallner. Approximation algorithms for developable surfaces. *Computer Aided Geometric Design*, 16:539–556, 1999.
 94. H. Pottmann and J. Wallner. *Computational Line Geometry*. Springer, Heidelberg, 2001.
 95. J.G. Ratcliffe. *Foundations of Hyperbolic Manifolds*. Graduate Text in Mathematics, 149, Springer, 1994.
 96. B. Ravani and J.W. Wang. Computer aided design of line constructs. *J. Mech. Des.*, 113:363–371, 1991.
 97. H. Sachs. *Ebene isotrope Geometrie*. Vieweg, Braunschweig/Wiesbaden, 1987.
 98. H. Sachs. *Isotrope Geometrie des Raumes*. Vieweg, Braunschweig/Wiesbaden, 1990.
 99. R. Sauer. *Differenzengeometrie*. Springer, Berlin/Heidelberg, 1970.

100. R. Schickentanz. Interpolating G^1 splines with rational offsets. *Computer Aided Geometric Design*, 14:881–885, 1997.
101. R. Schickentanz. *Interpolation und Approximation durch B-Spline-Flächen mit rationalen Offsets*. Diss., TU Darmstadt, 1999.
102. H. Schwerdtfeger. *Geometry of Complex Numbers*, Dover, New York, 1979.
103. J.M. Selig. *Geometrical Methods in Robotics*. Springer, New York, 1996.
104. K. Sprott and B. Ravani. Ruled surfaces, Lie groups and mesh generation. *Proc. ASME Design Eng. Techn. Conf. 1997*, No. DETC97/DAC-3966.
105. M.C. Stone and T.D. DeRose. A geometric characterization of cubic curves. *ACM Trans. Graphics*, 8:147–163, 1989.
106. E. Study. *Geometrie der Dynamen*. Leipzig, 1903.
107. J. Wallner, H.Y. Chen, and H. Pottmann. Galilei Laguerre geometry and rational circular offset surfaces. *Contributions to Algebra and Geometry*, 39:291–305, 1998.
108. J. Wallner, R. Krasauskas, and H. Pottmann. Error propagation in geometric constructions. *Computer-Aided Design*, 32:631–641, 2000.
109. J. Wallner and H. Pottmann. Rational blending surfaces between quadrics. *Computer Aided Geometric Design*, 14:407–419, 1997.
110. J. Wallner and H. Pottmann. Spline orbifolds. In A. Le Mé hauté, C. Rabut, and L.L. Schumaker, editors, *Curves and Surfaces with Applications in CAGD*, Vanderbilt University Press, Nashville (TN), pages 445–464, 1997.
111. J. Wallner and H. Pottmann. On the geometry of sculptured surface machining. In P.J. Laurent, P. Sablonnière, and L.L. Schumaker, editors, *Curve and Surface Design: Saint Malo 1999*, Vanderbilt University Press, Nashville (TN), pages 417–432, 2000.
112. E.A. Weiss. *Einführung in die Liniengeometrie und Kinematik*. Teubner, Leipzig/Berlin, 1935.
113. J. Xia and Q.J. Ge. On the exact representation of the boundary of the swept volume of a cylinder undergoing rational Bézier and B-spline motions. *Proceedings ASME Design Eng. Techn. Conf.*, Las Vegas, 1999.
114. J. Xia and Q.J. Ge. On the exact computation of the swept surface of a cylindrical surface undergoing two-parameter rational Bézier motions. *Proceedings ASME Design Automation Conference*, 2000.
115. H. Zieschang, E. Vogt, and H.D. Coldewey. *Surfaces and Planar Discontinuous Groups*. Lecture Notes in Mathematics, 835, Springer, 1980.
116. K. Zindler. *Liniengeometrie mit Anwendungen*. 2 volumes. De Gruyter, Berlin 1902, 1906.

Chapter 4

Bézier Techniques

Dianne Hansford

This chapter introduces the fundamentals of Bézier techniques. As a core tool of 3D Modeling, Bézier techniques provide a geometric-based method for describing and manipulating polynomial curves and surfaces.

4.1. WHY BÉZIER TECHNIQUES?

Bézier techniques bring sophisticated mathematical concepts into a highly geometric and intuitive form. From a practical standpoint, this form facilitates the creative design process. Equally as important, Bézier techniques are an excellent choice in the context of numerical stability of floating point operations.¹ For these reasons, Bézier techniques are at the core of 3D Modeling or Computer Aided Geometric Design (CAGD).

This chapter provides a thorough review of fundamental Bézier techniques. The primary topics being curves, rectangular surfaces, and triangular surfaces. With this knowledge, the reader should be able to access research articles on these topics. Additionally, the study of Bézier techniques is greatly recommended before studying piecewise schemes, B-splines, or other advanced modeling applications.

The notation for this chapter was chosen to make this subject accessible to readers new to 3D modeling, and also to best serve those who use this handbook as a reference. Quite a few 3D Modeling texts (see e.g., [43,56,91]) have adopted the *blossom*² notation, and although important, this notation has a tendency to abstract the geometric concepts, and is therefore not used here.

For in-depth information on Bézier techniques, a textbook is a good place to start. There are many to choose from: [17,43,45,69,80,81,86,91,102]. The origins of Bézier techniques may be found in the History chapter 1 of this handbook. Industrial uses of Bézier

¹Surprisingly, this result from Farouki and Rajan [49] was not known until many years after Bézier techniques had become popular. See also [28].

²The blossom approach is a very powerful tool, theoretically and practically, and it was brought to light by de Casteljau [32] and Ramshaw [88,89]. For a tutorial, see Goldman and DeRose [36].

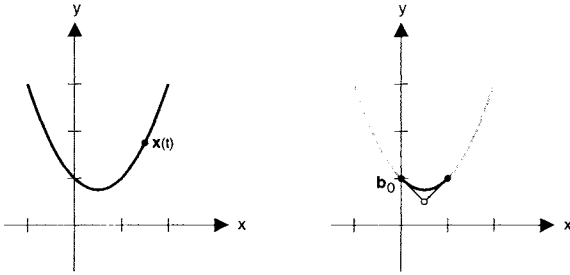


Figure 4.1. Left: The graph of a function. Right: The function as a Bézier curve.

techniques are described by Bézier [7] at Renault, de Casteljaou [33] at Citroën, Farin [40] at Daimler-Benz, and Hochfeld and Ahlers [65] at Volkswagen.

4.2. BÉZIER CURVES

In this section, we introduce the foundations of Bézier techniques via the Bézier curve. In particular, we present the properties and utility of Bézier curves, as well as an important evaluation algorithm: the de Casteljaou algorithm. Also, we study the building block of Bézier techniques, the Bernstein polynomials. A thorough understanding of Bézier curves is a good place to start, since nearly all the principles of curves carry over to Bézier surface techniques.

4.2.1. Parametric curves

Curve modeling is primarily concerned with *parametric curves*. The simple quadratic function in the left of Figure 4.1, written as a 2D parametric curve, takes the form

$$\mathbf{x}(t) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} t \\ 1 - t + t^2 \end{bmatrix}, \quad t \in \mathbb{R}. \quad (4.1)$$

Each coordinate is a function of the *parameter* t , and the real line is the *domain* of the curve. Only polynomial coordinate functions will be addressed here. A 3D parametric curve is formed by simply adding a $z(t)$ -component.

The boldface notation for points and vectors³ allows for a concise form for (4.1):

$$\mathbf{x}(t) = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2, \quad (4.2)$$

where

$$\mathbf{a}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \mathbf{a}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \mathbf{a}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The \mathbf{a}_i are called the *coefficients* of the curve and $1, t, t^2$ are the *quadratic monomial basis functions*.

³See the Geometric Fundamentals Chapter 2 for an introduction to these geometric entities.

There are many ways to represent a polynomial curve. The monomial form from above is one representation, however, it does not provide the most geometrically intuitive interpretation.⁴ A better formulation comes with the Bernstein basis functions as the building block for Bézier curves. Figure 4.1, right, illustrates the curve in quadratic Bézier form. This *quadratic Bézier curve* takes the form

$$\mathbf{x}(t) = \mathbf{b}_0 B_0^2(t) + \mathbf{b}_1 B_1^2(t) + \mathbf{b}_2 B_2^2(t), \quad (4.3)$$

where

$$\mathbf{b}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mathbf{b}_1 = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad (4.4)$$

are *Bézier control points* of the *Bézier polygon*, and

$$B_0^2(t) = (1-t)^2, \quad B_1^2(t) = 2t(1-t), \quad B_2^2(t) = t^2$$

are the *quadratic Bernstein polynomials* or *basis functions*. The standard procedure is to evaluate Bézier curves for $t \in [0, 1]$, although since it is a polynomial, it is defined for all t over the reals. The reason for this will be apparent in Section 4.2.2.

A degree n Bézier curve takes the form

$$\mathbf{x}(t) = \sum_{i=0}^n \mathbf{b}_i B_i^n(t) \quad t \in [0, 1], \quad (4.5)$$

where

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i, \quad (4.6)$$

are the *degree n Bernstein polynomials*, and the *binomial coefficients* are defined as

$$\binom{n}{i} = \frac{n!}{(n-i)!i!}.$$

Figure 4.2 illustrates several Bézier curves.

4.2.2. Properties of Bézier curves

The following list of properties characterizes Bézier curves. We'll revisit many of these properties in Sections 4.2.3 and 4.2.4. Many of these properties are apparent in Figure 4.2.

- *Endpoint interpolation*: The curve passes through the polygon endpoints: $\mathbf{x}(0) = \mathbf{b}_0$ and $\mathbf{x}(1) = \mathbf{b}_n$.
- *Symmetry*: The two polygons, $\mathbf{b}_0, \dots, \mathbf{b}_n$ and $\mathbf{b}_n, \dots, \mathbf{b}_0$, describe the same curve; the only thing that changes is the direction of traversal of the parameter.

⁴See Section 4.2.8 for the geometric interpretation of the monomial coefficients.

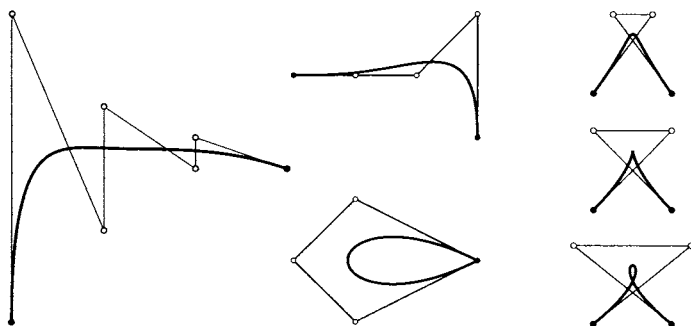


Figure 4.2. These Bézier curves reveal much about the relationship between the polygon and curve.

- *Affine invariance:* If an affine map Φ is applied to the control polygon, then the curve is mapped by the same map. More precisely,

$$\sum_{i=0}^n \Phi(\mathbf{b}_i) B_i^n(t) = \Phi\left(\sum_{i=0}^n \mathbf{b}_i B_i^n(t)\right) \quad (4.7)$$

- *Convex hull:* A point $\mathbf{x}(t)$ on the curve for $t \in [0, 1]$ is in the convex hull of the control polygon. See Figure 4.3.
- *Variation diminishing:* If a straight line intersects a planar Bézier polygon m times, then the line can intersect the curve at most m times. In other words, the curve doesn't wiggle more than the polygon. This is evident in the left most curve of Figure 4.2.
- *Linear precision:* If the control points \mathbf{b}_i for $i = 1, \dots, n - 1$ are evenly spaced on the straight line between \mathbf{b}_0 and \mathbf{b}_n , then the degree n Bézier curve is the linear interpolant between \mathbf{b}_0 and \mathbf{b}_n . See Figure 4.4.
- *Extrapolation:* For values of t outside $[0, 1]$, the curve will in general not remain within the control polygon's convex hull. See Figure 4.5 for an illustration. Numerical stability issues [49,50] and unpredictable behavior make this an undependable tool in a practical setting.
- *Special geometry:* On the right of Figure 4.2, three cubic Bézier curves are illustrated; From top to bottom we have one with two inflection points, one with a cusp, and one with a loop and therefore self-intersects.
- *Functional curves:* The quadratic curve defined by (4.3) and (4.4) is a functional curve, and thus one dimension is a linear polynomial. The linear precision property

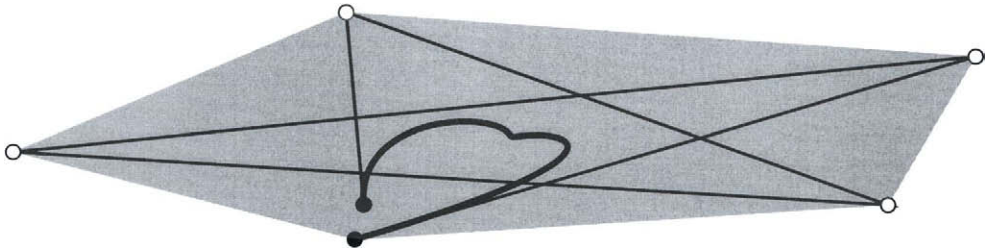


Figure 4.3. Convex hull property: The convex hull of the polygon is shaded. For $t \in [0, 1]$, the point $\mathbf{x}(t)$ lies in the convex hull of the Bézier polygon.

of Bézier curves dictates that a functional curve defined for $t \in [0, 1]$ takes the form

$$\mathbf{x}(t) = \sum_{i=0}^n \begin{bmatrix} i/n \\ b_i \end{bmatrix} B_i^n(t),$$

and an example is illustrated in the right of Figure 4.1; Also see Figure 4.4.

- *Pseudo-local control*: Suppose we move the i^{th} control point. The curve changes the most in the vicinity of $t = i/n$. In fact, all points on the curve move in a direction parallel to the vector formed by the difference of the old and new control point, as illustrated in Figure 4.6.
- *Invariance under affine parameter transformations*: Particularly in the context of piecewise curves, it might be necessary to associate a parameter interval $u \in [a, b]$ with a Bézier curve. The parameter interval does not effect the shape of the Bézier curve. It is common practice to associate the *global parameter* u with the *local parameter* $t \in [0, 1]$ via the simple transformation $t = (u - a)/(b - a)$.

Cubic Bézier curves are perhaps utilized more than any other degree. This is primarily due to the fact that their shape is flexible “enough,” while at the same time somewhat predictable because the degree is rather low.⁵ Complex shapes are typically modeled with *piecewise polynomials* as discussed in the B-Spline Basics Chapter 6.

Other interesting properties of Bézier curves are explored in [4,11,26,34,44,55], effective algorithms are analyzed in [50,85], and applications which take advantage of the Bézier curve form are found in [14,23,68].

⁵There is a historical reason too: They are another representation of cubic Hermite curves, which have been used for many years. See Section 4.2.8 for more on the Hermite form.

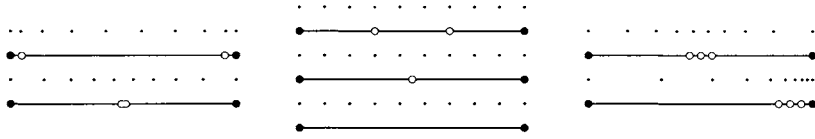


Figure 4.4. Linear precision property: For each curve, ten points with uniformly distributed parameters are plotted over the polygon. Middle column: Bézier curves with linear precision.

4.2.3. The de Casteljau algorithm for Bézier curves

The de Casteljau algorithm provides a means for evaluating Bézier curves, but it also provides for greater understanding of Bézier methods as a whole. Further insight into this important algorithm may be found in Boehm and Müller [16].

The de Casteljau algorithm for the evaluation of a degree n Bézier curve takes the following form.

de Casteljau Algorithm

Given: Bézier points \mathbf{b}_i for $i = 0, \dots, n$, and parameter $t \in [0, 1]$.

Find: The point $\mathbf{b}_0^n(t)$ on the curve.

Compute: Set $\mathbf{b}_i^0 = \mathbf{b}_i$ and compute the points

$$\mathbf{b}_i^r(t) = (1 - t)\mathbf{b}_i^{r-1} + t\mathbf{b}_{i+1}^{r-1} \quad \begin{cases} r = 1, \dots, n \\ i = 0, \dots, n - r. \end{cases} \tag{4.8}$$

Figure 4.7 illustrates the algorithm for a cubic at $t = 1/4$. Notice that this recursive algorithm simply consists of repeated linear interpolation. Each step builds a new point from two other points in the ratio $t : (1 - t)$.

A convenient schematic tool for describing the algorithm is to arrange the involved points in a triangular diagram. For example, evaluation of a cubic curve results in the following points.

$$\begin{matrix} \mathbf{b}_0 \\ \mathbf{b}_1 & \mathbf{b}_0^1 \\ \mathbf{b}_2 & \mathbf{b}_1^1 & \mathbf{b}_0^2 \\ \mathbf{b}_3 & \mathbf{b}_2^1 & \mathbf{b}_1^2 & \mathbf{b}_0^3 \end{matrix} \tag{4.9}$$

An interesting point to note is that each of the intermediate points $\mathbf{b}_i^r(t)$ in the de Casteljau algorithm is actually a point on a degree r curve. In Figure 4.8, a degree five curve illustrates this point for which $\mathbf{b}_0^r(t)$ is plotted for $r = 2, 3, 4, 5$.

Examining Figure 4.7, observe two special polygons:

$$\mathbf{b}_0, \mathbf{b}_0^1, \mathbf{b}_0^2, \mathbf{b}_0^3 \text{ and } \mathbf{b}_0^3, \mathbf{b}_1^2, \mathbf{b}_2^1, \mathbf{b}_3.$$



Figure 4.5. Extrapolation property: The curve is plotted for values of $t \in [-1.0, 2.3]$.

Each of these polygons defines the two segments of the curve corresponding to $[0, t]$ and $[t, 1]$ with respect to the original curve. Redefining a curve in this manner is called *subdivision*. In the schematic triangular diagram (4.9), the control points for these curves are along the diagonal and the base of the triangle, or more specifically the “left” and “right” control points are

$$\mathbf{l}_i = \mathbf{b}_0^i \quad \text{and} \quad \mathbf{r}_i = \mathbf{b}_i^{n-i}.$$

Subdivision must not necessarily take place within $[0, 1]$, although this is extrapolation. The foundations of subdivision are based on the work of de Casteljau [31] and Staerk [101]. Additional information may be found in [43,57,59,60]. Schwartz [95] develops formulas based on a *shift operator* technique.⁶

Subdivision may be repeated: Each of the two new control polygons may be subdivided, and so on. The resulting sequence of control polygons will ultimately converge to the curve. This result is explored by Cohen and Schumaker [24] and Dahmen [27]. Convergence is fast, and thus repeated subdivision could be used to render a curve. See Lane and Riesenfeld [75] and Bartels *et. al.* [6]. Another application of repeated subdivision is the intersection of a 2D Bézier curve with a line. Figure 4.9 illustrates the basic idea, which is to repeatedly subdivide at $t = 1/2$ and check the intersection of the control polygon’s minmax box and the line. When the line intersects a minmax box whose dimension is less than some input tolerance, then an intersection is recorded at the midpoint of the minmax box.

In Section 4.2.5 a practical method of computing derivatives of a Bézier curve via the de Casteljau algorithm is discussed. Generally, Bézier curves are evaluated via the de Casteljau algorithm rather than computing the Bernstein polynomials directly.

4.2.4. Bernstein polynomials

Let us take a closer look at the *Bernstein polynomials* from (4.6). This study will give insight into the appealing properties of Bézier curves. Bernstein polynomials’ relation to Bézier curves was discovered by Forrest [54], however also see Bézier [8] and de Casteljau [30].

A geometric approach to studying Bernstein polynomials is realized by formulating them as functional Bézier curves. To form the j^{th} basis function, recall from Section 4.2.2

⁶This technique, first introduced for Bézier techniques by Hosaka and Kimura [67] offers a concise notation. It is used to some extent in Hoschek and Lasser [69].

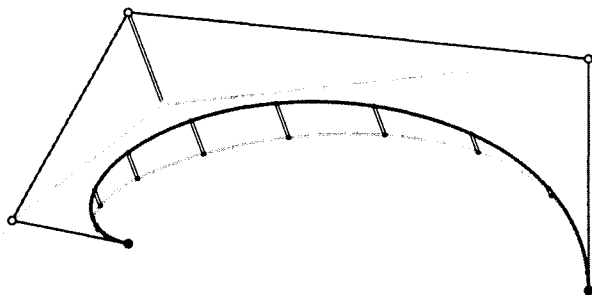


Figure 4.6. Pseudo-local control property: Moving the middle control point causes all points on the curve to move in the same direction. A double line indicates the change in a particular point.

that we assign

$$\mathbf{b}_i = \begin{bmatrix} i/n \\ b_i \end{bmatrix} \quad \begin{cases} b_i = 1 & \text{if } i = j \\ b_i = 0 & \text{if } i \neq j. \end{cases}$$

A few sets of Bernstein basis functions are illustrated in Figure 4.10.

Let us look at some properties of the Bernstein polynomials.

- *Partition of unity*: For any particular value of t , the sum of the Bernstein polynomials is one:

$$\sum_{i=0}^n B_i^n(t) = 1.$$

This is necessary for (4.5) to be a barycentric combination.⁷

- *Non-negativity*: Each Bernstein polynomial is non-negative within the interval $[0, 1]$. This property, along with the fact that they sum to one, results in the convex hull property from Section 4.2.2.
- *Symmetry*: The relation $B_i^n(t) = B_{n-i}^n(1-t)$, follows directly from (4.6). This is reflected in the symmetry property of Bézier curves.
- *Recursion*: The degree n polynomials can be generated from the degree $n-1$ polynomials,

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t).$$

See [43] for a derivation of the de Casteljau algorithm using this property.

⁷See the Geometric Fundamentals Chapter 2.

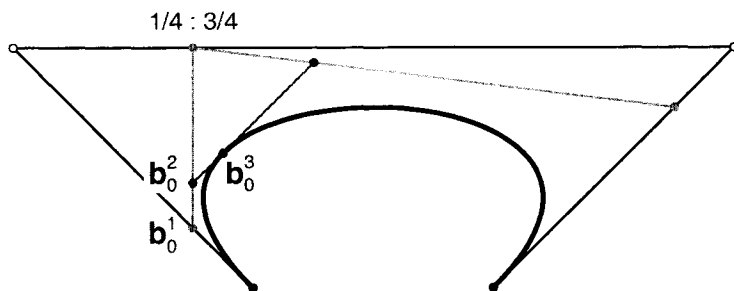


Figure 4.7. The de Casteljau algorithm applied to a cubic curve for $t = 1/4$.

- *Interval end conditions:*

$$B_i^n(0) = \delta_{i,0} \quad \text{and} \quad B_i^n(1) = \delta_{i,n} \quad \text{where} \quad \delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

As a result, Bézier curves have the endpoint interpolation property.

- *Linear precision:* The special linear combination

$$\sum_{i=0}^n \frac{i}{n} B_i^n(t) = t$$

results in the linear precision property of Bézier curves.

- *Single maximum:* $B_i^n(t)$ has one maximum, and this maximum occurs at $t = i/n$. This allows Bézier curves pseudo-local control.

Goldman [58] takes an interesting look at Bernstein polynomials in relation to other blending functions.

4.2.5. Derivatives of Bézier curves

Differentiating a parametric curve simply involves differentiating each component. The resulting vector is the *tangent vector* of the curve. If we take a Bézier curve of the form (4.5), and differentiate with respect to the parameter t , we obtain

$$\frac{d\mathbf{x}(t)}{dt} = \dot{\mathbf{x}}(t) = n \sum_{i=0}^{n-1} \Delta \mathbf{b}_i B_i^{n-1}(t), \quad (4.10)$$

where $\Delta \mathbf{b}_i = \mathbf{b}_{i+1} - \mathbf{b}_i$ is known as a *forward difference*. Notice that $\dot{\mathbf{x}}(t)$ is simply a degree $n - 1$ Bézier curve with control vectors (rather than points). Figure 4.11 illustrates

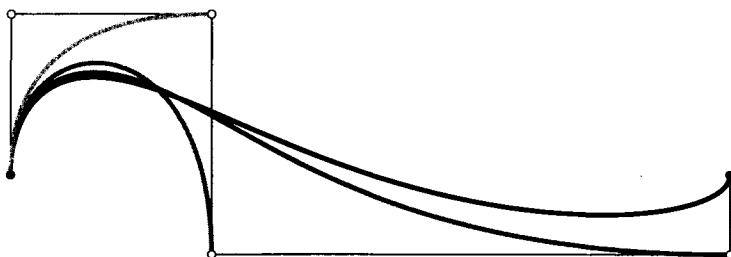


Figure 4.8. The intermediate points $\mathbf{b}_0^r(t)$ from the de Casteljau algorithm produce curves of degree r when plotted for all $t \in [0, 1]$.

this formulation of the first derivative, which is also called a *hodograph*. At the endpoints, the derivative takes the simple form

$$\dot{\mathbf{x}}(0) = n\Delta\mathbf{b}_0 \quad \text{and} \quad \dot{\mathbf{x}}(1) = n\Delta\mathbf{b}_{n-1},$$

which implies that the tangents at the ends are parallel to the polygon legs there. More on hodographs may be found in Bézier [8], Forrest [54], Sederberg and Wang [97]; and see Nachman [82] for more on derivatives.

The first derivative (4.10) can be reformulated using the commutativity of the summation and difference operators, thus becoming

$$\dot{\mathbf{x}}(t) = n\Delta \sum_{i=0}^{n-1} \mathbf{b}_i B_i^{n-1}(t) = n\Delta\mathbf{b}_0^{n-1}. \quad (4.11)$$

Recall from the de Casteljau algorithm and (4.9) that $n\Delta\mathbf{b}_0^{n-1}$ is computed in the next to last step of the algorithm. Figure 4.12 illustrates.

The second derivative of a Bézier curve follows by differentiating (4.10) again, producing

$$\ddot{\mathbf{x}}(t) = n(n-1) \sum_{i=0}^{n-2} \Delta^2 \mathbf{b}_i B_i^{n-2}(t), \quad (4.12)$$

where $\Delta^2 \mathbf{b}_i = \Delta(\Delta \mathbf{b}_i) = \mathbf{b}_{i+2} - 2\mathbf{b}_{i+1} + \mathbf{b}_i$. As with the first derivative, the second derivative can also be reformulated in terms of intermediate points in the de Casteljau algorithm. Now, the points in the second-to-last column of (4.9) define the second derivative:

$$\ddot{\mathbf{x}}(t) = \Delta^2 \mathbf{b}_0^{n-2}(t).$$

At the endpoints, the second derivative has a nice geometric interpretation which is illustrated in Figure 4.13. Higher derivatives follow similarly.

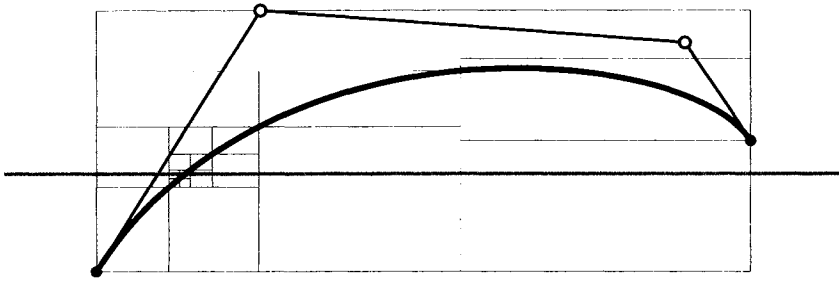


Figure 4.9. Finding an intersection of a cubic Bézier curve with the x -axis (gray) via repeated subdivision.

4.2.6. Degree elevation of Bézier curves

A degree n polynomial is also one of degree $n + 1$. A curve in monomial form will simply have a zero leading coefficient. Similarly, we may write a degree n Bézier curve as one of degree $n + 1$. We'll use a quadratic curve to demonstrate the principle. The trick is to multiply the quadratic expression by $[t + (1 - t)]$. Reassembling common powers of t yields

$$\mathbf{x}(t) = \mathbf{b}_0 B_0^3 + \left[\frac{1}{3}\mathbf{b}_0 + \frac{2}{3}\mathbf{b}_1\right] B_1^3 + \left[\frac{2}{3}\mathbf{b}_1 + \frac{1}{3}\mathbf{b}_2\right] B_2^3 + \mathbf{b}_2 B_3^3.$$

This is the process of *degree elevation*. The trace of the curve written as a cubic is identical to that of the original quadratic.

Generalizing this process: Degree elevation of a degree n Bézier curve with control points \mathbf{b}_j produces a curve of degree $n + 1$ with control points \mathbf{c}_i , where

$$\mathbf{c}_i = \frac{i}{n+1}\mathbf{b}_{i-1} + \left(1 - \frac{i}{n+1}\right)\mathbf{b}_i. \quad (4.13)$$

An example is illustrated in Figure 4.14.

Repeated degree elevation results in a polygon which converges to the curve, although Cohen and Schumaker [24] show this is not a practical method to render a Bézier curve. Trump and Prautzsch [103] examine arbitrarily high degree elevation. This convergence property follows from the *Weierstrass approximation theorem*, and more details are given by Farin [43]. Additionally, see Davis [29] and Korovkin [74].

A wealth of literature may be found on the reverse process: *degree reduction*. See [19,37,38,43,54,25,84,105] for a variety of methods. How can we tell if a Bézier curve is really a degree elevated curve? A degree $n - 1$ curve will have an n^{th} derivative that is identically zero, or $\Delta^n \mathbf{b}_0 = \mathbf{0}$.

4.2.7. Interrogation techniques for Bézier curves

Determining if a curve meets certain design specifications calls for methods to measure the curve. The interrogation methods below, curvature and torsion, are the most basic

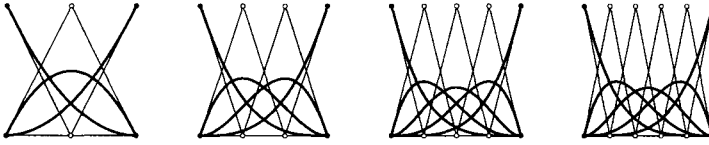


Figure 4.10. Quadratic, cubic, quartic, and quintic Bernstein basis functions plotted as functional Bézier curves.

measures. See the Geometric Fundamentals Chapter 2 for an introduction to these concepts. The discussion that follows focuses on these measures from the point of view of Bézier techniques.

The curvature of a curve is the most significant descriptor of its shape. Typically, curvature is visualized by graphing it as a function of t , which is called a *curvature plot*. An example is illustrated in Figure 4.15. For Bézier curves, curvature may be computed without having to compute derivatives explicitly. At $t = 0$, the curvature of a Bézier curve is given by

$$\kappa(0) = 2 \frac{n-1}{n} \frac{\text{area}[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]}{\|\mathbf{b}_1 - \mathbf{b}_0\|^3}. \quad (4.14)$$

We see that the curve has zero curvature at $t = 0$ if the three points $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$ are collinear. A similar formula follows for $t = 1$. If the curvature is desired at parameter values other than 0 or 1, employ subdivision.

By definition, a 3D curve has nonnegative curvature. For 2D Bézier curves, signed curvature is easily introduced by defining the area in terms of a determinant. A signed curvature allows for identifying *inflection points*: These are points where the curvature changes sign. Roulier [92] examines Bézier curves with positive curvature.

The *torsion* measures the 3D twisting of a curve, or in other words, the change in the binormal vector. For Bézier curves, torsion takes on a simple form at the ends. At $t = 0$, we have

$$\tau(0) = \frac{3n-2}{2} \frac{n}{n} \frac{\text{volume}[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]}{\text{area}[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]^2}.$$

The primary application of curvature and torsion has been in the context of piecewise curves, and some examples include [46,62,72,83,94].

4.2.8. Basis conversion

The Bernstein form has been shown to be numerically more stable than the monomial form by Farouki and Rajan [49]. Additionally, Farouki [47] showed that conversion between the monomial and Bernstein forms has the potential to be numerically unstable. A degree n curve in monomial form (4.2) is related to a curve in Bernstein form via

$$\mathbf{a}_i = \binom{n}{i} \Delta^i \mathbf{b}_0$$

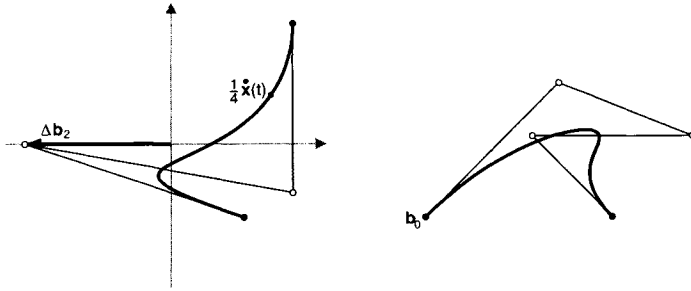


Figure 4.11. Right: A quartic Bézier curve. Left: The hodograph of the quartic, scaled by one-fourth.

for $i = 0, \dots, n$. Thus, the first monomial coefficient is a point, namely \mathbf{b}_0 , and the others are the scalings of the derivative vectors.

Cubic Hermite curves are another common curve form. They are specified by two points, \mathbf{p}_0 and \mathbf{p}_1 , and tangent vectors \mathbf{m}_0 and \mathbf{m}_1 at the data points. More specifically, the Hermite form is defined as

$$\mathbf{x}(t) = \mathbf{p}_0 H_0^3(t) + \mathbf{m}_1 H_1^3(t) + \mathbf{m}_2 H_2^3(t) + \mathbf{p}_1 H_3^3(t),$$

where the Hermite basis functions H_i^3 are defined as

$$\begin{aligned} H_0^3(t) &= B_0^3(t) + B_1^3(t), & H_1^3(t) &= \frac{1}{3} B_1^3(t), \\ H_3^3(t) &= B_2^3(t) + B_3^3(t), & H_2^3(t) &= -\frac{1}{3} B_2^3(t). \end{aligned}$$

This relation between the Bernstein and Hermite bases implies that

$$\mathbf{p}_0 = \mathbf{b}_0, \quad \mathbf{m}_0 = 3\Delta\mathbf{b}_0, \quad \mathbf{m}_1 = 3\Delta\mathbf{b}_2, \quad \mathbf{p}_1 = \mathbf{b}_3.$$

Li and Zhang [78] detail other basis conversions. Boehm [10,12] describes the conversion from B-spline to Bézier; See also the B-spline Basics Chapter 6.

4.2.9. Piecewise Bézier curves

Practical applications which take advantage of the geometric nature of Bézier curves typically depend on rather low degree curves. Complicated shapes are formed by piecing together curves. How to do so in a smooth manner is the focus of this section. A more rigorous examination of piecewise curves, called splines, may be found in the B-spline Basics Chapter 6. Issues of smoothness are investigated in the Geometric Continuity Chapter 8.

Suppose we begin with two degree n curves: $\mathbf{b}(u)$ with control points $\mathbf{b}_0, \dots, \mathbf{b}_n$, defined over $[u_0, u_1]$, and $\mathbf{c}(u)$ with control points $\mathbf{b}_n, \dots, \mathbf{b}_{2n}$, defined over $[u_1, u_2]$. Due to the

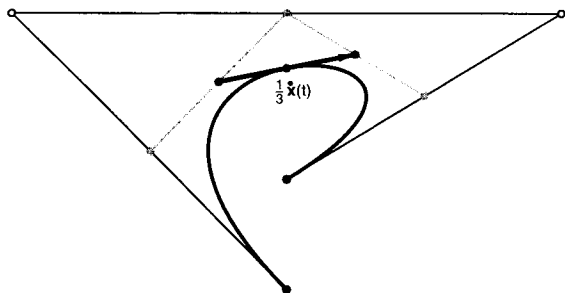


Figure 4.12. The first derivative of a Bézier curve is a scaling of the difference vector $(\mathbf{b}_1^{n-1} - \mathbf{b}_0^{n-1})$, and thus is a by-product of the de Casteljau algorithm.

endpoint interpolation property of Bézier curves, we know that these two curves meet, or are C^0 , at \mathbf{b}_n .

What are the conditions on the control points for the curves to be differentiable at \mathbf{b}_n ? The derivative of a Bézier curve defined over an arbitrary interval involves the chain rule, for example

$$\frac{d\mathbf{b}(u)}{du} = \frac{d\mathbf{b}}{dt} \frac{dt}{du} = \frac{n}{\Delta_0} \frac{d\mathbf{b}}{dt},$$

where $\Delta_0 = u_1 - u_0$. Drawing from Section 4.2.5, equate this expression for both curves at $u = u_1$, and we find

$$\frac{1}{\Delta_0} \Delta \mathbf{b}_{n-1} = \frac{1}{\Delta_1} \Delta \mathbf{b}_n$$

to be the condition for the curves to be C^1 . More geometrically, this requires that the three points $\mathbf{b}_{n-1}, \mathbf{b}_n, \mathbf{b}_{n+1}$ be collinear and their spacing must be in the ratio $\Delta_0 : \Delta_1$ as illustrated in Figure 4.16, left.

C^2 constructions are also of practical importance. For the curves to be twice differentiable at the junction, the two quadratic polynomials defined by $\mathbf{b}_{n-2}, \mathbf{b}_{n-1}, \mathbf{b}_n$ and $\mathbf{b}_n, \mathbf{b}_{n+1}, \mathbf{b}_{n+2}$ must describe the same global quadratic polynomial. The curves must satisfy the C^1 conditions, and the additional geometric conditions are described in Figure 4.16, right. This necessitates the existence of an auxiliary point, shaded gray in the figure. Notice that the left figure is not C^2 .

4.3. RECTANGULAR BÉZIER PATCHES

Let us extend Bézier techniques for curves to a surface form. A parametric surface is the result of a map of the real plane into 3-space. This plane, or *domain*, is defined by a (u, v) -coordinate system. A 3D surface point corresponding to a particular (u, v) is a

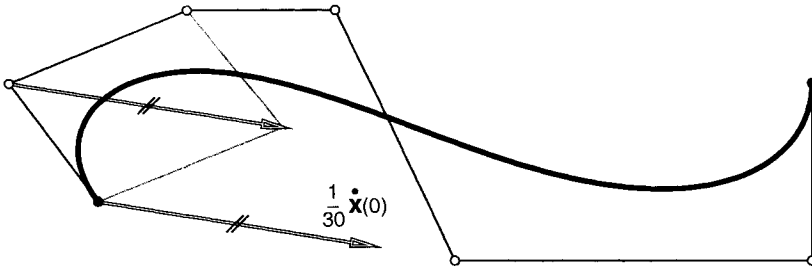


Figure 4.13. The second derivative of a degree six Bézier curve at $t = 0$ is a scaling of the diagonal of the parallelogram formed by the first three control points.

point:

$$\mathbf{x}(u, v) = \begin{bmatrix} f(u, v) \\ g(u, v) \\ h(u, v) \end{bmatrix}. \tag{4.15}$$

In this chapter, the functions f, g, h will be combinations of Bernstein polynomials. The surface $\mathbf{x}(u, v)$ is defined for all values of u and v , although we will primarily consider

$$\{(u, v) : 0 \leq u, v \leq 1\}. \tag{4.16}$$

As this indicates, the surface in this limited extent has a rectangular boundary. We will refer to this as a *patch*.

4.3.1. Bilinear patches

To begin our discussion of rectangular Bézier patches, let's start with the simplest form: bilinear patches. As the name suggests, the degree is linear in each parametric direction.

A bilinear Bézier patch $\mathbf{x}(u, v)$ is defined by four points $\mathbf{b}_{0,0}, \mathbf{b}_{0,1}, \mathbf{b}_{1,0}, \mathbf{b}_{1,1}$, and it takes the form

$$\mathbf{x}(u, v) = \begin{bmatrix} B_0^1(u) & B_1^1(u) \end{bmatrix} \begin{bmatrix} \mathbf{b}_{0,0} & \mathbf{b}_{0,1} \\ \mathbf{b}_{1,0} & \mathbf{b}_{1,1} \end{bmatrix} \begin{bmatrix} B_0^1(v) \\ B_1^1(v) \end{bmatrix}. \tag{4.17}$$

Figure 4.17, left, illustrates such a surface. The linear Bernstein polynomials, for example in u , are simply $B_0^1(u) = (1 - u)$ and $B_1^1(u) = u$.

At first glance, (4.17) does not convey very much geometric information. By simply rewriting the bilinear patch as

$$\mathbf{x}(u, v) = (1 - v)\mathbf{c}_0 + v\mathbf{c}_1 \tag{4.18}$$

where

$$\mathbf{c}_0 = (1 - u)\mathbf{b}_{0,0} + u\mathbf{b}_{1,0} \quad \text{and} \quad \mathbf{c}_1 = (1 - u)\mathbf{b}_{0,1} + u\mathbf{b}_{1,1},$$

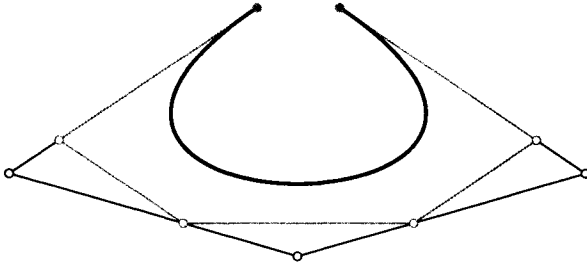


Figure 4.14. Degree elevation of a quartic Bézier curve. The quintic polygon is gray.

we can gather a much better feeling for the shape of the bilinear patch. Figure 4.17, right, illustrates this construction which builds intermediate points in the u -direction, and then builds the patch point by linear interpolation in the v -direction.

Alternatively, we could have built intermediate points in the v -direction:

$$\mathbf{d}_0 = (1 - v)\mathbf{b}_{0,0} + v\mathbf{b}_{0,1} \quad \text{and} \quad \mathbf{d}_1 = (1 - v)\mathbf{b}_{1,0} + v\mathbf{b}_{1,1},$$

and then the point on the patch is

$$\mathbf{x}(u, v) = (1 - u)\mathbf{d}_0 + u\mathbf{d}_1. \tag{4.19}$$

The result (4.18) is the same as (4.19).

Another name for a bilinear patch is a *hyperbolic paraboloid*. It is covered by two families of straight lines, which is apparent when considering the curves defined by (4.18) and (4.19). These two sets of curves on the patch are called *isoparametric curves*. The four isoparametric curves (lines) corresponding to the edges, $(u, 0), (u, 1), (0, v)$, and $(1, v)$, are commonly referred to as the *boundary curves* of the patch.

A hyperbolic paraboloid also contains curves. For instance, consider the line $u = v$ in the domain. In parametric form, it may be written as $u(t) = t, v(t) = t$. This domain diagonal is mapped to the 3D curve $\mathbf{c}(t) = \mathbf{x}(t, t)$ on the surface. In more detail:

$$\mathbf{c}(t) = \begin{bmatrix} 1 - t & t \end{bmatrix} \begin{bmatrix} \mathbf{b}_{0,0} & \mathbf{b}_{0,1} \\ \mathbf{b}_{1,0} & \mathbf{b}_{1,1} \end{bmatrix} \begin{bmatrix} 1 - t \\ t \end{bmatrix},$$

and after collecting terms gives a quadratic Bézier curve

$$\mathbf{c}(t) = \mathbf{b}_{0,0}B_0^2(t) + \left[\frac{1}{2}\mathbf{b}_{0,1} + \frac{1}{2}\mathbf{b}_{1,0}\right]B_1^2(t) + \mathbf{b}_{1,1}B_2^2(t).$$

4.3.2. Bézier patches

Generalizing (4.17) to higher degrees, a degree (m, n) rectangular Bézier patch takes the form

$$\mathbf{x}(u, v) = \begin{bmatrix} B_0^m(u) & \dots & B_m^m(u) \end{bmatrix} \begin{bmatrix} \mathbf{b}_{0,0} & \dots & \mathbf{b}_{0,n} \\ \vdots & & \vdots \\ \mathbf{b}_{m,0} & \dots & \mathbf{b}_{m,n} \end{bmatrix} \begin{bmatrix} B_0^n(v) \\ \vdots \\ B_n^n(v) \end{bmatrix}. \tag{4.20}$$

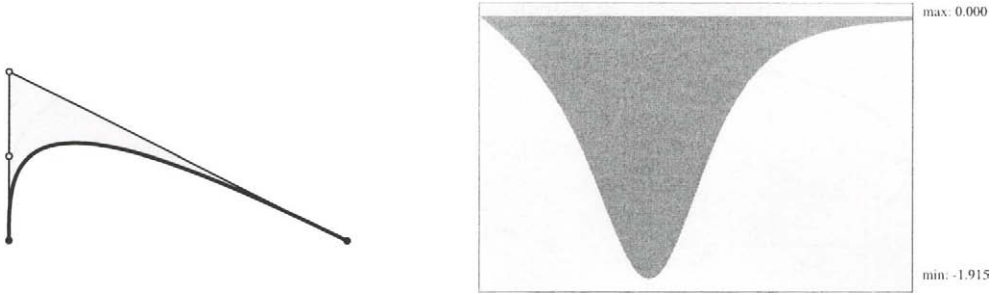


Figure 4.15. A curvature plot of a cubic Bézier curve.

Figure 4.18 illustrates a degree (3, 3) surface. The collection of control points is referred to as the *control net*. Equation (4.20) may be conveniently abbreviated as

$$\mathbf{x}(u, v) = M^T \mathbf{B} \mathbf{N}. \tag{4.21}$$

We may also write a Bézier patch as

$$\mathbf{x}(u, v) = \sum_{i=0}^m \sum_{j=0}^n \mathbf{b}_{i,j} B_i^m(u) B_j^n(v). \tag{4.22}$$

Bézier patches fall into the class of *tensor product surfaces*. The tensor product property is a very powerful conceptual tool for understanding Bézier patches. Figure 4.19 illustrates how the shape of a Bézier patch can be thought of as a record of the shape of a template moving and changing shape through space. Consider one value of v and the term $\mathbf{D} = \mathbf{B} \mathbf{N}$ in (4.21). This defines a point \mathbf{d}_i on each curve defined by the control polygon $\mathbf{b}_{i,j}, j = 0, \dots, n$. The \mathbf{d}_i are the control points for a curve in the u -direction, namely $M^T \mathbf{D}$. As u varies, this expression defines the shape of the template for one particular v . In a chapter written by Bézier [43], this concept is discussed from a practitioner’s point of view.

4.3.3. Properties of Bézier patches

Many of the properties of Bézier patches are direct generalizations of the curve ones.

- *Endpoint interpolation*: The patch passes through the four corner control points, that is

$$\begin{aligned} \mathbf{x}(0, 0) &= \mathbf{b}_{0,0} & \mathbf{x}(1, 0) &= \mathbf{b}_{m,0} \\ \mathbf{x}(0, 1) &= \mathbf{b}_{0,n} & \mathbf{x}(1, 1) &= \mathbf{b}_{m,n}. \end{aligned}$$

Also, the control net’s boundary control points are the control points of the patch boundary curves. For example: the curve $\mathbf{x}(u, 0)$ has the control polygon $\mathbf{b}_{i,0}$ for $i = 0, \dots, m$.

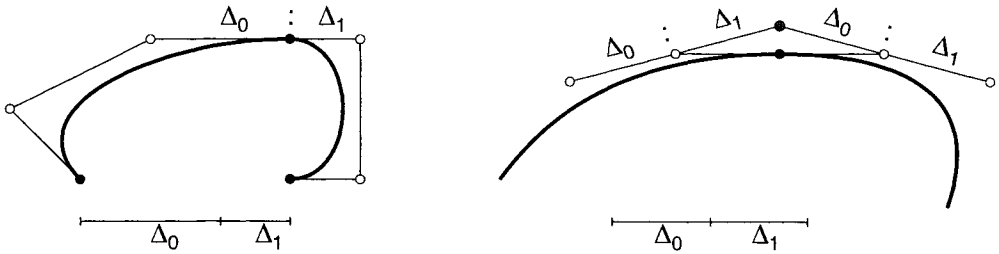


Figure 4.16. Bézier curve C^1 (left) and C^2 (right) conditions.

- *Symmetry:* We could re-index the control net so that any of the corners corresponds to $\mathbf{b}_{0,0}$, and evaluation would result in a patch with the same shape as the original one.
- *Affine invariance:* Apply an affine map to the control net, and then evaluate the patch. This surface will be identical to the surface created by applying the same affine map to the original patch. See (4.7) for the analogous property for curves.
- *Convex hull:* For $(u, v) \in [0, 1] \times [0, 1]$, the patch $\mathbf{x}(u, v)$ is in the convex hull of the the control net.
- *Bilinear precision:* A degree (m, n) patch is identical to the bilinear interpolant to the four corner control points if the control points satisfy the following conditions. The boundary curves are linearly precise, and the interior control points are uniformly-spaced on lines connecting corresponding boundary control points on adjacent edges
- *Tensor product:* Bézier patches are in the class of tensor product surfaces. This property allows Bézier patches to be dealt with in terms of isoparametric curves, which in turn simplifies evaluation and other operations. This also implies that the *total degree* of a (m, n) patch is $2mn$. The total degree is the maximum number of intersections of the patch with a straight line.
- *Functional patches:* A functional Bézier patch defined over $[0, 1] \times [0, 1]$ has Bézier control points

$$\mathbf{b}_{i,j} = \begin{bmatrix} i/m \\ j/n \\ b_{i,j} \end{bmatrix}.$$

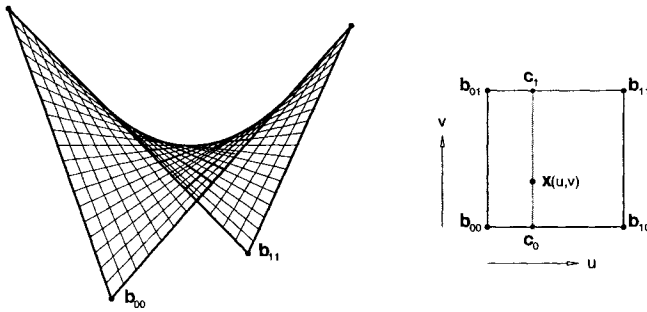


Figure 4.17. A bilinear Bézier patch illustrated in a 3D view on the left and the construction of $\mathbf{x}(1/3, 1/3)$ is on the right.

4.3.4. Evaluation of Bézier patches

Let us take advantage of the de Casteljau algorithm and the tensor product property as described in Section 4.3.2 to formulate a straightforward method to evaluate a Bézier patch, the *2-stage de Casteljau evaluation* method. First, consider $\mathbf{C} = M^T \mathbf{B}$ in (4.21). The elements of \mathbf{C} take the form

$$\mathbf{c}_j = \sum_{i=0}^m \mathbf{b}_{i,j} B_i^m(u) \tag{4.23}$$

for $j = 0, \dots, n$. Simply evaluate each degree m Bézier curve using the de Casteljau algorithm. The second and final evaluation step, $\mathbf{x}(u, v) = \mathbf{C}N$ or

$$\mathbf{x}(u, v) = \sum_{j=0}^n \mathbf{c}_j B_j^n(v), \tag{4.24}$$

consists of evaluating a degree n Bézier curve via the de Casteljau algorithm. The roles of u and v can be switched: First compute $\mathbf{D} = \mathbf{B}N$, and then $\mathbf{x} = M^T \mathbf{D}$.

Another evaluation method, the *3-stage de Casteljau evaluation* method, is quite useful if we want the first partial derivatives of the surface. It involves only a slight modification of the 2-stage method. Instead of computing the point on the curve in (4.23), stop the de Casteljau algorithm at the next to last step, saving the two points which span the tangent to the curve. As a result, we will have two “rows” of degree n curves. Next, evaluate these two curves at v , again stopping the de Casteljau algorithm at the next-to-last step, resulting in four points. These four points define a bilinear patch, and they span the tangent plane at (u, v) . Evaluate this as we did in Section 4.3.1. How we use this algorithm in the calculation of derivatives is discussed in more detail in Section 4.3.5. Other evaluation methods and comparisons between them may be found in [43,45,79].

4.3.5. Derivatives of Bézier patches

A derivative of a surface is the tangent vector of a curve on the surface. There are two isoparametric curves through $\mathbf{x}(u, v)$. Let us focus on the $u = \text{constant}$ curve, as in

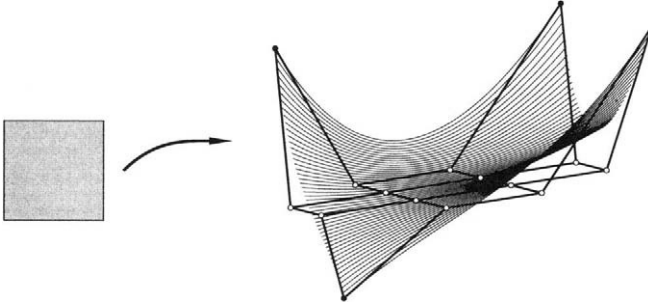


Figure 4.18. The control net and $u = \text{constant}$ isoparametric curves of a degree (3, 3) Bézier patch are illustrated.

(4.24), and differentiate it with respect to v . The resulting tangent vector \mathbf{x}_v is called the v -partial derivative or v -partial. More precisely, differentiating (4.22) with respect to v results in

$$\mathbf{x}_v(u, v) = \frac{\partial \mathbf{x}(u, v)}{\partial v} = n \sum_{i=0}^m \sum_{j=0}^{n-1} \Delta^{0,1} \mathbf{b}_{i,j} B_i^m(u) B_j^{n-1}(v), \quad (4.25)$$

where $\Delta^{0,1} \mathbf{b}_{i,j} = \mathbf{b}_{i,j+1} - \mathbf{b}_{i,j}$. The u -partial takes a very similar form,

$$\mathbf{x}_u(u, v) = m \sum_{i=0}^{m-1} \sum_{j=0}^n \Delta^{1,0} \mathbf{b}_{i,j} B_i^{m-1}(u) B_j^n(v), \quad (4.26)$$

where $\Delta^{1,0} \mathbf{b}_{i,j} = \mathbf{b}_{i+1,j} - \mathbf{b}_{i,j}$.

The second partials are also of practical use. For example, the second u -partial takes the form

$$\frac{\partial^2 \mathbf{x}(u, v)}{\partial u^2} = m(m-1) \sum_{i=0}^{m-2} \sum_{j=0}^n \Delta^{2,0} \mathbf{b}_{i,j} B_i^{m-2}(u) B_j^n(v),$$

where $\Delta^{2,0} \mathbf{b}_{i,j}$ is the second forward difference applied to the i indices. In other words, we simply take the derivative of the curve (4.26). The v -partial follows similarly, and formulae for higher order partials may be found in any of the texts cited in Section 4.1. More detail concerning computational efficiency may be found in Mann and DeRose [79] and Spitzmueller [100].

The mixed partial, or *twist vector*, is denoted by $\mathbf{x}_{u,v}(u, v)$ and is obtained in either of two ways:

$$\mathbf{x}_{u,v}(u, v) = \frac{\partial \mathbf{x}_u(u, v)}{\partial v} \quad \text{or} \quad \mathbf{x}_{u,v}(u, v) = \frac{\partial \mathbf{x}_v(u, v)}{\partial u}.$$

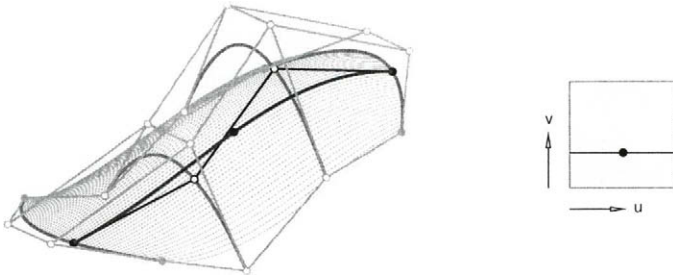


Figure 4.19. The shape of a Bézier patch may be described in terms of a moving template. The template for $v = 1/3$ is highlighted here.

Thus, by differentiating the v -partial (4.25) with respect to u ,

$$\mathbf{x}_{u,v}(u, v) = mn \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \Delta^{1,1} \mathbf{b}_{i,j} B_i^{m-1}(u) B_j^{n-1}(v),$$

The mixed forward difference $\Delta^{1,1} \mathbf{b}_{i,j}$ is equivalent to $\Delta^{0,1}(\Delta^{1,0} \mathbf{b}_{i,j})$, that is

$$\Delta^{1,1} \mathbf{b}_{i,j} = \mathbf{b}_{i+1,j+1} - \mathbf{b}_{i+1,j} - \mathbf{b}_{i,j+1} + \mathbf{b}_{i,j}. \tag{4.27}$$

which measures the deviation of the quadrilateral defined by the four points in (4.27) from a parallelogram, and this is illustrated in Figure 4.20. Notice that the twist at the corners involves only the control points at the corners.

The *normal* is a fundamental geometric concept which is used throughout computer graphics and CAD/CAM; See the Direct Rendering of Freeform Surfaces Chapter 30. At a given point $\mathbf{x}(u, v)$ on a patch, the normal is perpendicular to the surface at \mathbf{x} , and the normal and \mathbf{x} define the *tangent plane*. More precisely, the normal is defined by

$$\mathbf{n} = \frac{\mathbf{x}_u \wedge \mathbf{x}_v}{\|\mathbf{x}_u \wedge \mathbf{x}_v\|}, \tag{4.28}$$

and thus is a unit vector. Ideas for handling a denominator that is nearly zero are given attention in Farin [43].

Calculation of the normal requires knowledge of both the u - and v -partials. For this reason, the 3-stage evaluation method from Section 4.3.4 is more suitable in this situation than the 2-stage method.

4.3.6. Working with Bézier patches

Many of the operations applied to Bézier patches are direct generalizations of the techniques for curves. Again, this is due to the tensor product nature of Bézier patches, which allows for an algorithmic approach based on isoparametric curves.

Recall *degree elevation* for curves from Section 4.2.6; That same technique is used for an (m, n) Bézier patch in the following manner. Raising the degree from m to $m + 1$ results

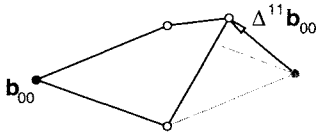


Figure 4.20. The mixed forward difference vector measures the deviation of the quadrangle from a parallelogram.

in a control net which has $n + 1$ “columns” of control points, each column containing $m + 2$ control points. These latter columns are simply obtained from the original columns by the process of degree elevation for curves.

Another curve operation from Section 4.2.3 is *subdivision*. Similarly, a patch may be subdivided into two patches. The u -parameter \hat{u} splits the domain unit square into two rectangles as shown in Figure 4.21. The patch is split along this isoparametric curve into two patches, together identical to the original patch. The algorithm: Perform curve subdivision for each degree m “row” of the control net at parameter \hat{u} . Since each of the two new patches are Bézier patches, they are each defined over the domain (4.16). See Schwartz [95] for more on this topic.

The properties of Bézier patches are utilized to benefit applications in [2,64,77,90], convexity conditions are explored in [20], fitting and design issues are examined in [39,73,99], and a hybrid Bézier patch is introduced in [52]. See the Geometric Fundamentals and Direct Rendering of Freeform Surfaces Chapters 2, 30 for information on interrogation techniques that may be applied to Bézier patches. Bézier patches can be extended to model volumes by generalizing the domain to a cube. For more information on this topic see [9,48,66,69,76,96].

4.3.7. C^1 Bézier patches

Following the discussion of piecewise curves in Section 4.2.9, here we examine the conditions under which two Bézier patches are differentiable. See the Geometric Continuity Chapter 8 for a more in-depth study.

We’ll assume that the patches are the same degree (m, n) and C^0 , that is, they share a common boundary curve. Additionally, define each Bézier patch over an arbitrary domain, thus we have $\mathbf{x}(u, v)$ defined over $[u_0, u_1] \times [v_0, v_1]$ and $\mathbf{y}(u, v)$ defined over $[u_1, u_2] \times [v_0, v_1]$. In order for \mathbf{x} and \mathbf{y} to be C^1 we require

$$\frac{\partial}{\partial u} \mathbf{x}(u, v) \Big|_{u=u_1} = \frac{\partial}{\partial u} \mathbf{y}(u, v) \Big|_{u=u_1} .$$

Drawing from the control point interpretation of the partials from Section 4.3.5 and applying the chain rule, this means that

$$\frac{1}{\Delta u_0} \sum_{j=0}^n \Delta^{1,0} \mathbf{b}_{m-1,j} B_j^n(v) = \frac{1}{\Delta u_1} \sum_{j=0}^n \Delta^{1,0} \mathbf{b}_{m,j} B_j^n(v),$$

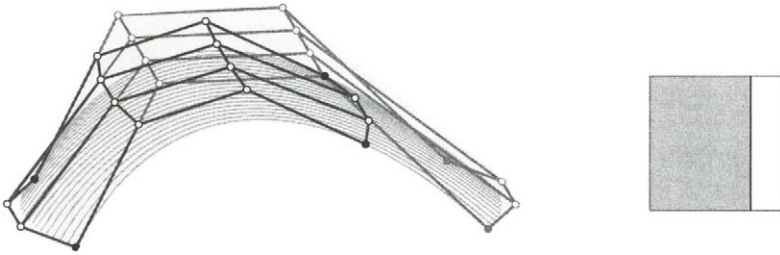


Figure 4.21. Subdivision of a Bézier patch at $\hat{u} = 0.75$. The original control net and the net for the patch over $[0, 0.75] \times [0, 1]$ are illustrated.

or in words, each row of control points must be collinear and reflect the corresponding ratio in the domain, as illustrated in Figure 4.22.

4.4. TRIANGULAR BÉZIER PATCHES

Triangular Bézier patches, or short *Bézier triangles*, are the true generalization of Bézier curves to a surface form.⁸ In practice, the tensor product Bézier patch from Section 4.3 receives more attention due to their prevalent use in industry. Most notably, tensor product patches are an entity in data transfer formats such as IGES.

However, triangular patches have much to offer. One important offering is the ability to model objects with arbitrary topology. Think of a sphere-like object, for example. Modeling this entirely with rectangular patches would require a degenerate patch. Another offering of triangular patches is the ease of modeling quadric surfaces. Rational patches are necessary, however, and these are discussed in the Rational Techniques Chapter 5. See the History Chapter 1 for more uses of triangular methods.

4.4.1. Bézier triangles introduced

Bézier triangles are constructed from a triangular domain. Thus *barycentric coordinates* are fundamental to their construction by providing an elegant tool for defining points in a plane with respect to this triangular reference frame. An in-depth study of barycentric coordinates is provided in the Geometric Fundamentals Chapter 2; See also Boehm and Prautzsch [17].

Since the domain of a Bézier triangle is a triangle, the conceptual picture of the control points of the surface takes a triangular form as illustrated in Figure 4.23. Differing from their rectangular counterparts, Bézier triangles have a single degree associated with them.

⁸For a discourse on this topic, see Barry and Goldman [5].

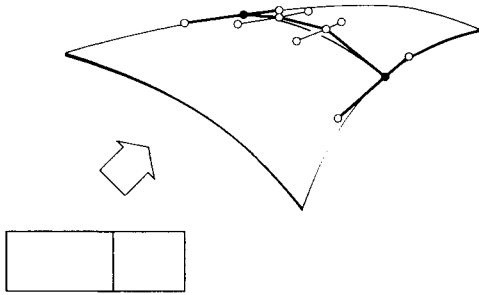
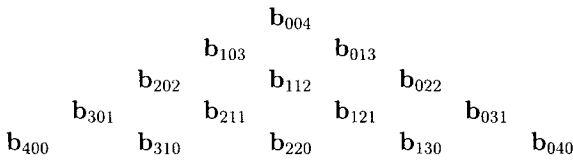


Figure 4.22. C^1 Bézier patches satisfy the criteria that the three control points in each row of along their common boundary curve are collinear, and the collinear points are positioned in the ratio dictated by the domain.

The control point indexing is described well by a quartic example.



Notice that the sum of the indices equals the degree. Often the abbreviated notation \mathbf{b}_i for \mathbf{b}_{ijk} is used. There is a pattern to the notation: For example, the control points between \mathbf{b}_{400} and \mathbf{b}_{040} each take the form \mathbf{b}_{**0} . The barycentric coordinates $(1, 0, 0)$ are associated with \mathbf{b}_{400} , $(0, 1, 0)$ with \mathbf{b}_{040} , and $(0, 0, 1)$ with \mathbf{b}_{004} .

A degree n triangular Bézier patch is defined as

$$\mathbf{x}(\mathbf{u}) = \sum_{|\mathbf{i}|=n} \mathbf{b}_i B_i^n(\mathbf{u}) \tag{4.29}$$

where $\mathbf{u} = (u, v, w)$ are barycentric coordinates and $|\mathbf{i}| = n$ represents all (ijk) combinations which sum to n . The $B_i^n(\mathbf{u})$ terms are the bivariate Bernstein polynomials

$$B_i^n(\mathbf{u}) = \frac{n!}{i!j!k!} u^i v^j w^k. \tag{4.30}$$

They are bivariate since $w = 1 - u - v$. More on these in Section 4.4.4. A Bézier triangle consists of all points $\mathbf{x}(\mathbf{u})$ with barycentric coordinates \mathbf{u} within the domain triangle, $0 \leq u, v, w \leq 1$. However, the surface is defined for \mathbf{u} outside of the domain triangle also.

4.4.2. Properties of Bézier triangles

Many of the properties of Bézier triangles are direct generalizations of the curve ones.

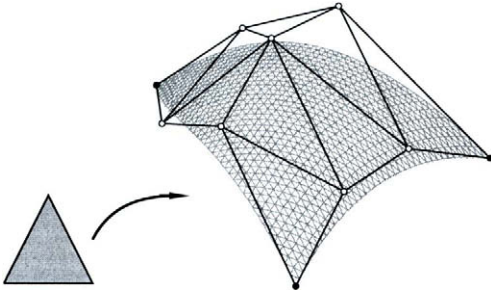


Figure 4.23. Illustrated is a cubic triangular Bézier patch with its control net.

- *Endpoint interpolation:* The patch passes through the three corner control points, that is

$$\mathbf{x}(1, 0, 0) = \mathbf{b}_{n,0,0} \quad \mathbf{x}(0, 1, 0) = \mathbf{b}_{0,n,0} \quad \mathbf{x}(0, 0, 1) = \mathbf{b}_{0,0,n}.$$

Also, each control net boundary corresponds to the control polygon for the patch boundary curves. For example: the curve $\mathbf{x}(0, v, w)$ has the control polygon $\mathbf{b}_{0,j,k}$ for $k = 0, \dots, n$ and $j + k = n$.

- *Symmetry:* We could re-index the control net so that any of the corners corresponds to $\mathbf{b}_{n,0,0}$, and evaluation would result in a patch with the same shape as the original one.
- *Affine invariance:* Apply an affine map to the control net, and then evaluate the patch. This surface will be identical to the surface created by applying the same affine map to the original patch.
- *Convex hull:* For $0 \leq u, v, w \leq 1$, the patch is in the convex hull of the control net.
- *Linear precision:* The three corner points define a plane. For a degree n patch, place the control points “uniformly” by constructing each $\mathbf{b}_{i,j,k}$ in this plane, and to have barycentric coordinates $(i/n, j/n, k/n)$. This Bézier triangle is identical to the linear Bézier triangle through the three corner points.
- *Total degree:* The total degree, or the maximum number of intersections of the patch with a straight line, of a degree n Bézier triangle is n^2 .
- *Functional patches:* A functional triangular Bézier patch defined over $0 \leq u, v, w \leq 1$ has Bézier control points as follows. Construct the degree n function as $z = f(x, y)$. Suppose the three corner points are given. Then the other Bézier control points are constructed to have linear precision in the x - and y -coordinates. The z -coordinate is independent.

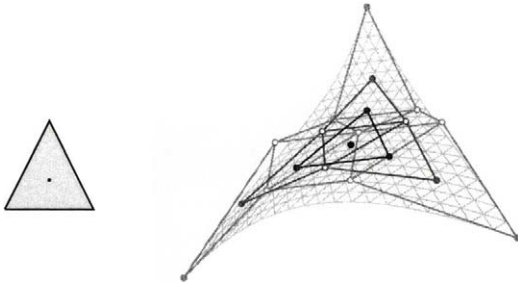


Figure 4.24. The triangular de Casteljau applied to a cubic patch for $\mathbf{u} = (1/3, 1/3, 1/3)$. The geometry of each step is differentiated by shades of gray.

4.4.3. The de Casteljau algorithm for Bézier triangles

Similar to the curve algorithm, the steps in the de Casteljau algorithm for Bézier triangles consists of repeated linear interpolation. Before describing the algorithm in more detail, we need to introduce the index notation $\mathbf{ei} = (100)$, $\mathbf{ej} = (010)$, and $\mathbf{ek} = (001)$.

Triangular de Casteljau Algorithm

Given: A degree n control net \mathbf{b}_i with $|\mathbf{i}| = n$ and barycentric coordinates \mathbf{u} .

Find: A point $\mathbf{x}(\mathbf{u}) = \mathbf{b}_{0,0,0}^n(\mathbf{u})$ on the Bézier triangle.

Compute: Set $\mathbf{b}_i^0 = \mathbf{b}_i$, and compute the points:

$$\mathbf{b}_i^r(\mathbf{u}) = u\mathbf{b}_{i+\mathbf{ei}}^{r-1} + v\mathbf{b}_{i+\mathbf{ej}}^{r-1} + w\mathbf{b}_{i+\mathbf{ek}}^{r-1} \quad \begin{cases} r = 1, \dots, n & \text{and} \\ |\mathbf{i}| = n - r. \end{cases} \quad (4.31)$$

Figure 4.24 illustrates the steps of the algorithm applied to a cubic triangular patch. Linear interpolation is applied to the six “upright” triangles of the given control net. This produces a quadratic control net. Linear interpolation is applied to the three “upright” triangles, producing a linear control net. One linear interpolation step is applied, resulting in a point on the patch. Applying (4.31) to \mathbf{u} on a domain boundary edge, for example $\mathbf{u} = (0, v, w)$, causes the triangular algorithm to take the form of the curve algorithm (4.8).

4.4.4. Bivariate Bernstein polynomials

Bivariate Bernstein polynomials were introduced in (4.30). Let us take a closer look at them with the purpose of gaining insight into the appealing geometric properties of Bézier triangles.

Just as with the univariate polynomials, a geometric approach to studying the bivariate Bernstein polynomials is achieved by formulating them as functional Bézier triangles. To plot the i^{th} basis function, recall the form the control points must take from Section 4.4.2. Giving one control point a $z = 1$ value, while all others have $z = 0$, will isolate the i^{th} basis function. Three cubic Bernstein basis functions are illustrated in Figure 4.25. A triangular diagram such as the following cubic one illustrates the correspondence between

- *Linear precision:*

$$\sum_{|i|=n} \frac{i}{n} B_i^n(\mathbf{u}) = u,$$

and similarly for v and w . This identity results in the linear precision property of Bézier triangles.

See [20–22,61,104] for convexity analyses.

4.4.5. Derivatives of Bézier triangles

The nature of Bézier triangles calls for a more general derivative notation than was needed for rectangular patches. Here we need a *directional derivative*

$$D_{\mathbf{d}} \mathbf{b}^n(\mathbf{u}) = d \frac{\partial}{\partial u} \mathbf{b}^n(\mathbf{u}) + e \frac{\partial}{\partial v} \mathbf{b}^n(\mathbf{u}) + f \frac{\partial}{\partial w} \mathbf{b}^n(\mathbf{u}), \quad (4.33)$$

which is the derivative at \mathbf{u} in the direction $\mathbf{d} = (d, e, f)$. The direction \mathbf{d} is equivalent to the difference of two barycentric coordinates in the domain, thus $d + e + f = 0$.

The partials in (4.33) are defined by differentiating the bivariate Bernstein polynomials, for example

$$\frac{\partial}{\partial u} \mathbf{b}^n(\mathbf{u}) = n \sum_{|i|=n-1} \mathbf{b}_{i+e\mathbf{i}} B_i^{n-1}(\mathbf{u}).$$

Combining these expression, we find

$$D_{\mathbf{d}} \mathbf{b}^n(\mathbf{u}) = n \sum_{|i|=n-1} [d\mathbf{b}_{i+e\mathbf{i}} + e\mathbf{b}_{i+e\mathbf{j}} + f\mathbf{b}_{i+e\mathbf{k}}] B_i^{n-1}(\mathbf{u}). \quad (4.34)$$

Taking advantage of the notation from the triangular de Casteljau algorithm, we may rewrite (4.34) as

$$D_{\mathbf{d}} \mathbf{b}(\mathbf{u}) = n \sum_{|i|=n-1} \mathbf{b}_i^1(\mathbf{d}) B_i^{n-1}(\mathbf{u}). \quad (4.35)$$

This may be interpreted as one step ($r = 1$) of the triangular de Casteljau algorithm with respect to \mathbf{d} , producing $\mathbf{b}_i^1(\mathbf{d})$, and then $n - 1$ steps with respect to \mathbf{u} . Due to the linear nature of the de Casteljau algorithm, it is possible to rewrite (4.35) as

$$D_{\mathbf{d}} \mathbf{b}(\mathbf{u}) = n \sum_{|i|=1} \mathbf{b}_i^{n-1}(\mathbf{u}) B_i^1(\mathbf{d}), \quad (4.36)$$

which should be interpreted as executing the de Casteljau algorithm on the original net with respect to \mathbf{u} for $n - 1$ steps, and then executing one step of the algorithm on the \mathbf{b}_i^{n-1} with respect to \mathbf{d} . This has a nice geometric interpretation: The three control points for the final step define the *tangent plane*, and thus the *normal* to the patch at \mathbf{u} .

The r^{th} directional derivative, mixed directional derivatives, and cross boundary directional derivatives are explored in detail by Farin [41,43].

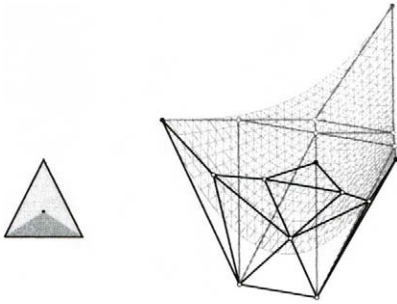


Figure 4.26. Subdivision of a Bézier triangle at $\hat{\mathbf{u}} = (1/3, 1/3, 1/3)$. One of the three resulting control nets is displayed in black.

4.4.6. Working with Bézier triangles

Just as for curves, *subdivision* of Bézier triangles is a by-product of the de Casteljau algorithm. Consider a point in the domain with barycentric coordinates $\hat{\mathbf{u}}$. The intermediate points from the de Casteljau algorithm, grouped appropriately, form three sub-patches as illustrated in Figure 4.26. More specifically, consider the patch across from \mathbf{b}_{n00} , it has control points

$$\hat{\mathbf{b}}_{r,j,k} = \mathbf{b}_{\mathbf{i}_0}^r \quad \begin{cases} r = 0, \dots, n \\ \mathbf{i}_0 = (0, j, k) \\ |\mathbf{i}_0| = n - r. \end{cases} \quad (4.37)$$

Repeated subdivision results in control nets which converge to the surface.

When $\hat{\mathbf{u}}$ is on a domain edge, for example $\hat{\mathbf{u}} = (0, v, w)$, then the control points from \mathbf{b}_{n00} to \mathbf{b}_{000}^n form the control polygon for the curve on the surface corresponding to the line in the domain from $(0, v, w)$ to $(1, 0, 0)$. This is called a *radial line*. When $\hat{\mathbf{u}}$ is outside the domain triangle, then continuity conditions for neighboring patches are revealed. See [13,15,51,57,69,98] for more detail and pointers to more literature.

Degree elevation for Bézier triangles allows a degree n patch to be written as a degree $n + 1$ patch by defining \mathbf{c}_j such that

$$\sum_{|\mathbf{j}|=n+1} \mathbf{c}_j B_j^{n+1}(\mathbf{u}) = \sum_{|\mathbf{i}|=n} \mathbf{b}_i B_i^n(\mathbf{u}). \quad (4.38)$$

By multiplying the right-hand side of (4.38) by $(u + v + w)$, and gathering the appropriate terms, the new control points are found to be

$$\mathbf{c}_j = \frac{1}{n + 1} (i\mathbf{b}_{j-\mathbf{e}_i} + j\mathbf{b}_{j-\mathbf{e}_j} + k\mathbf{b}_{j-\mathbf{e}_k}), \quad \begin{cases} \mathbf{j} = (i, j, k) \\ |\mathbf{j}| = n + 1. \end{cases}$$

Along a boundary curve, this expression reduces to that for curves. The degree elevated control net is within the convex hull of the original net. See Farin [41] for more details.

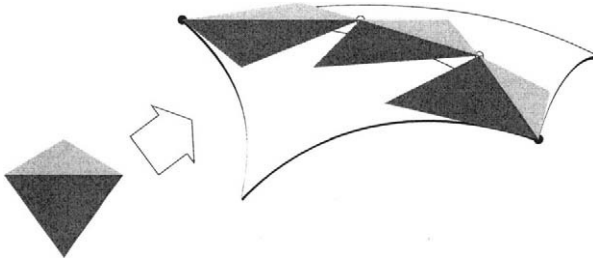


Figure 4.27. Two Bézier triangles are C^1 if adjacent triangles along the common boundary are coplanar and form affine pairs.

Utilizing the properties of Bézier triangles to benefit applications is explored in [3,42], triangular to rectangular patch conversion is studied in [18,70,71], and a classification of Bézier triangle may be found in [35].

To extend triangular patches to *volumes*, the domain becomes a tetrahedron. See [41,53,57,69,93] for more information.

4.4.7. C^1 Bézier triangles

The conditions under which two Bézier triangles are differentiable will draw from the directional derivative discussion of Section 4.4.5. We'll assume that the two patches are the same degree and C^0 , that is, they share a common boundary curve.

Consider all cross-boundary directional derivatives at a point on the boundary of one triangle, for example along $u = 0$. As a result, $\mathbf{u} = (0, v, 1 - v)$ in (4.35), and the expression becomes univariate, thus these derivatives correspond to tangents to curves across the triangles. If all such derivatives are equal for both patches, they are C^1 .

However, a more constructive description is needed. Examining (4.36), we observe that the directional derivative at a boundary involves the first two rows of control points at that boundary. The subdivision formula (4.37) reveals a geometric description of the conditions on these control points. The second row of the Bézier triangle $\hat{\mathbf{b}}$ is described by

$$\hat{\mathbf{b}}_{1,j,k} = \mathbf{b}_{0,j,k}^1 = u\mathbf{b}_{1,j,k} + v\mathbf{b}_{0,j+1,k} + w\mathbf{b}_{0,j,k+1}$$

for $j + k = n - 1$. This means that the triangle pairs along the boundary are coplanar and each of the triangle pairs can be described by the same affine map – they form affine pairs, as illustrated in Figure 4.27.

REFERENCES

1. G. Aumann. Interpolation with developable Bézier patches. *Computer Aided Geometric Design*, 8(5):409–420, 1991.

2. N. Aziz, R. Bataand, and S. Bhat. Bézier surface/surface intersection. *IEEE Computer Graphics and Applications*, 10(1):50–58, 1990.
3. R. Barnhill, B. Bloomquist, and A. Worsey. Adaptive contouring for triangular Bézier patches. In R. E. Barnhill, editor, *Geometry Processing*. SIAM, 1992.
4. P. Barry and R. Goldman. Three examples of dual properties of Bézier curves. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 61–70. Academic Press, 1989.
5. P. Barry and R. Goldman. What is the natural generalization of a Bézier curve? In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 71–86. Academic Press, 1989.
6. R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1987.
7. P. Bézier. Example of an existing system in the motor industry: The Unisurf system. *Proc. Royal Soc. of Loindon*, A321:207–218, 1971.
8. P. Bézier. *Numerical Control: Mathematics and Applications*. Wiley, 1972. translated from the French by R. Forrest.
9. P. Bézier. General distortion of an ensemble of biparametric patches. *Computer-Aided Design*, 10(2):116–120, 1978.
10. W. Boehm. Generating the Bézier points of B-spline curves and surfaces. *Computer-Aided Design*, 13(6):365–366, 1981.
11. W. Boehm. On cubics: a survey. *Computer Graphics and Image Processing*, 19:201–226, 1982.
12. W. Boehm. Generating the Bézier points of triangular splines. In R. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 77–91. North-Holland, 1983.
13. W. Boehm. Subdividing multivariate splines. *Computer-Aided Design*, 15(6):345–352, 1983.
14. W. Boehm. Bézier presentations of airfoils. *Computer Aided Geometric Design*, 4(1-2):17–22, 1987.
15. W. Boehm and G. Farin. Letter to the editor. *Computer-Aided Design*, 15(5):260–261, 1983. Concerning subdivision of Bézier triangles.
16. W. Boehm and A. Müller. On de Casteljaeu’s algorithm. *Computer Aided Geometric Design*, 16(7):583–586, 2000.
17. W. Boehm and H. Prautzsch. *Geometric Foundations of Geometric Design*. AK Peters, Boston, 1992.
18. I. Brueckner. Construction of Bézier points of quadrilaterals from those of triangles. *Computer-Aided Design*, 12(1):21–24, 1980.
19. G. Brunnett, T. Schreiber, and J. Braun. The geometry of optimal degree reduction of Bézier curves. *Computer Aided Geometric Design*, 13(8):773–788, 1996.
20. J. Carnicer, M. Floater, and J. Peña. Linear convexity conditions for rectangular and triangular Bernstein–Bézier surfaces. *Computer Aided Geometric Design*, 15(1):27–38, 1997.
21. G. Chang and P. Davis. The convexity of Bernstein polynomials over triangles. *J Approx Theory*, 40:11–28, 1984.
22. G. Chang and Y. Feng. An improved condition for the convexity of Bernstein–Bézier

- surfaces over triangles. *Computer Aided Geometric Design*, 1(3):279–283, 1985.
23. Y. Chua. Bézier brushstrokes. *Computer-Aided Design*, 22(9):550–555, 1990.
 24. E. Cohen and L. Schumaker. Rates of convergence of control polygons. *Computer Aided Geometric Design*, 2(1-3):229–235, 1985.
 25. J. Peters, D. Lutterkort, and U. Reif. Polynomial degree reduction in the L2-norm equals best Euclidean approximation of Bézier coefficients. *Computer Aided Geometric Design*, 16(7):607–612, 2000.
 26. J. Peters, D. Nairn, and D. Lutterkort. Sharp, quantitative bounds on the distance between a polynomial piece and its Bézier control polygon. *Computer Aided Geometric Design*, 16(7):613–631, 2000.
 27. W. Dahmen. Subdivision algorithms converge quadratically. *J. of Computational and Applied Mathematics*, 16:145–158, 1986.
 28. M. Daniel and J. Daubisse. The numerical problem of using Bézier curves and surfaces in the power basis. *Computer Aided Geometric Design*, 6(2):121–128, 1989.
 29. P. Davis. *Interpolation and Approximation*. Dover, New York, 1975. first edition 1963.
 30. P. de Casteljau. Outillages méthodes calcul. Technical report, A. Citroën, Paris, 1959.
 31. P. de Casteljau. Courbes et surfaces à pôles. Technical report, A. Citroën, Paris, 1963.
 32. P. de Casteljau. *Shape Mathematics and CAD*. Kogan Page, London, 1986.
 33. P. de Casteljau. de Casteljau’s autobiography: My time at Citroën. *Computer Aided Geometric Design*, 16(7):583–586, 2000.
 34. W. Degen. Some remarks on Bézier curves. *Computer Aided Geometric Design*, 5(3):259–268, 1988.
 35. W. Degen. The types of triangular Bézier surfaces. In G. Mullineux, editor, *The Mathematics of Surfaces VI*, pages 153–170. Oxford University Press, 1996.
 36. T. DeRose and R. Goldman. A tutorial introduction to blossoming. In H. Hagen and D. Roller, editors, *Geometric Modeling*. Springer, 1991.
 37. M. Eck. Degree reduction of Bézier curves. *Computer Aided Geometric Design*, 10(3-4):237–252, 1993.
 38. M. Eck. Least squares degree reduction of Bézier curves. *Computer-Aided Design*, 27(11):845–851, 1995.
 39. E. Eisele. Best approximations of symmetric surfaces by biquadratic Bézier surfaces. *Computer Aided Geometric Design*, 11(3):331–343, 1994.
 40. G. Farin. Some aspects of car body design at Daimler-Benz. In R. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 93–98. North-Holland, 1983.
 41. G. Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3(2):83–128, 1986.
 42. G. Farin. The use of triangular patches in CAD. In M. Wozny, H. McLaughlin, and J. Encarnacao, editors, *Geometric Modeling for CAD Applications*, pages 191–194. North-Holland, 1988.
 43. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1996. fourth edition.

44. G. Farin and P. Barry. A link between Lagrange and Bézier curve and surface schemes. *Computer-Aided Design*, 18:525–528, 1986.
45. G. Farin and D. Hansford. *The Essentials of CAGD*. AK Peters, 2000.
46. G. Farin and N. Sapidis. Curvature and the fairness of curves and surfaces. *IEEE Computer Graphics and Applications*, 9(2):52–57, 1989.
47. R. Farouki. On the stability of transformations between power and Bernstein polynomial forms. *Computer Aided Geometric Design*, 8(1):29–36, 1991.
48. R. Farouki and J. Hinds. A hierarchy of geometric forms. *IEEE Computer Graphics and Applications*, 5(5):51–78, 1985.
49. R. Farouki and V. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, 1987.
50. R. Farouki and V. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5(1):1–26, 1988.
51. Y. Feng. Rates of convergence of Bézier nets over triangles. *Computer Aided Geometric Design*, 4(3):245–249, 1987.
52. T. Foley and K. Opitz. Hybrid cubic Bézier patches. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in CAGD II*, pages 275–286. Academic Press, Boston, 1992.
53. T. Foley and H. Wolters. The hybrid quintic Bézier tetrahedron. *Computer Aided Geometric Design*, 14(7):603–618, 1997.
54. A. Forrest. Interactive interpolation and approximation by Bézier polynomials. *The Computer J*, 15(1):71–79, 1972. reprinted in *CAD* 22(9):527–537, 1990.
55. A. Forrest. The twisted cubic curve: a computer-aided geometric design approach. *Computer-Aided Design*, 12(4):165–172, 1980.
56. J. Gallier. *Curves and Surfaces in Geometric Modeling: Theory and Algorithms*. Morgan-Kaufmann, 1998.
57. R. Goldman. Using degenerate Bézier triangles and tetrahedra to subdivide Bézier curves. *Computer-Aided Design*, 14(6):307–311, 1982.
58. R. Goldman. An urnful of blending functions. *IEEE Computer Graphics and Applications*, 3(7):49–54, 1983.
59. R. Goldman and T. DeRose. Recursive subdivision without the convex hull property. *Computer Aided Geometric Design*, 3(4):247–265, 1986.
60. R. Goldman and D. Heath. Linear subdivision is strictly a polynomial phenomenon. *Computer Aided Geometric Design*, 1(3):269–278, 1984.
61. J. Gregory and J. Zhou. Convexity of Bézier on sub-triangles. *Computer Aided Geometric Design*, 8(3):207–213, 1991.
62. H. Hagen. Bézier-curves with curvature and torsion continuity. *Rocky Mtn. J of Math.*, 16(3):629–638, 1986.
63. D. Hilbert and S. Cohn-Vossen. *Geometry and the Imagination*. Chelsea, New York, 1952.
64. H. Hochfeld. Surface description in the application at Volkswagen. In R. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 35–42. North-Holland, 1983.
65. H. Hochfeld and M. Ahlers. Role of Bézier curves and surfaces in the Volkswagen CAD approach from 1967 to today. *Computer-Aided Design*, 22(9):598–608, 1990.

66. D. Holliday and G. Farin. A geometric interpretation of the diagonal of a tensor-product Bézier volume. *Computer Aided Geometric Design*, 16(8):837–840, 1999.
67. M. Hosaka and F. Kimura. Synthesis methods of curves and surfaces in interactive CAD, Bologna. In *Proc. Interactive Techniques in CAD*, pages 151–156, 1978.
68. J. Hoschek. Offset curves in the plane. *Computer-Aided Design*, 17(2):77–82, 1985.
69. J. Hoschek and D. Lasser. *Grundlagen der Geometrischen Datenverarbeitung*. B.G. Teubner, Stuttgart, 1989. English translation: *Fundamentals of Computer Aided Geometric Design*, AK Peters, 1993.
70. S.-M. Hu. Conversion of a triangular Bézier patch into three rectangular bézier patches. *Computer Aided Geometric Design*, 13(3):219–226, 1996.
71. K. Iino and D. Wilde. Subdivision of triangular Bézier patches into rectangular Bézier patches. *Transactions of the ASME*, 1992.
72. A. Jones. Curvature integration through constrained optimization. In N. Sapidis, editor, *Designing Fair Curves and Surfaces*, pages 29–44. SIAM, Philadelphia, 1994.
73. L. Kocić. Modification of Bézier curves and surfaces by degree elevation technique. *Computer-Aided Design*, 23(10):692–699, 1991.
74. P. Korovkin. *Linear Operators and Approximation Theory*. Hindustan Publishing Co., Delhi, 1960.
75. J. Lane and R. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Trans. Pattern Analysis Machine Intell.*, 2(1):35–46, 1980.
76. D. Lasser. Bernstein-Bézier representation of volumes. *Computer Aided Geometric Design*, 2(1-3):145–150, 1985.
77. D. Lasser. Intersection of parametric surfaces in the Bernstein-Bézier representation. *Computer-Aided Design*, 18(4):186–192, 1986.
78. Y.-M. Li and X.-Y. Zhang. Basis conversion among Bézier, Tchebyshev and Legendre. *Computer Aided Geometric Design*, 15(6):637–642, 1998.
79. S. Mann and T. DeRose. Computing values and derivatives of Bézier and B-spline tensor products. *Computer Aided Geometric Design*, 12(1):107–110, 1995.
80. D. Marsh. *Applied Geometry for Computer Graphics and CAD*. Springer-Verlag, 1999.
81. M. Mortenson. *Geometric Modeling*. Wiley, 1985.
82. L. Nachman. A note on control polygons and derivatives. *Computer Aided Geometric Design*, 8(3):223–226, 1991.
83. A. Nutbourne, P. McLellan, and R. Kensit. Curvature profiles for plane curves. *Computer-Aided Design*, 4(4):176–184, 1972.
84. J. Peterson. Degree reduction of Bézier curves. *Computer-Aided Design*, 23(6):460–461, 1991. Letter to the Editor.
85. H. N. Phien and N. Dejdumrong. Efficient algorithms for Bézier curves. *Computer Aided Geometric Design*, 17(3):247–250, 2000.
86. L. Piegl and W. Tiller. *The NURBS Book*. Springer Verlag, 1997. second edition.
87. H. Pottmann and G. Farin. Developable rational Bézier and B-spline surfaces. *Computer Aided Geometric Design*, 12(5):513–531, 1995.
88. L. Ramshaw. Blossoming: a connect-the-dots approach to splines. Technical report, Digital Systems Research Center, Palo Alto, Ca, 1987.

89. L. Ramshaw. Béziers and B-splines as multiaffine maps. In R. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, pages 757–776. Springer Verlag, 1988.
90. T. Reuding. Bézier patches on cubic grid curves - an application to the preliminary design of a yacht hull surface. *Computer Aided Geometric Design*, 6(1):11–21, 1989.
91. A. Rockwood and P. Chambers. *Interactive Curves and Surfaces*. Morgan Kaufmann, 1996.
92. J. Roulier. Bézier curves of positive curvature. *Computer Aided Geometric Design*, 5(1):59–70, 1988.
93. M. Sabin. Trinomial basis functions for interpolation in triangular regions (Bézier triangles). Technical report, British Aircraft Corporation, 1971.
94. N. Sapidis and G. Koras. Visualization of curvature plots and evaluation of fairness: an analysis of the effect of scaling. *Computer Aided Geometric Design*, 14(4):299–311, 1997.
95. A. Schwartz. Subdividing Bézier curves and surfaces. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 55–66. SIAM, Philadelphia, 1987.
96. T. Sederberg and S. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4):151–160, 1986. SIGGRAPH proceedings.
97. T. Sederberg and X. Wang. Rational hodographs. *Computer Aided Geometric Design*, 4(4):333–335, 1987.
98. H.-P. Seidel. A general subdivision theorem for Bézier triangles. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 573–582. Academic Press, 1989.
99. L. Shirman and C. Séquin. Local surface interpolation with Bézier patches: errata and improvements. *Computer Aided Geometric Design*, 8(3):217–222, 1991.
100. K. Spitzmueller. Partial derivatives of Bézier surfaces. *Computer-Aided Design*, 28(1), 1996.
101. E. Staerk. *Mehrfach differenzierbare Bézierkurven und Bézierflächen*. PhD thesis, T. U. Braunschweig, 1976.
102. B.-Q. Su and D.-Y. Liu. *Computational Geometry*. Academic Press, 1989.
103. W. Trump and H. Prautzsch. Arbitrarily high degree elevation of Bézier representations. *Computer Aided Geometric Design*, 13(5):387–398, 1996.
104. Z. Wang and Q. Liu. An improved condition for the convexity and positivity of Bernstein - Bézier surfaces over triangles. *Computer Aided Geometric Design*, 5(4):269–276, 1988.
105. M. Watkins and A. Worsey. Degree reduction for Bézier curves. *Computer-Aided Design*, 20(7):398–405, 1988.

Chapter 5

Rational Techniques

Hans J. Wolters

The following article will focus on rational parametric curves and surfaces and how they are used in geometric modeling applications. Aside from giving an overview of the state-of-the-art, I will emphasize the role of rational techniques in commercial CAD packages.

5.1. INTRODUCTION

Looking at the history of CAGD in industry, it can be argued that rational techniques and representations were at the root of geometric modeling. In particular conic sections and quadric surfaces were the initial building blocks of early CAD systems. Liming [27,28] detailed many geometric constructions for aircraft design using conics. Later S. Coons at Ford introduced conics into a CAD system; independently conics were used by engineers at Boeing. The quest for compatible formats and for exchanging data among different systems then led to the consideration of standard data formats. The need arose to find a common representation for basic spline curves and conics. Hence the *NURBS* representation was developed where *NURBS* stands for Non-Uniform Rational B-Spline. *NURBS* were first introduced in Versprille's thesis [43]; later A. Klosterman was instrumental in establishing *NURBS* as an industry wide standard by choosing them as data representation in SDRC's modeling software *I-DEAS*. Today *NURBS* are integral part of the IGES as well as the STEP standard.

Previous chapters already provided many of the building blocks for developing the material in this chapter. Concepts from projective geometry have been introduced in Chapter 2 on *Geometric Fundamentals*. In Chapter 4 on *Bézier Techniques*, Bézier curves and surfaces have been covered in depth. We will see that many of the algorithms presented there can easily be generalized to the rational case. I will present these algorithms briefly for the sake of completeness; the main focus will be on techniques which are specific to *NURBS*. Additionally, I will focus on topics related to the practical use of rational representations.

5.2. RATIONAL BÉZIER CURVES

In Chapter 4 on *Bézier Techniques*, non-rational Bézier curves have been introduced. In this section, we will generalize these curves to the rational case. Furthermore, we will give a brief outline of standard algorithms and then place special emphasis on algorithms which make use of the additional flexibility offered by rational curves.

5.2.1. Basic definitions

A rational Bézier curve of degree m is a parametric curve which is described by control points, $\mathbf{c}_i \in \mathbb{R}^n$, $n = 2, 3$, weights w_i and the parameter t . Without loss of generality we let t vary from 0 to 1. The curve has the form

$$\mathbf{c}(t) = \frac{\sum_{i=0}^m w_i \mathbf{c}_i B_i^m(t)}{\sum_{i=0}^m w_i B_i^m(t)} \quad (5.1)$$

The Bernstein Bézier basis functions are defined as

$$B_i^m(t) = \binom{m}{i} (1-t)^i t^{m-i} \quad (5.2)$$

Inspecting Equation (5.1) more closely reveals some simple properties: If we set all the weights w_i to 1, then by using the fact that $\sum_{i=0}^m B_i^m(t) = 1$, we obtain a non-rational Bézier curve. Furthermore, by ensuring that all $w_i \geq 0$, and some $w_i > 0$, we can guarantee that the curve does not contain any singularities. The curve $\mathbf{c}(t)$ in affine space can be viewed as the projection of a curve $\hat{\mathbf{c}}(t)$ which lives in projective space and whose control polygon consists of the homogeneous points $[w_i \mathbf{c}_i \ w_i]$. Hence, scaling all weights w_i by a common factor will not change the underlying control polygon or curve. Rational Bézier curves inherit some properties from the nonrational counterpart:

- *Affine Invariance*: This is easy to see: We just have to convince ourselves that Equation (5.1) is equivalent to an expression $\sum_{i=0}^m \alpha_i \mathbf{c}_i$ with $\sum_{i=0}^m \alpha_i = 1$. But this is readily verified by observing that

$$\alpha_i = \frac{w_i B_i^m}{\sum_{i=0}^m w_i B_i^m} \quad (5.3)$$

Here we dropped the parameter t since this reasoning is independent of t .

- *Convex hull property*: This property holds if $w_i \geq 0 \ \forall i$
- *Endpoint interpolation*: By inserting $t = 0$ and $t = 1$ into Equation (5.1), we can verify that

$$\mathbf{c}(0) = \frac{w_0 \mathbf{c}_0}{w_0} = \mathbf{c}_0 \text{ and } \mathbf{c}(1) = \frac{w_m \mathbf{c}_m}{w_m} = \mathbf{c}_m$$

- *Variational Diminishing Property*: The curve has no more intersections with any plane (or line) than its control polygon has. This can be seen by considering corner cutting algorithms. Corner cutting is equivalent to linear interpolation. Hence each corner cutting step reduces the number of intersections. Using the fact that

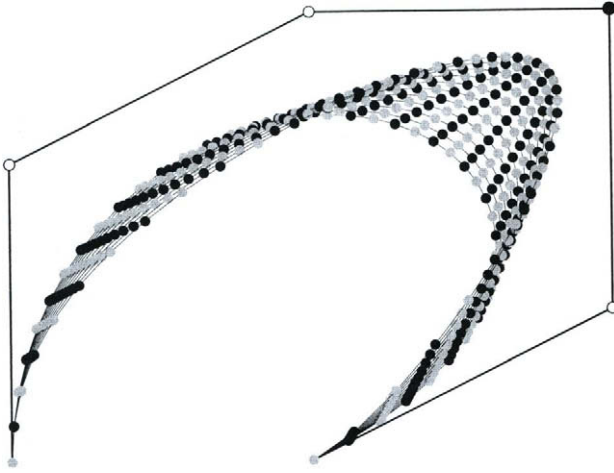


Figure 5.1. The influence of weights on the curve shape. The weight of the black control point is being changed. Note that all the curve points of a fixed parameter lie on a straight line containing the control point with the varying weight.

degree elevation is an instance of a corner cutting algorithm and repeated degree elevation converges to the curve itself, we have proved the variation diminishing property. Obviously, this property carries over directly from the non-rational case and introducing positive weights does not supply enough additional flexibility to violate this property.

There is one important property of a rational Bézier curve that is not shared by its non-rational counterpart: *projective invariance*. The curve will stay invariant under general projective transformations. This property can be exploited in graphics algorithms. Instead of applying a projective or perspective transformation to the curve points while rendering (or sampling) the curve, one can first apply the transformation to the control points, and then render the curve subsequently.

Weights and weight points

The weights w_i can be used as additional shape parameters in the following way. Let us increase a weight w_k whereby the other weights are staying unchanged. Then the α_k in Equation (5.3) increases as well. This means that the control point \mathbf{c}_k is weighted more heavily, hence the curve moves closer to the control point \mathbf{c}_k . This effect is illustrated in Figure 5.1. Furthermore it is illustrated that for a fixed parameter t_0 the points $\mathbf{c}(t_0)$ lie on a straight line for varying weight w_k . An elegant proof of this can be found in [16]. In practice, experienced stylists use the weights to fine-tune the shape of curves and surfaces. Some software packages for example support a dial-box interface which can be used to increase and decrease weights with fine granularity.

G. Farin introduced weight points in [13]. These are often called Farin points. The

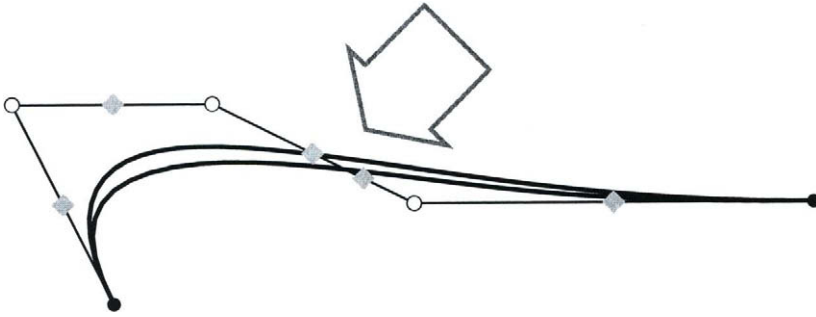


Figure 5.2. Manipulating the curve shape by sliding the weight point (arrow) along its control leg

weight points \mathbf{q}_i are defined as

$$\mathbf{q}_i = \frac{w_i \mathbf{c}_i + w_{i+1} \mathbf{c}_{i+1}}{w_i + w_{i+1}} \quad (5.4)$$

In Figure 5.2 we illustrate the effect of weight points on the curve. The weights are related to the weight points by the following formula:

$$w_{i+1} = w_i * \text{ratio}(\mathbf{c}_i, \mathbf{q}_i, \mathbf{c}_{i+1}), i = 0, \dots, m - 1 \quad (5.5)$$

This equation shows that if we set $w_0 = 1$ without loss of generality, then the weights are uniquely determined by the weight points. Hence, weight points can be used as shape handles as alternative to using the weights themselves. Farin[15] discovered that rational Bézier curves are not only contained within the convex hull formed by their control points. They are also contained within the convex hull formed by the two end control points and the weight points. Hence we obtain tighter bounds. This is a useful property for many algorithms as we will see later.

5.2.2. Derivatives

The computation of the derivative of a rational Bézier curve can be performed by using the quotient rule. For the first derivative we can perform the following manipulations which are equivalent to rewriting the quotient rule:

Define

$$\mathbf{p}(t) = w(t)\mathbf{c}(t) \quad (5.6)$$

Then we can compute

$$\dot{\mathbf{c}}(t) = \frac{\dot{\mathbf{p}}(t) - \dot{w}(t)\mathbf{c}(t)}{w(t)} \quad (5.7)$$

Rewriting the derivatives in this way we only need to compute derivatives of polynomial expressions. By using the Leibniz rule, this carries over to higher order derivatives:

$$\mathbf{c}^{(r)}(t) = \frac{\mathbf{p}^{(r)}(t) - \sum_{i=1}^r \binom{r}{i} w^{(i)}(t)\mathbf{c}^{(k-i)}(t)}{w(t)} \quad (5.8)$$

This formula is recursive and by using this recurrence relation together with equation (5.6), we can compute any derivative by computing the derivative of non-rational Bézier curves. This can be done efficiently using the de Casteljau algorithm (see Chapter 4 on *Bézier Techniques*). It is easy to see that the derivative of a rational Bézier curve can not simply be obtained by computing the derivative of a non-rational Bézier curve in 4D homogeneous space and subsequent projection. This has implications for piecing together such curves. If two rational Bézier curves have a common derivative in homogeneous space then they will have a common derivative in affine space. The opposite however is not true. We will revisit this topic in the section on geometric continuity. The first order derivatives at the two end points of a rational Bézier curve can be computed quite easily: by using our knowledge from the non-rational case we obtain:

$$\begin{aligned} \dot{\mathbf{c}}(0) &= \frac{\dot{\mathbf{p}}(0) - \dot{w}(0)\mathbf{c}(0)}{w(0)} = \frac{mw_1(\mathbf{c}_1 - \mathbf{c}_0)}{w_0} \\ \dot{\mathbf{c}}(1) &= \frac{\dot{\mathbf{p}}(1) - \dot{w}(1)\mathbf{c}(1)}{w(1)} = \frac{mw_{m-1}(\mathbf{c}_m - \mathbf{c}_{m-1})}{w_m} \end{aligned}$$

5.2.3. Fundamental algorithms

In the following we will discuss some of the most basic algorithms which are the foundation for evaluating and manipulating rational Bézier curves. We start with the classical algorithm for evaluating rational curves, the *de Casteljau algorithm*. For rational curves there are two variants of the de Casteljau algorithm. Variant 1 is the straightforward generalization of the non-rational case: The setting here is the projective space and we treat the curve as having homogeneous control points $[w_i\mathbf{c}_i \ w_i]$, $i = 0, \dots, m$.

de Casteljau Algorithm I

Given: Homogeneous control points $\hat{\mathbf{c}}_i := [w_i\mathbf{c}_i \ w_i]$, $i = 0, \dots, m$ and parameter $t \in [0, 1]$.
Find: Point $\mathbf{c}(t)$ on the curve in affine space.

Compute

$$\hat{\mathbf{c}}_i^r(t) = (1 - t)\hat{\mathbf{c}}_i^{r-1} + t\hat{\mathbf{c}}_{i+1}^{r-1} \quad \begin{cases} r = 1, \dots, m \\ i = 0, \dots, m - r. \end{cases} \tag{5.9}$$

$$\mathbf{c}(t) = \pi(\hat{\mathbf{c}}_0^m(t)) \tag{5.10}$$

Here π is the projection operator. We apply π to the curve $\hat{\mathbf{c}}(t)$ by applying π to every control point.

The second variant of the de Casteljau algorithm consists of projecting every intermediate point into affine space (see Farin [13]) and is therefore more geometric:

de Casteljau Algorithm II

Given: Homogeneous control points $\hat{\mathbf{c}}_i := [w_i\mathbf{c}_i \ w_i]$, $i = 0, \dots, m$ and parameter $t \in [0, 1]$.
Find: Point $\mathbf{c}(t)$ on the curve in affine space.

Compute

$$\mathbf{c}_i^r(t) = \frac{(1 - t)w_i^{r-1}\mathbf{c}_i^{r-1} + tw_{i+1}^{r-1}\mathbf{c}_{i+1}^{r-1}}{(1 - t)w_i^{r-1} + tw_{i+1}^{r-1}} \quad \begin{cases} r = 1, \dots, m \\ i = 0, \dots, m - r. \end{cases} \tag{5.11}$$

$$w_i^r(t) = (1-t)w_i^{r-1}(t) + tw_{i+1}^{r-1}(t) \quad (5.12)$$

$$\mathbf{c}(t) = \mathbf{c}_0^m(t) \quad (5.13)$$

Floater [18] points out that the derivative formulae in the previous section do not yield any geometric insight. He rewrites the first derivatives using the intermediate weights and points from the de Casteljau algorithm. One advantage of this formulation is the ease by which one can derive upper bounds for the derivatives of a curve. This is important for many divide and conquer algorithms in practice. For example, when computing curve-curve intersections, one often needs to decide quickly how flat a given segment is.

The next important algorithm is the reparameterization algorithm. It is common to transform rational Bézier curves into a *standard representation*. This means that the two end weights w_0 and w_m are both 1. It is known (see [14]) that two rational Bézier curves \mathbf{c} and $\tilde{\mathbf{c}}$ describe the same shape if their weights are related by $w_i = f^i \tilde{w}_i$, where f is an arbitrary scalar. Since we can scale arbitrarily, it is obvious how to achieve one weight being equal to 1. Let us now assume that $w_0 = 1$. If we set

$$f = \left(\frac{w_0}{w_m}\right)^{\frac{1}{m}} \quad (5.14)$$

we have achieved that $w_m = w_0$. Dividing by w_0 yields the curve in standard form. It should be noted that changing the parameterization of the curve in this manner has practical implications. Figure 5.3 shows how the parameter spacing is changed. In practice, one often evaluates curves in equal parameter increments just as in Figure 5.3. This indicates that the parameterization has to be taken into account to avoid sampling artifacts. In [10], a projective parameterization was introduced which allows an evenly spaced sampling of a full circle. On the other hand, this also shows that an injudicious choice of weight factors can substantially skew the parameterization of a curve.

Another useful technique is degree elevation. This is the process of raising the degree of a m -degree curve to a larger degree n . This technique is useful in practice for building a surface from cross sections; this process is called lofting. In order for the resulting loft surface to interpolate the sections, all sections have to be of identical degree. One way to achieve this is to perform degree elevation. The degree elevation algorithm for rational curves is a straightforward generalization of the non-rational case. Its setting is the projective space with homogeneous control points.

Degree Elevation

Given: A Bézier curve $\mathbf{c}(t)$ of degree m with homogeneous control points $\hat{\mathbf{c}}_i := [w_i \mathbf{c}_i \ w_i]$.
Find: The Bézier curve $\mathbf{d}(t)$ of degree $m+1$ such that $\mathbf{d} \equiv \mathbf{c}$ on $[0, 1]$.

Compute

$$\hat{\mathbf{d}}_i = \frac{i}{m+1} \hat{\mathbf{c}}_{i-1} + \left(1 - \frac{i}{m+1}\right) \hat{\mathbf{c}}_i, \quad i = 0, \dots, m+1. \quad (5.15)$$

Here we set $\mathbf{c}_{-1} = \mathbf{c}_{m+1} = 0$. The interplay between degree elevation and reparameterization has been investigated in [14]. One can first degree elevate a given curve \mathbf{c} in standard form to obtain a new curve $\tilde{\mathbf{c}}$. Then one can reparameterize the curve \mathbf{c} and subsequently degree-elevate this curve. Bringing the degree-elevated curve into standard form yields a

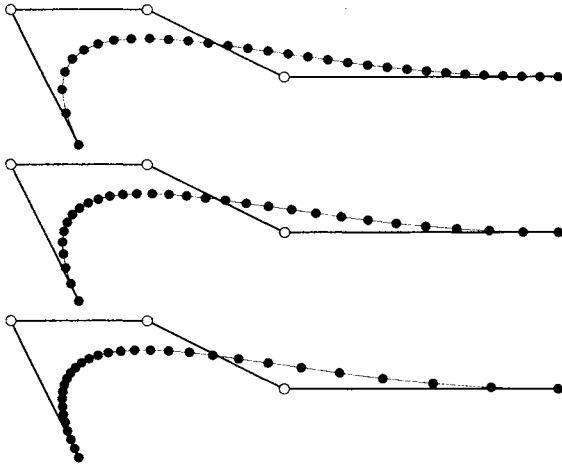


Figure 5.3. Effect of parameterization on sampling density, $f = 2$ (top), $f = 1$ (middle), and $f = 0.5$ (bottom).

curve \tilde{c} . The curves \hat{c} and \tilde{c} describe the same curve but have different control polygons and weights. This is a distinction from the non-rational case where the degree-elevated representation is unique.

In the polynomial case it is easy to tell if a curve of degree m is actually of degree $m - 1$. One just needs to check that the m -th derivative is identically 0. This is equivalent to check that the m -th difference of Bézier points is equal to 0. The same approach can be taken for rational curves in projective space.

As mentioned in Chapter 4 on *Bézier Techniques*, one is often interested in the opposite direction as well; that means one would like to find a curve of a lower degree which represents the given curve of higher degree. This problem often occurs in practice, when one needs to exchange data between different CAD systems. Some CAD systems can handle curves and surfaces of higher degrees than other systems. It is easily seen that in most cases there will be no exact solution. Eck [11,12] and many others gave algorithms for degree reducing non-rational Bézier curves. There are not many algorithms for degree reducing rational curves specifically. Assuming that we have an algorithm for non-rational curves we can try to apply this algorithm to the homogenous coordinates. However it is not guaranteed that the weights remain positive.

5.2.4. Conics

Conic sections can be represented exactly as a rational quadratic Bézier curve. Without loss of generality, we can assume that the two weights w_0 and w_1 are equal to 1. This means that a conic section $c(t)$ has the representation

$$c(t) = \frac{\mathbf{b}_0 B_0^2(t) + w_1 \mathbf{b}_1 B_1^2(t) + \mathbf{b}_2 B_2^2(t)}{B_0^2(t) + w_1 B_1^2(t) + B_2^2(t)} \tag{5.16}$$

with $t \in [0, 1]$. For a conic in this representation it is always true that the shoulder point of the conic is at parameter value $t = 1/2$. Furthermore, the tangent at the shoulder point, often called the shoulder tangent, is parallel to the line connecting \mathbf{b}_0 and \mathbf{b}_2 . Another interesting fact is that by reversing the sign of w_1 we obtain the complementary conic segment $\hat{c}(t)$. It is easy to show that the three points \mathbf{b}_1 , $c(t)$ and $\hat{c}(t)$ are always collinear. It is well known, that in affine space, there are three classes of conics: hyperbolas, parabolas and ellipses. It is now natural to ask if this classification can be performed based on the above representation. Since hyperbolas have two singularities and parabolas only one, it is also intuitively clear that the weight w_1 plays a crucial role, because this is the only parameter affecting the denominator. It turns out that indeed we can classify conics by inspecting the weight w_1 , or more precisely, solve the quadratic equation given by the denominator of the complementary segment:

$$t_{1,2} = \frac{1 + w_1 \pm \sqrt{w_1^2 - 1}}{2 + 2w_1} \quad (5.17)$$

One sees that for $w_1 > 1$ we have two zeros and hence two singularities, yielding a hyperbola. The case $w_1 < 1$ yields an ellipse and for $w_1 = 1$ we obtain a parabola. The latter is obvious, since this means that the curve is a non-rational quadratic, hence it must be a parabola. We will now look at some specific constructions. The first is a circular segment that is a special case of an ellipse. We know that the weight w_1 must be greater than 0 and smaller than 1. Due to the symmetry we also know that the control points \mathbf{b}_0 , \mathbf{b}_1 , and \mathbf{b}_2 must form an isosceles triangle. It turns out that $w_1 = \cos(\alpha)$ where α is the angle formed by the lines $\overline{\mathbf{b}_0\mathbf{b}_1}$ and $\overline{\mathbf{b}_0\mathbf{b}_2}$ (see Figure 5.4). In order to obtain a full circle, one needs to piece together several segments, for example one could piece together 3 segments each with angle $\alpha = 60$ degrees. Another construction that is often used in practice, in particular for constructing blend surfaces is the so-called ρ -conic (see [30]). Here we prescribe the two endpoints \mathbf{b}_0 and \mathbf{b}_2 as well as the end tangents hereby defining the point \mathbf{b}_1 . Additionally, the shoulder point \mathbf{p} at $t = 1/2$ is restricted to lie on the line joining \mathbf{b}_1 and the midpoint \mathbf{p}_m of $\overline{\mathbf{b}_0\mathbf{b}_2}$. This means that $\mathbf{p} - \mathbf{p}_m = \rho(\mathbf{b}_1 - \mathbf{p}_m)$ where ρ is the parameter sliding the point up and down the line and hence pulling the conic closer or farther away from \mathbf{p}_1 . One could say that ρ describes the "fullness" of the curve. It turns out that the corresponding weight w_1 can be computed as

$$w_1 = \frac{\tau_1}{2\sqrt{\tau_0\tau_2}} \quad (5.18)$$

where the τ_i are the barycentric coordinates of \mathbf{p} with respect to the triangle formed by the control points. Note that this equation is always solvable if \mathbf{p} lies inside the triangle, but it might not be solvable otherwise. The construction described above is a special case of the classic construction of a conic from two points and their tangents and an additional point. More on conics in Bézier form can be found in [30].

5.3. RATIONAL B-SPLINE CURVES

In Chapter 6 on *Spline Basics*, the basic theory of splines as pioneered by Schoenberg [39] has been developed. Just as is the case for Bézier techniques, we can generalize non-rational B-Splines to the rational case. This defines non-uniform rational B Splines

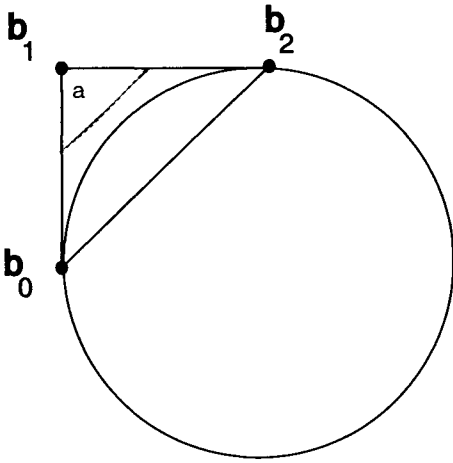


Figure 5.4. Construction of a circular segment in rational Bézier form.

(NURBS) which are the current industry standard. Here we will deal with rational B-Spline curves, so we present the basic definitions first.

5.3.1. Basic definitions

A NURBS curve of degree m is given by:

control points also known as *de Boor points* $\mathbf{d}_i, i = 0, \dots, n, \mathbf{d}_i \in \mathbb{R}^k, k = 2, 3$.

weights $w_i, i = 0, \dots, n$,

knot vector $\tau = \{t_0, \dots, t_{n+m+1}\}$.

We assume that the first and last knots have multiplicity $m+1$ each. This is industry-wide convention and it ensures end point interpolation.

A rational NURBS curve $\mathbf{d}(t)$ is defined as

$$\mathbf{d}(t) = \frac{\sum_{i=0}^n w_i \mathbf{d}_i N_i^m(t)}{\sum_{i=0}^n w_i N_i^m(t)} \tag{5.19}$$

The normalized B-Spline basis functions are defined recursively:

$$N_i^m(t) = \frac{t - t_i}{t_{i+m} - t_i} N_i^{m-1}(t) + \frac{t_{i+m+1} - t}{t_{i+m+1} - t_{i+1}} N_{i+1}^{m-1} \quad m \geq 1; \quad i = 0, \dots, n \tag{5.20}$$

where

$$N_i^0(t) = \begin{cases} 1 & \text{for } t_i \leq t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{5.21}$$

The properties of these basis functions are discussed in detail in Chapter 6 on *Spline Basics*. We can now proceed as in the Bézier case and define the curve as a linear

combination of control points:

$$\mathbf{d}(t) = \sum_{i=0}^n \alpha_i \mathbf{d}_i \quad (5.22)$$

with

$$\alpha_i = \frac{w_i N_i^m(t)}{\sum_{i=0}^n w_i N_i^m(t)} \quad (5.23)$$

This leads immediately to the following properties:

- Affine Invariance
- Convex hull property when the weights w_i are nonnegative.
- Variation diminishing property.

The degree elevation and knot insertion algorithms can again be viewed as examples of corner-cutting algorithms and this can be used to prove the variation diminishing property just as in the Bézier case. Furthermore, we can easily deduce that the NURBS curve as defined above interpolates the endpoints:

$$\mathbf{d}(t_0) = \mathbf{d}_0 \quad (5.24)$$

$$\mathbf{d}(t_{m+n+1}) = \mathbf{d}_n \quad (5.25)$$

The effect of the weights on the curve is similar to Bézier curves. Increasing the weight w_i relative to its neighbors causes the curve to move towards the control point \mathbf{d}_i .

5.3.2. Derivatives

In order to compute derivatives of rational B-spline curves, we proceed in the same manner as for rational Bézier curves. We define

$$\mathbf{p}(t) = w(t)\mathbf{d}(t) \quad (5.26)$$

where $w(t)$ is the 1D B-Spline weight function, and again arrive at the equation

$$\dot{\mathbf{d}}(t) = \frac{\dot{\mathbf{p}}(t) - \dot{w}(t)\mathbf{d}(t)}{w(t)} \quad (5.27)$$

Now we just need to recall how to compute derivatives of a non-rational B-Spline curve $\mathbf{p}(t)$.

$$\dot{\mathbf{p}}(t) = m \sum_{i=0}^{n-1} \frac{\mathbf{d}_{i+1} - \mathbf{d}_i}{t_{m+i+1} - t_{i+1}} N_{i+1}^{m-1}(t) \quad (5.28)$$

Higher order derivatives can be computed by using the recurrence relation for the B-Spline basis functions. A different approach for computing the first derivative of a NURBS

curve has been developed by Floater [19]. Assuming that the parameter t is contained in $[t_I, t_{I+1}]$, and using

$$\alpha_{i,k} = (u - u_i)/(u_{i+1+m-k} - u_i) \quad (5.29)$$

$$w_{i,k} = (1 - \alpha_{i,k})w_{i-1,k-1} + \alpha_{i,k}w_{i,k-1} \quad (5.30)$$

one can show that the NURBS curve $\mathbf{d}(t)$ obeys:

$$\dot{\mathbf{d}}(t) = \sum_{i=1}^n L_i(t)(\mathbf{d}_i - \mathbf{d}_{i-1}) \quad (5.31)$$

with

$$L_i(t) = \frac{1}{w_{I+1,m}(t)} \sum_{j=0}^{i-1} \sum_{k=i}^n (\dot{N}_k^m(t)N_j^m(t) - N_k^m(t)\dot{N}_j^m(t))w_jw_k \quad (5.32)$$

Floater uses this expression to derive upper bounds for the derivatives.

5.3.3. Fundamental algorithms

In Chapter 6 on *Spline Basics* different algorithms for evaluating B-Spline curves have been presented. One can directly make use of the recurrence relation. Alternatively one can use knot insertion or the Oslo algorithm to evaluate a spline. The most commonly used algorithm is the de Boor algorithm. This algorithm can be viewed as a generalization of the de Casteljau algorithm. Again we can give two versions, the first version using homogeneous coordinates:

de Boor Algorithm I

Given: Homogeneous control points $\hat{\mathbf{d}}_i := [w_i \mathbf{d}_i \ w_i]$, $i = 0, \dots, n$, knot vector $\tau = \{t_0, \dots, t_{m+n+1}\}$ and parameter t .

Find: Point $\mathbf{d}(t)$ on curve in affine space.

Compute I such that $t \in [t_I, t_{I+1}] \subset [t_m, t_{n+1}]$.

$$\hat{\mathbf{d}}_i^r(t) = \frac{t_{i+n-k} - t}{t_{i+n-k} - t_{i-1}} \hat{\mathbf{d}}_{i-1}^{k-1}(t) + \frac{t - t_{i-1}}{t_{i+n-k} - t_{i-1}} \hat{\mathbf{d}}_i^{k-1}(t) \quad (5.33)$$

with $k = 1, \dots, n - r$ and $i = I - n + k + 1, \dots, I - r + 1$.

$$\mathbf{d}(t) = \pi(\hat{\mathbf{d}}(t)) = \hat{\mathbf{d}}_{I-r+1}^{n-r}(u). \quad (5.34)$$

Here we used the convention that r denotes the multiplicity of t in case t is an element of the knot vector, and 0 if t is not contained in the knot vector. In analogy to the Bézier case we can define a rational version of the de Boor algorithm:

de Boor Algorithm II

Given: Homogeneous control points $\hat{\mathbf{c}}_i := [w_i \mathbf{d}_i \ w_i]$, $i = 0, \dots, n$, knot vector $\tau = \{t_0, \dots, t_{m+n+1}\}$ and parameter t .

Find: Point $\mathbf{d}(t)$ on curve in affine space.

Compute: I such that $t \in [t_I, t_{I+1}] \subset [t_m, t_{n+1}]$.

$$\mathbf{d}_i^r(t) = \left(\frac{t_{i+n-k} - t}{t_{i+n-k} - t_{i-1}} w_{i-1}^{k-1} \mathbf{d}_{i-1}^{k-1}(t) + \frac{t - t_{i-1}}{t_{i+n-k} - t_{i-1}} w_i^{k-1} \mathbf{d}_i^{k-1}(t) \right) / w_i^k \quad (5.35)$$

$$w_i^k = \frac{t_{i+n-k} - t}{t_{i+n-k} - t_{i-1}} w_{i-1}^{k-1} + \frac{t - t_{i-1}}{t_{i+n-k} - t_{i-1}} w_i^{k-1} \quad (5.36)$$

with $k = 1, \dots, n-r$ and $i = I-n+k+1, \dots, I-r+1$. As pointed out before, this version uses convex combinations and hence is more stable. On the other hand the operations count is increased.

Reparameterizations

It is possible to reparameterize a NURBS curve. The key observation is due to Lee and Lucian [31]: If we apply a rational linear (Möbius) parameter transformation to a NURBS curve, the resulting curve is again a NURBS curve with the same control polygon but different weights and knots. More precisely: Define the linear rational map ϕ by

$$\phi(t) = \frac{\alpha t + \beta}{\gamma t + \delta} \quad (5.37)$$

If we apply this transformation to a NURBS curve with knot vector τ we obtain a curve with knot vector $\zeta = \{s_0, \dots, s_n\}$ and $s_i = \phi(t_i)$. The new weights are given by

$$\tilde{w}_i = w_i / \prod_{j=1}^m (\gamma t_{i+j} + \delta) \quad (5.38)$$

Since the rational linear map has 3 variables (set $\alpha = 1$ without loss of generality), we can now pick the unknowns in such a way that \tilde{w}_0 and \tilde{w}_n are equal. This leads to a NURBS curve in standard form (see [31]). Conversely, one could also try to determine the unknowns in such a way that intermediate knots are mapped to certain target values. However, this seems to be of limited use since in most cases one would have to solve a least squares system and the new weights are unbounded.

5.4. GEOMETRIC CONTINUITY FOR RATIONAL CURVES

In this section we will address the topic of geometric continuity for rational curves. If we recall how we derived the derivatives of a rational Bézier curve, it is easily seen that C^1 continuity in projective space implies C^1 continuity in affine space but the reverse is not true. We abbreviate continuity in projective space with HC^k or HG^k respectively, whereas continuity in affine space is denoted by C^k or G^k . Let us start with two Bézier curves of degree m where the first curve has a control polygon $\mathbf{c}_0, \dots, \mathbf{c}_m$, weights w_0, \dots, w_m and is defined over the interval $[u_0, u_1]$. The second curve has control points $\mathbf{c}_m, \dots, \mathbf{c}_{2m}$ and weights w_m, \dots, w_{2m} and is defined over the interval $[u_1, u_2]$. Furthermore, we define $\delta_0 := u_1 - u_0$ and $\delta_1 := u_2 - u_1$. If we recall the results in Section 5.2.2, it turns out that the two curves are C^1 continuous at u_1 if

$$\frac{w_{m-1}}{\delta_0} (\mathbf{c}_m - \mathbf{c}_{m-1}) = \frac{w_{m+1}}{\delta_1} (\mathbf{c}_{m+1} - \mathbf{c}_m). \quad (5.39)$$

One can also see that for G^1 continuity the weights do not play any role at all. In [5] conditions for curvature and torsion continuity of rational Bézier curves have been derived. These conditions have nice geometric interpretations: Let us define the following

quantities:

$$a_+ = \|\mathbf{c}_{m+1} - \mathbf{c}_m\| \tag{5.40}$$

$$a_- = \|\mathbf{c}_{m-1} - \mathbf{c}_m\| \tag{5.41}$$

$$h_+ = \|\mathbf{c}_{m+2} - T_+\| \tag{5.42}$$

$$h_- = \|\mathbf{c}_{m-2} - T_-\| \tag{5.43}$$

$$d_+ = \|\mathbf{c}_{m+3} - O_+\| \tag{5.44}$$

$$d_- = \|\mathbf{c}_{m-3} - O_-\| \tag{5.45}$$

Here T_+ is the tangent defined by \mathbf{c}_m and \mathbf{c}_{m+1} , T_- is the corresponding tangent defined by \mathbf{c}_m and \mathbf{c}_{m-1} . Furthermore, O_+ is the osculating plane spanned by \mathbf{c}_m , \mathbf{c}_{m+1} , and \mathbf{c}_{m+2} , whereas O_- is the osculating plane spanned by \mathbf{c}_m , \mathbf{c}_{m-1} , and \mathbf{c}_{m-2} . The two curves are curvature continuous or G^2 continuous if they are C^1 continuous, the 5 points $\mathbf{c}_{m-2}, \dots, \mathbf{c}_{m+2}$ are coplanar, \mathbf{c}_{m-2} and \mathbf{c}_{m+2} lie on the same side of tangent T_+ (which is equal to T_- in that case) and the following relationship holds:

$$\frac{w_m w_{m+2} h_+}{w_{m-1}^2 a_+^2} = \frac{w_m w_{m-2} h_-}{w_{m-1}^2 a_-^2} \tag{5.46}$$

For a non-planar rational Bézier curve, the conditions for torsion continuity are also of practical interest. Two curvature continuous rational Bézier curves are torsion continuous at \mathbf{c}_m if \mathbf{c}_{m-3} and \mathbf{c}_{m+3} lie in different half spaces defined by the osculating plane O_+ (by assumption O_+ equals O_-), and the distances obey

$$\frac{d_+}{d_-} = \frac{a_+ h_+}{a_- h_-} \frac{w_{m-3} w_{m+1} w_{m+2}}{w_{m-1} w_{m-2} w_{m+3}} \tag{5.47}$$

In Figure 5.5 we illustrate the geometry and the quantities involved for defining curvature and torsion continuity, including the G^1 condition. In [5], Boehm also presents construction algorithms for curvature continuous cubic B-Splines as well as torsion continuous quartic B-Splines. The algorithms make use of the interplay of Bézier control points and de Boor points. The de Boor points \mathbf{d}_i can be derived from the intersection of segments formed by the Bézier control points.

One should note that the above conditions and constructions are valid in affine space but they do not guarantee homogeneous continuity. For this to achieve one has to investigate continuity in projective space using homogeneous coordinates. Degen [7] derived conditions for G^r continuity using homogeneous coordinates. He showed that for two curves to be G^r continuous, there needs to exist a triangular matrix which transforms the control points from the first curve directly into the control points for the second curve. Furthermore, he gives a recurrence formula for G^r conditions with arbitrary r .

5.5. RATIONAL CURVE APPROXIMATION AND INTERPOLATION

In practical applications, one of the most common and important tasks is to approximate or interpolate point data. Furthermore, one often needs to approximate curves which are given in analytical form but are not representable exactly as NURB curve. Examples of these are intersection curves or offset curves. It is also a fact that in most commercial

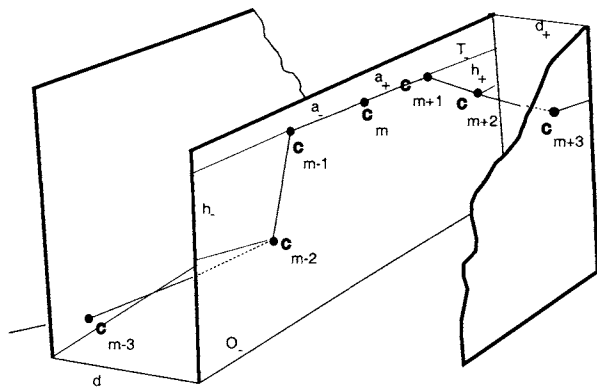


Figure 5.5. Construction of curvature and torsion continuous curves. We depict all the control points and scalar quantities (distances) that are involved.

software packages, the weights are not used when approximating or interpolating with NURB curves, it would be more accurate to talk about *NUB* curves. It remains the case that the main justification for using true rational curves in practice is their ability to represent conics and circles exactly. Their use is mainly limited for that purpose. The reason why the weights are not utilized for fitting is twofold:

- 1) Treating weights and control points as unknowns immediately requires the solution of a nonlinear problem.
- 2) The desired range of values that weights should attain is rather restrictive. Weights should be positive, be bounded away from zero and also have a reasonable upper bound when we assume standard form. More precisely, one often prescribes weights to be in the range $f * [0.5, 2.0]$ where f is a common factor.

Nonetheless, algorithms have been developed to use the weights for fitting. We will present the relevant work below.

5.5.1. Rational curve interpolation

The direct approach to rational curve interpolation is to consider data points in projective space and then interpolate in homogeneous coordinates just as can be done in the nonrational case: Given are data points \mathbf{p}_i and weights v_i and corresponding parameters t_i , $i = 0, \dots, n$. In 4D we compute a rational spline curve $\mathbf{c}(t)$ such that $\mathbf{c}(t_i)^T = [v_i \mathbf{p}_i \ v_i]$. This problem can be solved analogously to the non-rational case. The heuristics for choosing knots can also be taken from the nonrational case. The problem is however somewhat ill-posed, since in practice we usually have only data points but not the weights. There

is no algorithm for choosing weights; in addition, solving this problem globally can easily produce weight functions with singularities. A different and more practical approach has been investigated by Schneider in his Ph.D. thesis [37]. Here, the weights are treated as additional unknowns; this allows to specify more data points. Concretely, it turns out that if we have $m + 1$ data points, a curve with $n + 1$ de Boor points can be determined uniquely if the relation $n = 3m/4$ holds. The interpolation problem is then solved using homogeneous coordinates. One of the remaining problems is that the resulting interpolation curves can exhibit singularities. Even more interestingly, the parameterization has an effect on this, there are cases where by using chord length parameterization the curve exhibits singularities whereas by using centripetal parameterization the curve has no singularities, however it interpolates the same data. A practical concern is that many times curves are constructed as building blocks for surfaces. This poses severe limitations on the parameterization. As evidenced above, the rational interpolation problem is very sensitive to the parameterization, even more so than in the non-rational case.

For reasons outlined above the more practical interpolation methods are those which are based on *osculatory* interpolation. The basic idea here is that one pieces together locally fitted pieces with prescribed continuity. For example, let us assume we have data points \mathbf{p}_i , tangents \mathbf{q}_i , and curvature values κ_i , or alternatively curvature vectors K_i , $i = 0, \dots, n$. We parameterize the data yielding parameters t_i . We can now fit say rational cubic Bézier curve segments \mathbf{c}_i such that $\mathbf{c}_i(0) = \mathbf{p}_i$, $\mathbf{c}_i(1) = \mathbf{p}_{i+1}$ and furthermore the unit tangent of \mathbf{c}_i at 0 and 1 equals \mathbf{q}_i and \mathbf{q}_{i+1} respectively. In addition, the curvature of \mathbf{c}_i at 0 equals κ_i and at 1 it equals κ_{i+1} . Hoellig proved in [23] that such a curve exists under certain restrictions on the geometry induced by the data. Furthermore, he gave a method to compute such a rational spline curve. Note that this construction is closely related to the continuity conditions we derived earlier. It turns out that this method has approximation order 6 which is an improvement over non-rational methods which can only achieve order 4 in the general case. Another popular approach to osculatory interpolation is to use piecewise conic segments. This method is constrained to planar data and makes use of the fact that 5 data points determine a conic uniquely. One approach is to fit conics locally to five data points each and piece the segments together. In general the resulting spline will only be C^0 continuous. A more useful approach is to prescribe three data points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ and two end tangents \mathbf{q}_0 and \mathbf{q}_1 . If the middle point \mathbf{p}_1 is contained within the triangle formed by $\mathbf{p}_0, \mathbf{p}_2$ and the intersection of the two tangents \mathbf{q}_0 and \mathbf{q}_1 , then there is a solution. The resulting interpolating spline is convexity preserving and G^1 continuous. Schaback [36] gives a method to construct these segments. It is also possible to prescribe curvature values κ_0 and κ_1 at the endpoints of a segment together with three data points. In general these constraints can not be satisfied, in [36] Schaback gives conditions that ensure the existence of a unique interpolant. It turns out that these conditions are satisfied if one samples dense enough a smooth regular curve with non-vanishing curvature. Goodman et al. [22] presented a method for fitting planar data with piecewise cubic rational Bézier segments. Their goal was to compute shape-preserving interpolants: if all data points lie on one side of a given line, then the interpolant should not cross that line. Their method proceeds by first computing a G^2 continuous rational cubic spline matching two positional and two curvature constraints at the ends. If one of the spline segments violates the line condition, the two interior weights are modified

until the curve just touches the line. Obviously this will cause a G^2 discontinuity with the neighboring segment and so the weight of the neighboring segment needs to be modified as well.

Note that the osculating interpolation methods can also be viewed as approximation methods, in the case that the data is sampled off a given curve that needs to be fitted by NURBS.

5.5.2. Rational curve approximation

The classical scheme for performing approximation with rational curves is the Padé scheme. Let us briefly go back to the functional setting. We assume that we have a function $F(x)$ that has a Taylor expansion

$$F(x) = \sum_{i=0}^{\infty} a_i x^i \quad (5.48)$$

We are interested in a rational polynomial function $R(x)$ that approximates F . R is of the form

$$R(x) = \frac{a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n}{b_0 + b_1 x + b_2 x^2 + \dots + b_m x^m} \quad (5.49)$$

By matching degrees of freedom we can easily see that we have $m + n + 1$ free coefficients (b_0 can be normalized to 1) and hence the best approximation order one can hope for is $\mathcal{O}(n + m + 1)$. One computes the approximant by multiplying both sides by the denominator of R and obtains a system of equations that can be solved uniquely if the Hankel determinant of the denominator of R is nonzero. This gives rise to many interesting theoretical investigations. An introduction can be found in [6], other references are [1,21]. For CAGD applications, it is more useful to use multi-point approximation schemes, in particular the two-points schemes that have been introduced in the previous section. We can construct a parametric Padé approximant that approximates a given function up to a certain order at two parameter values [45]. Here is the problem statement for approximating a curve at two points up to order 2:

Given: A parametric curve $\mathbf{F}(t) = (x(t), y(t), z(t))$.

Find: A rational Bézier curve $\mathbf{b}(t)$ such that

$$\mathbf{F}(t) - \mathbf{b}(t) = \mathcal{O}(t - t_0)^3 \quad (5.50)$$

$$\mathbf{F}(t) - \mathbf{b}(t) = \mathcal{O}(t - t_1)^3 \quad (5.51)$$

where $t \in [t_0, t_1]$. There are several strategies for solving this problem. For example one could solve two one-point problems and blend the results, or one could solve three one-dimensional two-point problems and subsequently multiply to make the denominators equal. In [45] a two-points problem was solved for all three components simultaneously. This allows to use a rational polynomial of degree 4 over 4. This is of lower degree than was possible before ([3]). Results show that this approximation method is very effective and that high quality C^2 approximations can be obtained.

A variety of different approximation schemes can be found in the literature. The most general scheme treats control points, weights and nodes as free and solves a non-linear

optimization problem. Obviously, it will be challenging to avoid getting stuck in a local minimum. Other more specialized methods can be found for example in [2,44]. In [46,8] some geometrically motivated methods are presented to approximate smooth curves by rational Bézier curves. Finally, a somewhat different application is the fairing of curves. Here, one is interested in smoothing a curve, most often one would like a smooth change in curvature without spikes. In [24], a method is developed that automatically adjusts the weights of a B-Spline curve. The goal is to produce a curve with smoother curvature variation that is still very close in position to the original curve. Another approach for selecting weights is given in [34].

5.6. RATIONAL BÉZIER SURFACES

In this section we will introduce rational Bézier patches. In practice, one of the most challenging problems is the composition of multiple patches. In most cases, one wants to maintain at least G^1 continuity to avoid creases. It turns out that this is particularly difficult for rational patches. Hence we will devote a special section to this subject. Furthermore, we will see, that many algorithms for surfaces can be reduced to applying the curve algorithms just as is commonly done for non-rational Bézier patches. However this is not easily possible for approximation algorithms.

5.6.1. Basic definitions

I will briefly introduce rational Bézier patches along with their most important properties; assuming the reader to be familiar with rational curves. A rational Bézier surface of degree m, n is given by an array of control points $\mathbf{c}_{ij}, i = 0, \dots, m, j = 0, \dots, n \in \mathbb{R}^3$, and an array of corresponding weights w_{ij} . Furthermore we assume a parameterization in s and t where without loss of generality s and t vary from 0 to 1. The patch is defined as:

$$\mathbf{p}(s, t) = \frac{\sum_{j=0}^n (\sum_{i=0}^m w_{ij} \mathbf{c}_{ij} B_i^m(s)) B_j^n(t)}{\sum_{j=0}^n (\sum_{i=0}^m w_{ij} B_i^m(s)) B_j^n(t)} \quad (5.52)$$

We can omit the parentheses in Equation (5.52), I included them for clarity. In the nonrational case, the patch basis functions obey the tensor product condition:

$$F_{ij}(s, t) = G_i(s) H_j(t) \quad (5.53)$$

In the rational case the basis functions are

$$F_{ij}(s, t) := \frac{w_{ij} B_i^m(s) B_j^n(t)}{\sum_{i=0}^m \sum_{j=0}^n w_{ij} B_i^m(s) B_j^n(t)} \quad (5.54)$$

Due to the sum in the denominator a factorization is not possible. This has practical consequences: In order to exploit the tensor product structure, we have to apply the algorithms in projective space using 4 dimensional homogeneous coordinates. Afterwards we have to apply the projection operator π to return into affine space. But one needs to be careful since this approach is not always valid: Remember that for instance derivatives are not correctly computed using this method. We can generalize some of the properties for non-rational patches directly to the rational case:

- Affine invariance

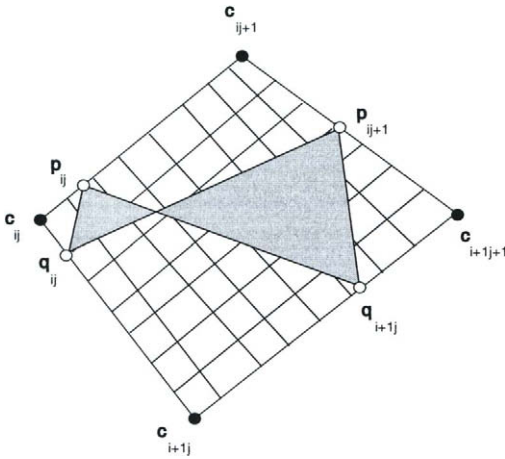


Figure 5.6. Weight points for rational Bézier patches: The shaded triangles need to be coplanar.

- Corner points interpolation
- Convex hull property

Weights and weight points

By a similar reasoning to the curve case, we can deduce that increasing the value of weight w_{ij} relative to its neighbors causes the surface to move towards control point c_{ij} . Hence, we can use the weights again as shape parameters. It is also possible to define weight points. We define these points in the s and in the t direction:

$$p_{ij} = \frac{w_{ij}c_{ij} + w_{i+1j}c_{i+1j}}{w_{ij} + w_{i+1j}} \tag{5.55}$$

$$q_{ij} = \frac{w_{ij}c_{ij} + w_{ij+1}c_{ij+1}}{w_{ij} + w_{ij+1}} \tag{5.56}$$

Considering the fact, that the weights in s direction and in t direction overlap, it is clear that the p_{ij} and q_{ij} are not independent of each other. In fact, the relationship is rather constrained: The 4 points p_{ij} , p_{i+1j} , q_{ij} and q_{i+1j} have to be coplanar, see Figure 5.6. Hence it is somewhat more awkward to provide an interface where it is possible to modify surface shape by specifying weight points. A solution to this problem has been proposed by Theisel [42]. There, extended Farin points are first defined in the surface domain and then mapped into 3D. In addition, the author presents a scheme which allows only a subset of the Farin points to be moved in order to avoid conflicting definitions.

5.6.2. Derivatives

Computing partial derivatives of rational Bézier patches is somewhat more complex. I will first give the straightforward derivation and subsequently, I will describe an algorithm by Sederberg that sacrifices the numerical stability of the Bézier basis for a more efficient evaluation of the derivative. For the straightforward version we will proceed very similar to the curve case. Define

$$\mathbf{v}(s, t) = w(s, t)\mathbf{p}(s, t) \quad \text{with} \quad w(s, t) = \sum_{i=0}^m \sum_{j=0}^n w_{ij} B_i^m(s) B_j^n(t) \tag{5.57}$$

Then we have for the two partial derivatives

$$p_s(s, t) = \frac{\mathbf{v}_s(s, t) - w_s(s, t)\mathbf{p}(s, t)}{w(s, t)} \tag{5.58}$$

$$p_t(s, t) = \frac{\mathbf{v}_t(s, t) - w_t(s, t)\mathbf{p}(s, t)}{w(s, t)} \tag{5.59}$$

$$p_t(s, t) = \frac{\mathbf{v}_t(s, t) - w_t(s, t)\mathbf{p}(s, t)}{w(s, t)} \tag{5.60}$$

For higher order partial derivatives, we can again apply Leibniz' rule and obtain a formula analogous to (5.8). We can also use the approach above to compute mixed derivatives. The general formula is

$$\frac{\partial}{\partial s^k \partial t^l} \mathbf{q}(s, t) = w(s, t) \frac{\partial}{\partial s^k \partial t^l} \mathbf{p}(s, t) + \sum_{\substack{i=0, j=0 \\ i+j \neq 0}}^{k, l} \binom{k}{i} \binom{l}{j} \frac{\partial}{\partial s^i \partial t^j} w(s, t) \frac{\partial}{\partial s^{k-i} \partial t^{l-j}} \mathbf{p}(s, t) \tag{5.62}$$

This expression can easily be transformed to yield the desired mixed derivative of $\mathbf{p}(s, t)$. The two first order partial derivatives need to be evaluated repeatedly for rendering a Bézier patch. More precisely, we need the normal vector of the patch and hence we only need the tangent directions but not the derivative magnitude. Sederberg [40] has presented a very efficient method for doing so. As a first step, he transforms the Bézier basis into the power basis to be able to apply the Horner-like scheme which also appears in [17]:

$$\frac{\mathbf{p}(s, t)}{(1-s)^m (1-t)^n} = \sum_{i=0}^m \sum_{j=0}^n \hat{\mathbf{c}}_{ij} u^i v^j \tag{5.63}$$

with $u = s/(1-s)$ and $v = t/(1-t)$. For stability reasons, one should change the parameter transformation to $u = (1-s)/s$ and $v = (1-t)/t$ when either s or t are close to 1. The new control points $\hat{\mathbf{c}}_{ij}$ are computed as

$$\hat{\mathbf{c}}_{ij} = \binom{m}{i} \binom{n}{j} \mathbf{c}_{ij} \tag{5.64}$$

Incidentally, the formulae (5.63) and (5.64) can be used to evaluate a rational Bézier patch, if we perform the Horner scheme in projective space and perform a subsequent projection. More interesting is the potential for the two tangents:

Let us assume for now that the \mathbf{c}_{ij} are 4D points (homogeneous coordinates). Let us define subpatches:

$$\frac{\mathbf{p}^{\alpha,\beta}(s,t)}{(1-s)^{m-1}(1-t)^{n-1}} = \sum_{i=\alpha}^{m+\alpha-1} \sum_{j=\beta}^{n+\beta-1} \hat{\mathbf{c}}_{ij}^{\alpha\beta} u^i v^j \quad (5.65)$$

with

$$\hat{\mathbf{c}}_{ij}^{\alpha\beta} = \binom{m-1}{i-\alpha} \binom{n-1}{j-\beta} \mathbf{c}_{ij}. \quad (5.66)$$

Here α and β vary from 0 to 1. Sederberg then shows that these 4 subpatches combined yield the tangents:

$$\mathbf{p}_s(s,t) = K(\pi((1-t)\mathbf{p}^{00}(s,t) + t\mathbf{p}^{01}(s,t)) - \pi((1-t)\mathbf{p}^{10}(s,t) + t\mathbf{p}^{11}(s,t))) \quad (5.67)$$

$$\mathbf{p}_t(s,t) = K(\pi((1-s)\mathbf{p}^{00}(s,t) + s\mathbf{p}^{10}(s,t)) - \pi((1-s)\mathbf{p}^{01}(s,t) + s\mathbf{p}^{11}(s,t))) \quad (5.68)$$

It is further shown that for a surface of degree (n,n) the evaluation algorithm is of order $O(n^2)$. This is a substantial improvement over the straightforward evaluation.

5.6.3. Algorithms

In this section some of the most common algorithms will be presented. Let us first present the de Casteljau algorithm for evaluating rational Bézier patches. I will assume that we are performing the computations in projective space, hence we have control points $\hat{\mathbf{c}}_{ij} = [w_{ij}\mathbf{c}_{ij} \ w_{ij}]$. If we write the patch with the parenthesis as in (5.52) we can readily see the evaluation strategy:

$$\mathbf{p}(s,t) = \sum_{j=0}^n \left(\sum_{i=0}^m \hat{\mathbf{c}}_{ij} B_i^m(s) \right) B_j^n(t) \quad (5.69)$$

If we assume that we want to evaluate the patch at parameter (s_0, t_0) , then the algorithm in pseudo code is as follows:

for $j=0, j \leq n; j++$

Compute point $C_j(s_0)$ by performing the de Casteljau algorithm for curve defined by j -th row of control points

Define new curve $C(t)$ of degree n with control points $C_j(s_0)$

Perform de Casteljau algorithm once to evaluate $C(t)$ at parameter t_0 .

It should be immediately obvious that we could have reversed the roles of i and j and obtain the same result. However, from a practical point of view there is a difference: If the degree in s direction differs from the degree in t direction, a quick operations count shows that it is cheaper to perform more de Casteljau algorithms for a lower degree. So it is advantageous to perform the final de Casteljau algorithm in the direction corresponding to the higher degree. We did not present the direct de Casteljau algorithm for patches. Its non-rational version can be found in [17]. Since it is more complex and also requires case distinctions, it is rarely used in practice.

Reparameterizations

We have seen that rational Bézier curves can be reparameterized such that the end points have weights equal to 1. Obviously, we can perform this reparameterization for either the two $s = \text{const}$ boundary curves or the two $t = \text{const}$ boundary curves independently. The result will be a patch where the four corner points have weights equal to 1. We can consider such a patch to be in *standard form*. If we extract a isoparametric curve from a patch, say the curve corresponding to the i th row of the patch, then the resulting rational curve will not be in standard form. However, we can simply apply the curve reparameterization algorithm to convert this curve to standard form.

5.7. RATIONAL B-SPLINE SURFACES

This section introduces the most important entity in industrial applications, the rational B-spline surface or *NURBS* surface. This representation comprises all the surface representations encountered previously: rational and non-rational Bézier patches as well as non-rational B-spline patches. For surfaces the same is true as for curves: true rational patches are mostly used for representing entities such as cylinders, cones, spheres, tori and surfaces of revolution exactly. However, they are only used to a very limited extent for actual modeling purposes. An exception might be styling application where the weights are used as fairing or sculpting parameters.

5.7.1. Basic definitions

A rational B-spline surface of degree m in s and n in t is given by control points \mathbf{d}_{ij} , $i = 0, \dots, k$, $j = 0, \dots, l$, $\mathbf{d}_i \in \mathbb{R}^3$.

weights w_{ij} , $i = 0, \dots, k$, $j = 0, \dots, l$.

knot vectors $\zeta = \{s_0, \dots, s_{k+m+1}\}$, and $\tau = \{t_0, \dots, t_{l+n+1}\}$.

Again we assume that the two knots vectors have start and end knots of multiplicity $m + 1$ and $n + 1$ respectively. The NURBS patch $\mathbf{p}(s, t)$ is defined as

$$\mathbf{p}(s, t) = \frac{\sum_{i=0}^l (\sum_{j=0}^k w_{ij} \mathbf{d}_{ij} N_j^n(t)) N_i^m(s)}{\sum_{i=0}^l (\sum_{j=0}^k w_{ij} N_i^n(t)) N_i^m(s)} \quad (5.70)$$

In Figure 5.7 we show a rational B-spline patch. The normalized B-spline basis functions are the same as encountered before. They allows us to derive some of the properties of NURBS patches:

- Affine Invariance
- Convex hull property for non-negative weights
- Corner point interpolation

As can be expected, increasing the value of weight w_{ij} causes the surface to move towards control point \mathbf{d}_{ij} .

5.7.2. Derivatives

Computing derivatives follows the same recursive formula as for rational Bézier patches (see (5.62)). Using that formula, the computation of derivatives can be reduced to the

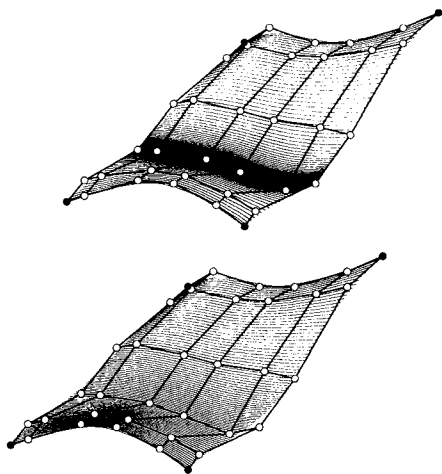


Figure 5.7. Example of a rational B-Spline patch: The top patch below has modified weights such that the vertical fold is more pronounced.

computation of derivatives of non-rational B-Spline surfaces. For a non-rational patch \mathbf{q} , the following formula holds:

$$\frac{\partial^{\alpha+\beta}}{\partial s^\alpha \partial t^\beta} \mathbf{q}(s, t) = \sum_{i=0}^{l-\alpha} \sum_{j=0}^{k-\beta} N_i^{m-\alpha} N_j^{n-\beta} \mathbf{d}_{ij}^{(\alpha,\beta)} \quad (5.71)$$

The intermediate B-Spline points $\mathbf{d}_{ij}^{(\alpha,\beta)}$ can be obtained as by-products of the de Boor algorithm; their recursive definition is

$$\mathbf{d}_{ij}^{(\alpha,\beta)} = \frac{(n-\beta+1)(m-\alpha+1)}{(s_{i+n+1}-s_{i+\alpha})(t_{j+n+1}-t_{\beta+1})} \mathbf{d}_{i+1,j+1}^{(\alpha-1,\beta-1)} - \mathbf{d}_{i+1,j}^{(\alpha-1,\beta-1)} - \mathbf{d}_{i,j+1}^{(\alpha-1,\beta-1)} + \mathbf{d}_{i,j}^{(\alpha-1,\beta-1)} \quad (5.72)$$

From a practical point of view, it might be advantageous to convert the rational B-Spline representation into a rational Bézier representation via knot insertion and then compute the derivatives of rational Bézier patches. This is true for applications that require the evaluation of a large number of points on the patch. The overhead of knot insertion would then be amortized by being able to use Sederberg's method that has been presented in Section 5.6.2. Let us take a special look at the partials at the start and end points of the curve. If we assume multiplicity equal to the order of the surface at the ends, the first partial derivatives can be computed as follows:

$$\frac{\partial \mathbf{p}}{\partial s}(s_0, t_0) = \frac{m}{(s_{m+1}-s_0)} \frac{w_{10}}{w_{00}} (\mathbf{d}_{10} - \mathbf{d}_{00}) \quad (5.73)$$

$$\frac{\partial \mathbf{p}}{\partial t}(s_0, t_0) = \frac{n}{(t_{n+1}-t_0)} \frac{w_{01}}{w_{00}} (\mathbf{d}_{01} - \mathbf{d}_{00}) \quad (5.74)$$

5.7.3. Algorithms

The most fundamental task is the evaluation of rational B-Spline patches. There are several options:

- Convert the patch to a rational Bézier patch via knot insertion and apply 'Bézier' algorithms
- Use the recurrence relation of the B-Spline basis functions to evaluate the patch in 4D making use of the tensor-product structure
- Perform the de Boor algorithm in 4D

For completeness, we briefly sketch the evaluation procedure using the de Boor algorithm. Assume that we are given a NURBS patch in homogeneous coordinates $\hat{\mathbf{p}}(s, t)$ with a $(k + 1) \times (l + 1)$ array of control points $\hat{\mathbf{d}}$. Then we can repeatedly apply the de Boor algorithm in the following fashion:

```

for j=0, j <= l; j++
  Compute point  $C_j(s_0)$  by performing the de Boor algorithm for curve
  defined by j-th row of control points
Define new curve  $C(t)$  of degree n with control points  $C_j(s_0)$ 
Perform de Casteljaou algorithm once to evaluate  $C(t)$  at parameter  $t_0$ .

```

We can apply the Moebius transform to each knot vector ζ and τ independently to change the knot spacings. However, it is in general not possible to use the Moebius transformation for manipulating weights. For example, we can not reparameterize a patch in such a way that all the four corner points have a weight equal to 1 since this would require to apply two transformation in say s and hence we would produce two different knot vectors. Hence, this situation differs from the rational Bézier setting.

5.8. GEOMETRIC CONTINUITY FOR RATIONAL PATCHES

As I have mentioned before, when dealing with geometric continuity, it is important to distinguish between continuity conditions in homogeneous setting and affine setting. Affine C^1 continuity for example only means that $\mathbf{q}' - w'\mathbf{p}$ (with $\mathbf{q} = w\mathbf{p}$) is continuous; it does not imply continuity of \mathbf{q}' and w' . The effect is that both quantities independently may have cusps. This can have ramifications for surface construction algorithms. For example isoparametric lines can exhibit cusps that may cause geometry processing algorithms applied to this patch to fail. An example of this can be found in [20]. The remedy for this situation is to construct HG^1 continuous patch complexes.

Necessary and sufficient conditions for tangent plane continuity of rational Bézier patches have been developed by Liu [29] and deRose [9]. These conditions treat the rational Bézier patches as polynomial patches in 4-space: Given two patches $\hat{\mathbf{p}}_1(u, u)$ and $\hat{\mathbf{p}}_2(u, v)$ with a common boundary along $v = 0$, then the patches are G^1 continuous if and only if for all points along the common boundary

$$\det[\hat{\mathbf{p}}_1, D_u\hat{\mathbf{p}}_1, D_v\hat{\mathbf{p}}_1, D_v\hat{\mathbf{p}}_2] = 0 \quad (5.75)$$

The situation is illustrated in Figure 5.8.

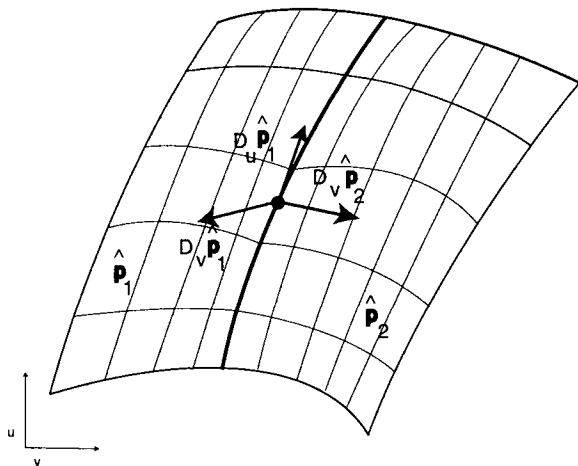


Figure 5.8. Two patches meeting at a common boundary. The depicted derivatives need to be linearly dependent.

Note that this determinant is considered a 4×4 determinant in 4 space. In [9] some construction algorithms using this condition are given. This condition is often generalized to the following equation:

$$D_v \hat{\mathbf{p}}_2 = \alpha_1 \hat{\mathbf{p}}_1 + \beta_1 D_v \hat{\mathbf{p}}_1 + \gamma_1 D_u \hat{\mathbf{p}}_1 \tag{5.76}$$

where $\alpha_1(u), \beta_1(u)$ and γ_1 are rational scalar-valued polynomials in u . Analogously, one can derive the conditions for curvature continuity: Two patches \mathbf{p}_1 and \mathbf{p}_2 are curvature continuous along a common boundary $v = 0$ if and only if Equation (5.76) holds and in addition

$$D_v^2 v \hat{\mathbf{p}}_2 = \alpha_2 \hat{\mathbf{p}}_1 + \beta_2 D_v \hat{\mathbf{p}}_1 + \gamma_2 D_u \hat{\mathbf{p}}_1 + \beta_1^2 D_v^2 v \hat{\mathbf{p}}_1 + 2\beta_1 \gamma_1 D_v^2 u \hat{\mathbf{p}}_1 + \gamma_1^2 D_u^2 u \hat{\mathbf{p}}_1 \tag{5.77}$$

Again the coefficients $\alpha_{1,2}, \beta_{1,2}$ and $\gamma_{1,2}$ are rational polynomials in u . They are often called the connection functions. In [47] these conditions are refined and rational expressions for these connection functions are computed. The tangent plane and curvature continuity conditions are the most important conditions in practice. In some cases, it might be necessary to require higher order continuity, for example continuity of the rate of change of curvature (G^3). Zheng et al. [48] present some general G^n conditions building upon the framework presented above. The interested reader is referred to this paper. A more in depth treatment of geometric continuity can be found in Chapter 8 on *Geometric Continuity*.

5.9. INTERPOLATION AND APPROXIMATION ALGORITHMS

The interpolation problem for rational patches is often posed as the task of finding a rational patch that interpolates data points \mathbf{p}_i given in homogeneous coordinates $\mathbf{p}_i =$

$[w\mathbf{x}\ w\mathbf{y}\ w\mathbf{z}\ w]_1^T$. As pointed out before, there is no good method to determine the weights a priori. An alternative has been proposed by Ma and Kruth [33]. They assume that the parameters for the data points as well as the knot values are given. Then they use the interpolation conditions to set up a system of equations containing the weights and the control points as unknowns. This system is usually overdetermined. The system is transformed so that a two step method can be applied. First one solves for the weights using a singular value decomposition, then one solves for the control points. As an alternative, the weights can be computed by using a constrained minimization scheme in order to keep the weights from becoming negative or otherwise ill-behaved. A very similar method results when applying the method by Schneider [37] presented in Section 5.5.2 to surfaces. One just needs to order the array of data points and enumerate them as a list of points. If we have m data points we again need to make sure to have n control points with $n = 3m/4$.

Even though these methods are feasible in practice, it is more common to look at the problem of approximating a discrete set of points or of approximating a surface given by a different representation. The second problem arises very frequently since NURBS are not closed under operations such as offsetting or surface-surface intersections. Schneider and Juettler [38] presented an approximation technique for curves that can be readily generalized to surfaces: We consider an $(k+1) \times (l+1)$ array of data points $\mathbf{q}_{ij}, i = 0, \dots, k, j = 0, \dots, l$. Assume that we have a rational patch $\mathbf{p}(s, t)$ with knot vectors ζ and τ and control points \mathbf{d}_j and weights w_{ij} ($i = 0 \dots, m, j = 0, \dots, n$). We want to compute the rational NURBS surface that minimizes

$$\mathcal{F}(\mathbf{W}, \mathbf{D}, \mathbf{S}, \mathbf{T}) = \sum_{i=0}^k \sum_{j=0}^l \|\mathbf{q}_{ij} - \mathbf{p}(s_i, t_i)\| \tag{5.78}$$

Here \mathbf{W} denotes the collection of weights, \mathbf{D} denotes the set of control points, and \mathbf{S} and \mathbf{T} denotes the set of parameters. Note that we leave the parameters free and iteratively adjust these to find a better solution. The adjustment can be performed by using Hoschek's parameter correction scheme [25]. We can now collect and enumerate all data points and control points in the obvious way: $I = i_s + i_t * k, i_s = 0, \dots, k, i_t = 0, \dots, l$ and $J = j_s + j_t * n, j_s = 0 \dots, m, j_t = 0, \dots, n$. With $M = k * l - 1$ and $N = m * n - 1$ we can transform the minimization problem into:

$$\mathcal{F}(\mathbf{W}, \mathbf{D}, \mathbf{S}, \mathbf{T}) = \sum_{l=0}^3 \|\mathbf{P} - \Phi(\mathbf{W}, \mathbf{S}, \mathbf{T})\mathbf{D}^l\| \longrightarrow \min \tag{5.79}$$

The matrix Φ is crucial. It has entries

$$\Phi_{IJ} = \frac{w_J N_{j_s}^{k_s}(s_{i_s}) N_{j_t}^{k_t}(t_{i_t})}{\sum_{H=0}^N w_H N_{h_s}^{k_s}(s_{i_s}) N_{h_t}^{k_t}(t_{i_t})} \tag{5.80}$$

Note the interplay of two-dimensional indices i_s, i_t with I, j_s, j_t with J and h_s and h_t with H . k_s and k_t is the degree of the patch in s and t . If we fix an initial parameterization \mathbf{S}_0 and \mathbf{T}_0 as well as an initial set of weights \mathbf{W}_0 , we can now compute the control points by applying the pseudo inverse Φ^+ of Φ :

$$\mathbf{D} = \Phi^+(\mathbf{W}_0, \mathbf{S}_0, \mathbf{T}_0)\mathbf{P} \tag{5.81}$$

To compute or update a set of weights, we need to solve another nonlinear minimization problem:

$$\mathcal{G}(\mathbf{W}) = \sum_{l=1}^3 \|\mathbf{P} - \Phi^+ \Phi \mathbf{P}\| \longrightarrow \min \quad (5.82)$$

In order to have a sensible range of the weights, the authors apply a transformation to the w_k : $\hat{w}_k = \epsilon + \pi/2 + \arctan(w_k)$ where $\epsilon > 0$ is chosen arbitrarily. The entire method is an iterative process; a) start with an initial parameterization, b) compute a set of weights by performing a non-linear minimization, c) solve for the de Boor points using the pseudo-inverse, and d) perform parameter correction to update the set of parameters. This loop is performed until a desired error tolerance is reached. The reader should note that no proof for the convergence of this method has been given here.

I presented this method here, since this constitutes one of the very few if not the only practical algorithm for using the weights as unknowns to solve the surface approximation problem. Another approach is the method presented in [26]. Here the parameters, the weights as well as the control points are all treated as free variables for the solution of a global minimization problem. Solving this system without getting stuck in the first local minimum requires some preconditioning. Timings show that this method is very inefficient and no geometric insight is applied.

5.10. RATIONAL SURFACE CONSTRUCTIONS

In this section I will briefly cover some of the more important surface constructions involving rational patches.

5.10.1. Surfaces of revolution

A surface of revolution is typically given as

$$\mathbf{s}(u, v) = (r(v)\cos(u), r(v)\sin(u), z(v))^T \quad (5.83)$$

Closer inspection reveals that each isoparametric line $v = \text{const}$ traces out a circle with radius $r(v)$. Under the assumption that the generatrix $\mathbf{g}(v) = [r(v), 0, z(v)]^T$ can be represented in rational form, we can construct the surface using rational Bézier patches. For each vertex of the generatrix, we generate four circular segments in Bézier form (see Figure 5.9).

This yields all the control points. For all midpoints of the circular segments the weights of the generatrix are copied. The other points are assigned these weights multiplied by $\sqrt{2}/2$. Example of surfaces generated this way are cylinders, tori and spheres.

5.10.2. Canal and pipe surfaces

In practice it is often necessary to construct surfaces $\mathbf{s}(u, v)$ from a given curve $\mathbf{c}(v)$ whose isoparametric curve $v = \text{const}$ is a circle with center $C(v_0)$ and radius r lying in the normal plane to $\mathbf{c}(v_0)$. Such surfaces are called pipe surfaces and they can be viewed as generalizations to offset curves in the plane. They can be expressed as

$$\mathbf{s}(u, v) = \mathbf{c}(v) + r\mathbf{n}(u, v) \quad (5.84)$$

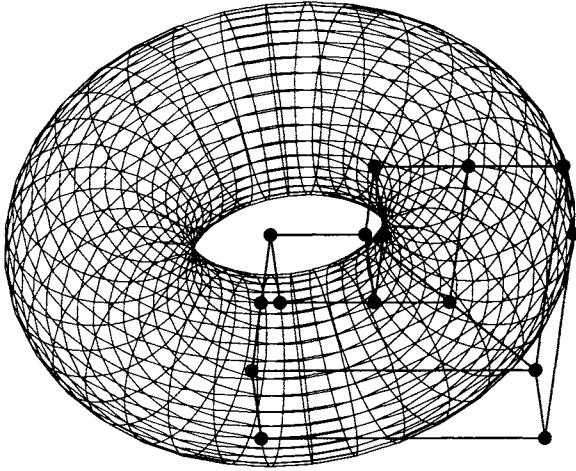


Figure 5.9. Construction of a surface of revolution using 4 patches per meridian

Here $\mathbf{n}(u, v)$ is the unit circle with center $\mathbf{c}(t)$ lying in the normal plane.

If we assume that the spine curve is rational, then it is clear that we need to find a function $\mathbf{n}(u, v)$ with norm equal to 1 and additional constraint that $\langle \mathbf{n}(u, v), \mathbf{c}'(v) \rangle = 0$. In [32] the authors prove the following: if $\mathbf{c}(v) = [x(v), y(v), z(v)]^T$ is a rational curve with $z'(v) \neq 0$, then the pipe surface with spine $\mathbf{c}(v)$ can be rationally parameterized if and only if we can find two rational functions $f(v)$ $g(v)$ such that

$$x'^2(v) + z'^2(v) - f^2(v)(x^2(v) + y^2(v) + z^2(v)) = g^2(v). \quad (5.85)$$

The authors go on to prove that any pipe surface with rational spine curve is rational and give a construction. Furthermore, it turns out that the result is valid for the more general case where the radius r varies with parameter v . Such surfaces are often called canal surfaces. Figure 5.10 depicts a canal surface that has been built as a collection of piecewise quadratic rational patches.

5.11. CONCLUDING REMARKS

This chapter gave an overview of rational techniques with particular emphasis on practical applications. Due to space limitations some topics had to be omitted entirely, I also had to make choices on how to present the material. A very important class of surfaces are *quadric* surfaces. These surfaces can be represented precisely by triangular rational Bézier patches. The reader should consult Chapter 31 on *Quadratics*. In that chapter, the author covers in depth the parametric representation of quadric surfaces using rational Bézier patches. Note that I did not discuss triangular rational Bézier patches in this chapter. Even though they are interesting in their own right, they are not frequently used in practice except for representing quadrics. Unfortunately, the vast majority of



Figure 5.10. A canal surface composed of piecewise quadratic patches. (Figure courtesy of D. Hansford).

commercial CAD packages do not support triangular patches. The interested reader can consult [16,4].

I chose not to follow the blossoming notation - the reader can get a first glimpse in [17] if so inclined. The theory of blossoms is developed completely in [35]. Furthermore, I did not cover any of the data representation standards in depth. In [16], the reader can find the basic IGES format for NURBS curves and surfaces. NURBS are also included in the STEP standard. The National Institute of Standards is involved in defining STEP; more information can be found in [41].

REFERENCES

1. G.A. Baker. *Essentials of Padé approximants*. Addison-Wesley, Reading, 1975.
2. C. Baumgarten and G. Farin. Approximation of logarithmic spirals. *Computer Aided Geometric Design*, 14(6), 1996.
3. C. Bajaj and G. Xu. NURBS approximation of surface surface intersection curves. *Advances in Computational Mathematics*, 2(1):1-21, 1994.
4. W. Boehm and D. Hansford. Bézier patches on quadrics. In G. Farin, editor, *NURBS for Curve and Surface Design*, SIAM, 1991.
5. W. Boehm. Rational geometric splines. *Computer Aided Geometric Design*, 4(1-2), 1987.
6. C. Brezinski. An introduction to Padé approximations. In Laurent, Le Méhauté, Schumaker, editors, *Curves and Surfaces for Geometric Design*. A.K. Peters, Boston, 1994.
7. W. Degen. Some remarks on Bézier curves. *Computer Aided Geometric Design*, 5(3),

- 1988.
8. W. Degen. High accuracy rational approximation of parametric curves. *Computer Aided Geometric Design*, 10(3-4), 1993.
 9. A. de Rose. Necessary and sufficient conditions for tangent plane continuity of Bézier surfaces. *Computer Aided Geometric Design*, 7(1-4), 1990.
 10. A. de Rose. Rational Bézier curves and surfaces on projective domains. In G. Farin, editor, *NURBS for Curve and Surface Design*, SIAM, 1991.
 11. M. Eck. Degree reduction of Bézier curves. *Computer Aided Geometric Design*, 10(3-4), 1993.
 12. M. Eck. Least squares degree reduction of Bézier curves. *Computer-Aided Design*, 27(11), 1995.
 13. G. Farin. Algorithms for rational Bézier curves. *Computer-Aided Design*, 15(2), Feb 1983.
 14. G. Farin and A. Worsley. Reparameterization and degree elevation for rational Bézier curves. In G. Farin, editor, *NURBS for Curve and Surface Design*, SIAM, 1991.
 15. G. Farin. Tighter convex hulls for rational Bézier curves. *Computer Aided Geometric Design*, 10(2), 1993.
 16. G. Farin. *NURB Curves and Surfaces*, 1st Edition, AK Peters, 1994.
 17. G. Farin. *Curves and Surfaces for CAGD*, 4th Edition, Academic Press, San Diego, 1997.
 18. M.S. Floater. Derivatives of rational Bézier curves. *Computer Aided Geometric Design*, 9(3), 1992.
 19. M.S. Floater. Evaluation and properties of the derivative of a NURBS curve. In T. Lyche and L.L. Schumaker, editors, *Mathematical Methods in CAGD*, pages 261–274. Academic Press, Boston, 1992.
 20. R. Fuhr, L. Hsieh, and M. Kallay. Object-oriented paradigm for NURBS curve and surface design. *Computer-Aided Design*, 27(2), 1995.
 21. M. Gutknecht. Block structure and recursiveness in rational interpolation. In Cheney, Chui, and Schumaker, editors, *Approximation Theory VII*, Academic Press, 1993.
 22. T.N.T. Goodman, B.H. Ong, and K. Unsworth. Constrained interpolation using rational cubic splines. In G. Farin, editor, *NURBS for Curve and Surface Design*, SIAM, 1991.
 23. K. Hoellig. Algorithms for rational spline curves. ARO-Report 88-1, pages 287–300, 1988.
 24. W. Hohenberger and T. Reuding. Smoothing rational B-spline curves using the weights in an optimization procedure. *Computer Aided Geometric Design*, 12(8), 1995.
 25. J. Hoschek. Intrinsic parameterization for approximation. *Computer Aided Geometric Design*, 5(1), 1988.
 26. P. Laurent-Gengoux and M. Mekhilef. Optimization of a NURBS representation. *Computer-Aided Design*, 25(11), 1993.
 27. R. Liming. *Mathematics for Computer Graphics*. Aero publishers, 1979.
 28. R. Liming. *Practical Analytical Geometry with Applications to Aircraft*. Macmillan, 1944.
 29. D. Liu. GC^1 continuity conditions between two adjacent rational Bézier surface

- patches. *Computer Aided Geometric Design*, 7(1-4), 1990.
30. E. Lee. The rational Bézier representation for conics. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, SIAM, Philadelphia, 1987.
 31. E. Lee and M. Lucian. Moebius reparameterizations of rational B-Splines. *Computer Aided Geometric Design*, 8(3), August 1991.
 32. W. Lu and H. Pottmann. Pipe surfaces with rational spine curve are rational. *Computer Aided Geometric Design*, 13(7), 1996.
 33. W. Ma and J.P. Kruth. NURBS curve and surface fitting and interpolation. In M. Daehlen, T. Lyche, and L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, Vanderbilt University Press, 1994.
 34. L. Piegl. Modifying the shape of rational B-splines, Part I: Curves. *Computer-Aided Design*, 21(8), 1989.
 35. L. Ramshaw. Blossoming: A connect-the-dots approach to splines. Research Report 19, Digital Systems Research Center, Palo Alto, 1987.
 36. R. Schaback. Planar curve interpolation by piecewise conics of arbitrary type. *Constructive Approximation*, 9:373-389, 1993.
 37. F.J. Schneider. *Interpolation, Approximation und Konvertierung mit rationalen B-Splines*. PhD thesis, TH Darmstadt, 1993.
 38. F.J. Schneider and B. Juettler. Interpolation and approximation with rational B-spline curves and surfaces. Preprint, 1994.
 39. I.J. Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. *Quart. Appl. Mathematics*, 4, 1946.
 40. T. Sederberg. Point and tangent computation of tensor product rational Bézier surfaces. *Computer Aided Geometric Design*, 12(1), 1995.
 41. The STEP Project at NIST. Home page <http://www.nist.gov/sc4/www/stepdocs.htm>
 42. H. Theisel. Using Farin points for rational Bézier surfaces. *Computer Aided Geometric Design*, 16(8), 1999.
 43. K. Versprille. *Computer Aided Design Applications of the Rational B-Spline Approximation Form*. PhD thesis, Syracuse University, 1975.
 44. W. Wang and B. Joe. Interpolation on quadric surfaces with rational quadratic spline curves. *Computer Aided Geometric Design*, 14(3), 1997.
 45. H.J. Wolters. *Rational Geometric Curve Approximation*. PhD thesis, Arizona State University, 1994.
 46. H.J. Wolters and G. Farin. Geometric curve approximation. *Computer Aided Geometric Design*, 14(7), 1997.
 47. J. Zheng, G. Wang, and Y. Liang. Curvature continuity between adjacent rational Bézier patches. *Computer Aided Geometric Design*, 9(5), 1992.
 48. J. Zheng, G. Wang, and Y. Liang. GC^n continuity conditions for adjacent rational parametric surfaces. *Computer Aided Geometric Design*, 12(2), 1995.

Chapter 6

Spline Basics

Carl de Boor

This chapter promotes, details and exploits the fact that (univariate) splines, i.e., smooth piecewise polynomial functions, are weighted sums of B-splines.

6.1. PIECEWISE POLYNOMIALS

A **piecewise polynomial** of **order** k with **break sequence** ξ (necessarily strictly increasing) is, by definition, any function f that, on each of the half-open intervals $[\xi_j \dots \xi_{j+1})$, agrees with some polynomial of degree $< k$. The term ‘order’ used here is not standard but handy.

Note that this definition makes a piecewise polynomial function **right-continuous**, meaning that, for any x , $f(x) = f(x+) := \lim_{h \downarrow 0} f(x+h)$. This choice is arbitrary, but has become standard. Keep in mind that, at its break ξ_j , the piecewise polynomial function f has, in effect, two values, namely its limit from the left, $f(\xi_j-)$, and its limit from the right, $f(\xi_j+) = f(\xi_j)$.

The set of all piecewise polynomial functions of order k with break sequence ξ is denoted here

$$\Pi_{<k,\xi}.$$

6.2. B-SPLINES DEFINED

B-splines are defined in terms of a **knot sequence** $\mathbf{t} := (t_j)$, meaning that

$$\dots \leq t_j \leq t_{j+1} \leq \dots.$$

The j th **B-spline of order 1 for the knot sequence** \mathbf{t} is the characteristic function of the half-open interval $[t_j \dots t_{j+1})$, i.e., the function given by the rule

$$B_{j1}(x) := B_{j,1,\mathbf{t}}(x) := \begin{cases} 1, & \text{if } t_j \leq x < t_{j+1}; \\ 0, & \text{otherwise.} \end{cases}$$

Note that each of these functions is piecewise constant, and that the resulting sequence (B_{j1}) is a **partition of unity**, i.e.,

$$\sum_j B_j(x) = 1, \quad \inf_j t_j < x < \sup_j t_j.$$

In particular,

$$t_j = t_{j+1} \text{ implies } B_{j1} = 0.$$

From these first-order B-splines, B-splines of higher order can be derived inductively by the following **B-spline recurrence**.

(6.2.1) Property (i): Recurrence relation. *The j th B-spline of order $k > 1$ for the knot sequence \mathbf{t} is*

$$(6.2.2) \quad B_{jk} := B_{j,k,\mathbf{t}} := \omega_{jk} B_{j,k-1} + (1 - \omega_{j+1,k}) B_{j+1,k-1},$$

with

$$(6.2.3) \quad \omega_{jk}(x) := \omega_{j,k,\mathbf{t}}(x) := \frac{x - t_j}{t_{j+k-1} - t_j}.$$

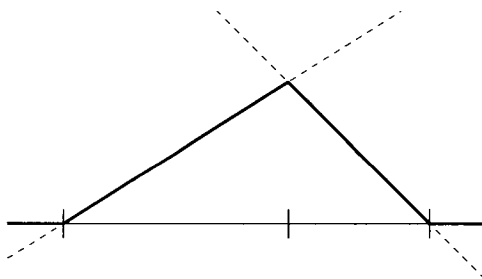


Figure 6.2.4 The functions ω_{j2} and $1 - \omega_{j+1,2}$ (dashed), and the linear B-spline B_{j2} (solid) formed from them.

For example, the j th **second-order** or **linear** B-spline is given by

$$B_{j2} = \omega_{j2} B_{j1} + (1 - \omega_{j+1,2}) B_{j+1,1},$$

and so consists of two nontrivial linear pieces and is continuous, unless there is some equality in the inequalities $t_j \leq t_{j+1} \leq t_{j+2}$.

In order to appreciate just how remarkable the recurrence relation is, consider the 3rd-order B-spline

$$B_{j3} = \omega_{j3} B_{j2} + (1 - \omega_{j+1,3}) B_{j+1,2}$$

in the generic case, i.e., when $t_j < t_{j+1} < t_{j+2} < t_{j+3}$. As is illustrated in Figure (6.2.5), both summands have corners (i.e., jumps in their first derivative), but these corners appear to be perfectly matched so that their sum is smooth.

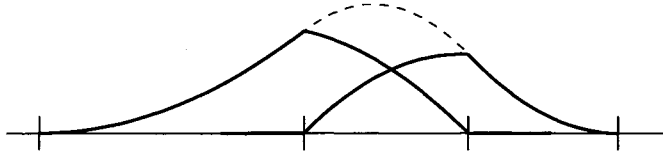


Figure 6.2.5 The two functions $\omega_{j3}B_{j2}$ and $(1 - \omega_{j+1,3})B_{j+1,2}$ have corners, but their sum, B_{j3} , does not.

6.3. SUPPORT AND POSITIVITY

Directly from (6.2.2) by induction on k ,

$$B_{jk} = b_j B_{j1} + \dots + b_{j+k-1} B_{j+k-1,1},$$

with each b_r a product of $k - 1$ polynomials of (exact) degree 1, hence a polynomial of (exact) degree $k - 1$. This shows B_{jk} to be a piecewise polynomial of order k , with breaks at t_j, \dots, t_{j+k} .

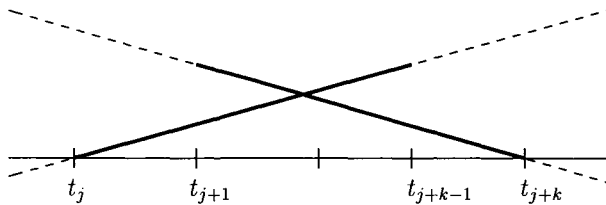


Figure 6.3.1 The two weight functions, ω_{jk} and $1 - \omega_{j+1,k}$, in (6.2.2) are positive on $\text{supp}(B_{jk}) = (t_j \dots t_{j+k})$.

Further, B_{jk} is zero off the interval $[t_j \dots t_{j+k}]$. Hence, since both ω_{jk} and $1 - \omega_{j+1,k}$ are positive on the interval $(t_j \dots t_{j+k})$, it follows, by induction on k , that B_{jk} is positive there.

(6.3.2)Property (ii): Support and positivity. The B-spline $B_{j,k} = B_{j,k,t}$ is piecewise polynomial of order k with at most k nontrivial polynomial pieces, and breaks only at t_j, \dots, t_{j+k} , vanishes outside the interval $[t_j \dots t_{j+k})$, and is positive on the interior of that interval, that is,

$$(6.3.3) \quad B_{j,k}(x) > 0, \quad t_j < x < t_{j+k},$$

while

$$(6.3.4) \quad t_j = t_{j+k} \implies B_{j,k} = 0.$$

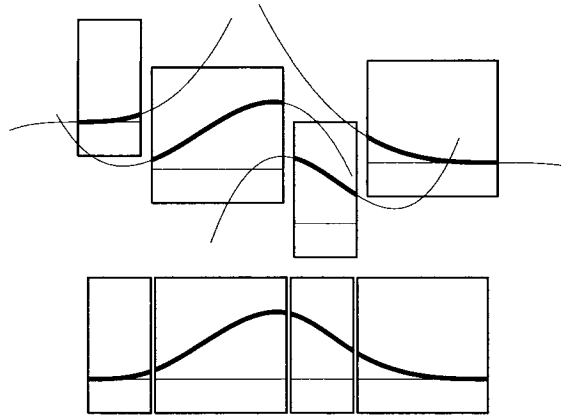


Figure 6.3.5 The four cubic polynomials whose pieces join to form a certain cubic B-spline.

Notice that B_{ik} is completely determined by the $k + 1$ knots t_i, \dots, t_{i+k} . For this reason, the notation

$$B(\cdot | t_i, \dots, t_{i+k}) := B_{i,k,t} = B_{ik}$$

is sometimes used. Other notations in use include

$$N_{ik} := B_{ik} \text{ and } M_{ik} := (k/(t_{i+k} - t_i))B_{ik}.$$

The latter is special in that

$$\int_{\mathbf{R}} M_{ik} = 1,$$

as follows from (6.11.4).

The many other properties of B-splines are derived most easily by considering not just one B-spline but the linear span of *all* B-splines of a given order k for a given knot sequence \mathbf{t} . This brings us to splines.

6.4. SPLINE SPACES DEFINED

A **spline of order k with knot sequence \mathbf{t}** is, by definition, a linear combination of the B-splines B_{ik} associated with that knot sequence. We denote by

$$(6.4.1) \quad S_{k,\mathbf{t}} := \left\{ \sum_i a_i B_{ik} : a_i \in \mathbf{R} \right\}$$

the collection of all such splines. .

It has become customary in CAGD to use the term ‘B-spline’ for what has just been defined to be a spline. This unfortunate mistake will not be made in this chapter, particularly since, once made, one has to make up another term (such as ‘B-spline basis function’ and the like) for what is called here by its original name, namely a ‘B-spline’.

So far, the knot sequence \mathbf{t} has been left unspecified except for the requirement that it be nondecreasing. In any practical situation, \mathbf{t} is necessarily a *finite* sequence. But, since on any nontrivial interval $[t_j \dots t_{j+1})$ at most k of the B_{ik} are nonzero, namely $B_{j-k+1,k}, \dots, B_{jk}$ (see Figure (6.4.2)), it does not really matter whether \mathbf{t} is finite, infinite, or even bi-infinite; the sum in (6.4.1) always makes pointwise sense, meaning that $\sum_i a_i B_{ik}(x)$ is well-defined for any x , since at most k of its summands are not zero.

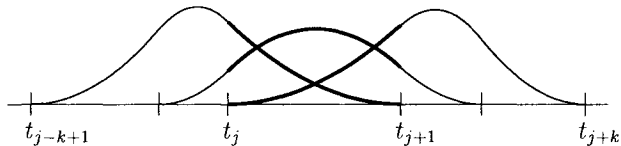


Figure 6.4.2 The k B-splines whose support contains $[t_j \dots t_{j+1})$; here $k = 3$.

However, while each B_{jk} is defined on the entire real line, \mathbb{R} , it is convenient to restrict all claims concerning the spline space $S_{k,\mathbf{t}}$ to its **basic interval**

$$I_{k,\mathbf{t}}$$

which, by definition, is the union of all knot intervals $[t_j \dots t_{j+1}]$ on which the full complement of k different B-splines from (B_{ik}) have some support. Correspondingly, if $I_{k,\mathbf{t}}$ has a finite right endpoint, it is very convenient to modify the earlier definition of B-splines to make them *left-continuous at that right endpoint*.

At times, it will be convenient to assume that

$$t_i < t_{i+k}, \text{ all } i,$$

which can always be achieved by removing from \mathbf{t} its i th entry as long as $t_i = t_{i+k}$. This does not change the space $S_{k,\mathbf{t}}$ since the only k th order B-splines removed thereby are zero anyway. In fact, another way to state this condition is:

$$B_{ik} \neq 0, \text{ all } i.$$

6.5. SPECIFIC KNOT SEQUENCES

The following two ‘extreme’ knot sequences have received special attention:

$$\mathbb{Z} := (\dots, -2, -1, 0, 1, 2, \dots), \quad \mathbb{B} := (\dots, 0, 0, 0, 1, 1, 1, \dots).$$

A spline associated with the knot sequence \mathbb{Z} is called a **cardinal** spline. This term was chosen by Schoenberg [10] because of a connection to Whittaker’s Cardinal Series. This is not to be confused with its use in earlier spline literature where it refers to a spline that vanishes at all points in a given sequence except for one at which it takes the value 1. The latter splines, though of great interest in spline interpolation, do not interest us here.

Because of the uniformity of the knot sequence $\mathbf{t} = \mathbb{Z}$, formulæ involving cardinal B-splines are often much simpler than corresponding formulæ for general B-splines. To begin with, *all cardinal B-splines (of a given order) are translates of one another*. With the natural indexing $t_i := i$, all i , for the entries of the uniform knot sequence $\mathbf{t} = \mathbb{Z}$, we have

$$B_{ik} = N_k(\cdot - i),$$

with

$$(6.5.1) \quad N_k := B_{0k} = B(\cdot | 0, \dots, k).$$

The recurrence relation (6.2.2) simplifies as follows:

$$(6.5.2) \quad (k-1)N_k(t) = tN_{k-1}(t) + (k-t)N_{k-1}(t-1).$$

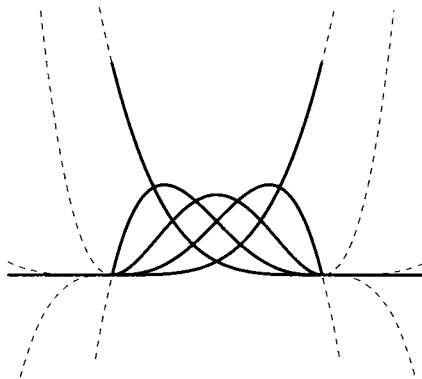


Figure 6.5.3 Bernstein basis of degree 4 or order 5

The knot sequence $\mathbf{t} = \mathbb{B}$ contains just two points, namely the points 0 and 1, but each with infinite multiplicity. The only nontrivial B-splines for this sequence are those that have both 0 and 1 as knots, i.e., those B_{ik} for which $t_i = 0$ and $t_{i+k} = 1$; see Figure (6.5.3). There seems to be no natural way to index the entries in the sequence \mathbb{B} . Instead, it is customary to index the corresponding B-splines by the multiplicities of their two distinct knots. Precisely,

$$(6.5.4) \quad B_{(\mu,\nu)} := B(\cdot | \underbrace{0, \dots, 0}_{\mu+1 \text{ times}}, \underbrace{1, \dots, 1}_{\nu+1 \text{ times}}).$$

With this, the recurrence relations (6.2.2) simplify as follows:

$$(6.5.5) \quad B_{(\mu,\nu)}(x) = xB_{(\mu,\nu-1)}(x) + (1-x)B_{(\mu-1,\nu)}(x).$$

This gives the formula

$$(6.5.6) \quad B_{(\mu,\nu)}(x) = b_{\nu}^{\mu+\nu}(x) := \binom{\mu+\nu}{\mu} (1-x)^{\mu} x^{\nu} \quad \text{for } 0 < x < 1$$

for the one nontrivial polynomial piece of $B_{(\mu,\nu)}$, as one verifies by induction. The formula enables us to determine the *smoothness* of the B-splines in this simple case: Since $B_{(\mu,\nu)}$ vanishes identically outside $[0..1]$, it has exactly $\nu - 1$ continuous derivatives at 0 and $\mu - 1$ continuous derivatives at 1. This amounts to ν **smoothness conditions** at 0 and μ smoothness conditions at 1. Since the order of $B_{(\mu,\nu)}$ is $\mu + \nu + 1$, this is a simple illustration of the generally valid formula

$$(6.5.7) \quad \#\text{smoothness conditions at knot} + \text{multiplicity of knot} = \text{order}.$$

For fixed $\mu + \nu$, the polynomials in (6.5.6) form the so-called **Bernstein** basis (for polynomials of degree $\leq \mu + \nu$) and, correspondingly, the representation

$$(6.5.8) \quad p = \sum_{\mu+\nu=h} a_{(\mu,\nu)} B_{(\mu,\nu)}$$

is the **Bernstein-Bézier** form for the polynomial $p \in \Pi_h$. It may be simpler to use the short term **BB-form** instead.

6.6. THE POLYNOMIALS IN THE SPLINE SPACE: MARS DEN'S IDENTITY

Directly from the recurrence relation,

$$(6.6.1) \quad \sum a_j B_{jk} = \sum ((1 - \omega_{jk})a_{j-1} + \omega_{jk}a_j) B_{j,k-1}.$$

On the other hand, for the special sequence

$$a_j := \psi_{jk}(\tau) := (t_{j+1} - \tau) \cdots (t_{j+k-1} - \tau),$$

one finds for $B_{j,k-1} \neq 0$, i.e., for $t_j < t_{j+k-1}$ that

$$(1 - \omega_{jk})a_{j-1} + \omega_{jk}a_j = (\cdot - \tau)\psi_{j,k-1}(\tau).$$

Hence, induction on k establishes the following.

(6.6.2) B-spline property (iii): Marsden's identity. For any $\tau \in \mathbb{R}$,

$$(6.6.3) \quad (\cdot - \tau)^{k-1} = \sum_j \psi_{jk}(\tau) B_{jk} \quad \text{on } I_{k,t},$$

with

$$(6.6.4) \quad \psi_{jk}(\tau) := (t_{j+1} - \tau) \cdots (t_{j+k-1} - \tau).$$

Since τ here is arbitrary, it follows that $S_{k,t}$ contains all polynomials of degree $< k$. More than that, differentiation of (6.6.3) with respect to τ leads to the following explicit B-spline expansion of an arbitrary $p \in \Pi_{<k}$:

$$(6.6.5) \quad p = \sum_i B_{ik} \lambda_{ik} p, \quad \text{on } I_{k,t},$$

with λ_{ik} given by the rule

$$(6.6.6) \quad \lambda_{ik} f := \sum_{\nu=1}^k \frac{(-D)^{\nu-1} \psi_{ik}(\tau)}{(k-1)!} D^{k-\nu} f(\tau).$$

For the particular choice $p = 1$, this gives

(6.6.7) B-spline property (iv): (Positive and local) partition of unity. The sequence (B_{jk}) provides a positive and local partition of unity, that is, each B_{jk} is positive on $(t_j \dots t_{j+k})$, is zero off $[t_j \dots t_{j+k}]$, and

$$(6.6.8) \quad \sum_j B_{jk} = 1 \quad \text{on } I_{k,t}.$$

Further, by considering $p = \ell \in \Pi_2$, one obtains

(6.6.9) B-spline property (v): Knot averages. For $k > 1$ and any $\ell \in \Pi_2$,

$$\ell = \sum_j \ell(t_{jk}^*) B_{jk} \quad \text{on } I_{k,t},$$

with t_{jk}^* the Greville sites:

$$(6.6.10) \quad t_{jk}^* := \frac{t_{j+1} + \dots + t_{j+k-1}}{k-1}, \quad \text{all } j.$$

6.7. THE PIECEWISE POLYNOMIALS IN THE SPLINE SPACE

Each $s \in S_{k,t}$ is piecewise polynomial of order k , with breaks only at its knots. If ξ is the strictly increasing sequence of *distinct* knots, then we can write this as

$$S_{k,t} \subseteq \Pi_{<k,\xi}.$$

But $S_{k,t}$ is usually a proper subset of $\Pi_{<k,\xi}$. Which subset exactly depends on the **knot multiplicities**

$$\#t_j := \#\{i : t_i = t_j\}$$

according to the rule (6.5.7). This is usually proved by showing that $S_{k,t}$ contains the **truncated power** function $(\cdot - t_i)_+^{k-r}$ if and only if $r \leq \#t_i$. Here

$$\alpha_+^j := \begin{cases} \alpha^j, & \alpha > 0; \\ 0, & \alpha < 0, \end{cases}$$

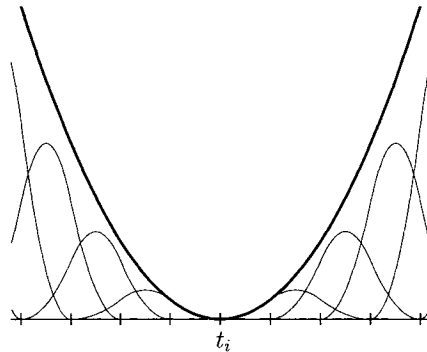


Figure 6.7.1 The terms $\psi_{jk}(t_i)B_{jk}$ and their sum, $(\cdot - t_i)^{k-1}$, for $k = 3$. Note that *all* these terms are zero at t_i . Hence, by summing only the terms that are nonzero somewhere to the right of t_i , one gets instead the *truncated* power, $(\cdot - t_i)_+^{k-1}$.

with its value at 0 determined by whatever convention is adopted with respect to right or left continuity at a break.

Figure (6.7.1) gives an illustration of how Marsden's Identity can be used to prove that the truncated power function $(\cdot - t_i)_+^{k-1}$ is in $S_{k,t}$. For $r > 1$, one may use the $(r - 1)$ st derivative with respect to τ of that identity in the same way, provided only that $B_{jk}(t_i) \neq 0$ implies that $D^{r-1}\psi_{jk}(t_i) = 0$, i.e., provided $\#t_i \geq r$.

Now, the truncated power function $f := (\cdot - t_i)_+^\nu$ satisfies exactly ν **smoothness conditions** across t_i in the sense that $D^{j-1}f$ is continuous across t_i for $j = 1, \dots, \nu$. This, finally, leads to the following B-spline property.

(6.7.2) B-spline property (vi): Local linear independence. For any knot sequence t , and any interval $I = [a \dots b] \subseteq I_{k,t}$ containing finitely many of the t_i , the sequence

$$(6.7.3) \quad \mathcal{B} := (B_{j,k}|_I : B_{j,k}|_I \neq 0)$$

is a basis for the restriction to I of the space

$$\Pi_{<k,\xi}^{(\nu)}$$

of all piecewise polynomials of order k with break sequence ξ the strictly increasing sequence containing a, b , as well as every $t_i \in I$, and satisfying $\nu_i := k - \min(k, \#\{r : t_r = \xi_i\})$ smoothness conditions across each such t_i . In particular, \mathcal{B} is linearly independent.

It is worthwhile to think about this the other way around. Suppose we start off with a partition

$$a =: \xi_1 < \xi_2 < \dots < \xi_\ell < \xi_{\ell+1} := b$$

of the interval $I := [a \dots b]$ and wish to consider the space

$$\Pi_{<k,\xi}^{(\nu)}$$

of all piecewise polynomial functions of degree $< k$ on I with breaks ξ_i that satisfy ν_i smoothness conditions at ξ_i , i.e., are $\nu_i - 1$ times continuously differentiable at ξ_i , all i . Then a B-spline basis for this space is provided by (6.7.3), with the knot sequence \mathbf{t} constructed from the break sequence ξ in the following way: To the sequence

$$(6.7.4) \quad (\underbrace{\xi_2, \dots, \xi_2}_{k-\nu_2 \text{ terms}}, \underbrace{\xi_3, \dots, \xi_3}_{k-\nu_3 \text{ terms}}, \dots, \underbrace{\xi_\ell, \dots, \xi_\ell}_{k-\nu_\ell \text{ terms}}),$$

adjoin at the beginning k points $\leq a$ and at the end k points $\geq b$. While the knots in (6.7.4) have to be exactly as shown to achieve the specified smoothness at the specified breaks, the $2k$ additional knots are quite arbitrary. They are often chosen to equal a resp. b , and this has certain advantages (among other things that of simplicity). In any case, the basic interval $I_{k,\mathbf{t}}$ for the resulting spline space is $[a..b]$, and, on this interval, it coincides with the piecewise polynomial space $\Pi_{<k,\xi}^{(\nu)}$ we started out with, – keeping in mind that we agreed earlier to make all elements of $S_{k,\mathbf{t}}$ be left-continuous at the right endpoint of $I_{k,\mathbf{t}}$.

The fact that, in this way, B-splines can be used to staff a basis for any of the spaces $\Pi_{<k,\xi}^{(\nu)}$ is also known as the **Curry-Schoenberg theorem** and has led their creator, Schoenberg, to call them ‘B-splines’ or ‘basic splines’.

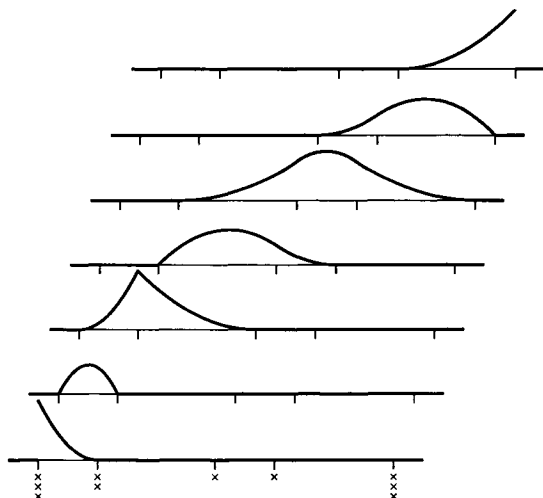


Figure 6.7.5 The B-spline basis for $\Pi_{<3,\xi}^{(\nu)}$ with $\xi = (0, 1, 3, 4, 6)$ and $\nu = (1, 2, 2)$ is, by the recipe, the quadratic B-spline sequence for the knot sequence $\mathbf{t} = (0, 0, 0, 1, 1, 3, 4, 6, 6, 6)$. Note how the smoothness of each B-spline exactly mirrors the multiplicity of each of its 4 knots.

The representation of a piecewise polynomial f as a weighted sum of B-splines is called a **B-form** for f .

Any basis $\Phi = (\varphi_1, \dots, \varphi_n)$ of a linear space F provides a unique (linear) representation $f = \sum_j \alpha_j \varphi_j$ for each $f \in F$. The usefulness of such a representation of $f \in F$ is judged in many ways.

- (i) *How robust is the representation in floating-point arithmetic with its inevitable rounding errors?* This is a question of the **condition** of the basis.
- (ii) *How easy is it to derive from the coefficient vector α the information about f that one is really interested in?* In our particular case, this concerns evaluation, differentiation, and integration, determination of zeros, etc, of a spline given in B-form.
- (iii) *How easy is it to determine the coefficient vector from some other information about f ?* In our particular case, this concerns the construction of a B-form for f by interpolation, discrete least-squares, smoothing, and the like.

In terms of these questions, the B-spline basis for $\Pi_{<k,\xi}^{(\nu)}$ does remarkably well, as is explored in the remaining sections.

6.8. DUAL FUNCTIONALS AND BLOSSOMS

Information about a basis is not complete without some information about its ‘inverse’, i.e., about the map that associates an element of the space spanned by that basis with its coordinates with respect to that basis. For the B-spline basis, this information is provided by the following explicit formula which we already met in (6.6.6).

(6.8.1) B-spline property (vii): Dual functionals. For any $f \in S_{k,t}$,

$$f = \sum_j \lambda_{jk} f B_{jk},$$

with

$$(6.8.2) \quad \lambda_{jk} f := \sum_{\nu=1}^k \frac{(-D)^{k-\nu} \psi_{jk}(\tau_j)}{(k-1)!} D^{\nu-1} f(\tau_j)$$

and $t_j+ \leq \tau_j \leq t_{j+k}-$, all j . Hence

$$(6.8.3) \quad \lambda_{ik} \left(\sum_j \alpha_j B_{jk} \right) = \alpha_i, \quad \text{all } i.$$

To be sure, there are many different dual functionals for B-splines available, but these particular ones have proven quite useful in various contexts.

As a particular example, notice that, according to (6.8.2), $\lambda_{jk} f$ depends only on part of the knot sequence \mathbf{t} , namely only on $t_{j+1}, \dots, t_{j+k-1}$, and on these it depends linearly (since ψ_{jk} does). Further, for $f \in \Pi_{<k}$, $\lambda_{jk} f$ is independent of τ_j . In other words,

$$\lambda_{jk} p = \lambda_k(t_{j+1}, \dots, t_{j+k-1}) p, \quad p \in \Pi_{<k},$$

with the precise algebraic structure of λ_k neatly captured by the following notion.

Associated with each $p \in \Pi_r$, there is a unique symmetric r -affine form called its **polar form** (in Algebra) or its **blossom** (in CAGD), denoted therefore here by

$$\overset{\omega}{p},$$

for which

$$\forall \{x \in \mathbb{R}\} p(x) = \overset{\omega}{p}(x, \dots, x).$$

E.g., the blossom of $(\cdot - \tau)^r \in \Pi_r$ is $s \mapsto (s_1 - \tau) \cdots (s_r - \tau)$. If $p = \sum_j \binom{r}{j} c_j \in \Pi_r$, then

$$\overset{\omega}{p}(s_1, \dots, s_r) := \sum_j c_j \sum_{I \subset \{1, \dots, r\}, \#I=j} \left(\prod_{i \in I} s_i \right) / \binom{r}{j}.$$

We deduce from the above that

$$\overset{\omega}{p}(t_1, \dots, t_{k-1}) = \lambda_k(t_1, \dots, t_{k-1}) p, \quad p \in \Pi_{<k}.$$

In particular, the j th B-spline coefficient of a k th order spline with knot sequence \mathbf{t} is the value at $(t_{j+1}, \dots, t_{j+k-1})$ of the blossom of *every* of the k polynomial pieces associated with the intervals $[t_i \dots t_{i+1}]$, $i = j, \dots, j+k-1$. This observation was made, in language incomprehensible to the uninitiated, by *de Casteljau* in the sixties. It was discovered independently and made plain (and given the nice name of ‘blossom’) by *Lyle Ramshaw* in the early eighties.

6.9. GOOD CONDITION

(6.9.1) B-spline property (viii): Good condition. $(B_i : i = 1:n)$ is a relatively well conditioned basis for $S_{k,\mathbf{t}}$ in the sense that there exists a positive constant $D_{k,\infty}$, which depends only on k and not on the particular knot sequence \mathbf{t} , so that for all i ,

$$(6.9.2) \quad |\alpha_i| \leq D_{k,\infty} \left\| \sum_j \alpha_j B_j \right\|_{[t_{i+1} \dots t_{i+k-1}]}.$$

Smallest possible values for $D_{k,\infty}$ are

k	2	3	4	5	6
$D_{k,\infty}$	1	3	5.5680...	12.0886...	22.7869...

Based on numerical calculations, it is conjectured that, in general,

$$D_{k,\infty} \sim 2^{k-3/2}.$$

As of 2001, the best result concerning this conjecture is to be found in [9]: $D_{k,\infty} \leq k2^{k-1}$.

6.10. CONVEX HULL

Since the $B_{j,k}$ are nonnegative, sum to 1, yet at most k are nonzero at any particular x , the next property is immediate.

(6.10.1) B-spline property (ix): Convex hull. For $t_i < x < t_{i+1}$, the value of the spline function $f := \sum_j \alpha_j B_j$ at the site x is a strictly convex combination of the k numbers $\alpha_{i+1-k}, \dots, \alpha_i$.

On the other hand, by (6.9.1), the B-spline coefficients cannot be too far from the nearby function values. Precisely, if, on the interval $[t_{i+1} \dots t_{i+k-1}]$, the spline $f = \sum_j \alpha_j B_{jk}$ is bounded from below by m and from above by M , then

$$(6.10.2) \quad |\alpha_i - (M + m)/2| \leq D_{k,\infty}(M - m)/2.$$

Much more precise estimates have become available more recently. See, for example, [8].

6.11. DIFFERENTIATION AND INTEGRATION

The striking structure of the dual functionals (6.8.2) readily provides the following formula for the derivative of a spline.

(6.11.1) B-spline property (x): Differentiation.

$$(6.11.2) \quad D\left(\sum_j \alpha_j B_{jk}\right) = (k - 1) \sum_j \frac{\alpha_j - \alpha_{j-1}}{t_{j+k-1} - t_j} B_{j,k-1}.$$

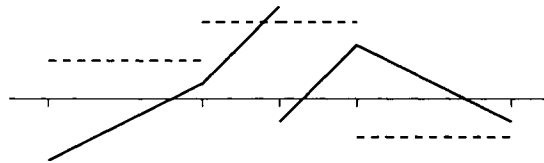


Figure 6.11.3 A piecewise linear function (solid) and its derivative (dashed) taken in the piecewise polynomial sense.

To be sure, the derivative of a spline f is taken here *in the piecewise polynomial sense*, meaning that the derivative, Df , is the piecewise polynomial whose j th polynomial piece is the derivative of the j th polynomial piece of f . In particular, if, e.g., $t_j = t_{j+k-1} < t_{j+k}$, then $B_{jk}(t_{j-}) = 0 < 1 = B_{jk}(t_{j+})$, i.e., B_{jk} has a jump across t_j and is certainly not differentiable there. However, in this case (see (6.3.4)), $B_{j,k-1}$ is just the zero function and, sticking to the useful maxim that *anything times zero is 0*, we won't have to worry about the fact that, in this case, the coefficient of $B_{j,k-1}$ in (6.11.2) involves division by zero since there is no need to compute it. In practical terms, this means that, in this case, the knot sequence for Df has one less knot (see the discussion at the end of Section 6.4).

By taking derivatives in this piecewise polynomial sense, we ensure that, for every $f \in S_{k,t}$, $Df \in S_{k-1,t}$, making (6.11.2) possible. However, this has the following, perhaps negative, consequence: When we integrate Df , we may not recover f itself since, after all, the integral of a piecewise continuous function is continuous.

It follows from (6.11.1) that $\sum_j \beta_j B_{j,k+1}$ is the *antiderivative* or *primitive* of $\sum_j \alpha_j B_{jk}$ provided

$$(6.11.4) \quad \beta_j = c + \begin{cases} \sum_{i=j_0}^j \alpha_i (t_{i+k} - t_i) / k, & j \geq j_0; \\ \sum_{i=j}^{j_0-1} \alpha_i (t_{i+k} - t_i) / k, & j < j_0, \end{cases}$$

with c and j_0 arbitrary. However, this is strictly true only in case the knot sequence is biinfinite. In the contrary case, it is only locally true since, in general, it requires infinitely many B-splines to write down the integral of a spline; see Figure 6.11.5.

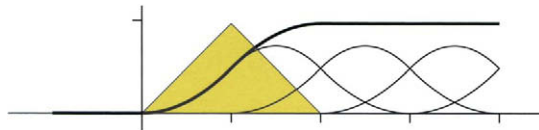


Figure 6.11.5 It takes infinitely many B-splines to express the integral of one B-spline, as is illustrated here for

$$\int_0^x B(\cdot | 0, 1, 2) = \sum_{j \geq 0} B(x | j, j + 1, j + 2, j + 3).$$

6.12. EVALUATION

The recurrence relations (6.2.2) lead directly to a stable algorithm for the evaluation of a spline

$$s = \sum_i a_i B_{ik}$$

from its B-spline coefficients (a_i).

The recurrence relations imply

$$s = \sum_i a_i B_{ik} = \sum_i a_i^{[1]} B_{i,k-1},$$

with

$$(6.12.1) \quad a_i^{[1]} := (1 - \omega_{ik}) a_{i-1} + \omega_{ik} a_i.$$

Note that $a_i^{[1]}$ is not a constant, but is the straight line through the points (t_i, a_{i-1}) and (t_{i+k-1}, a_i) . In particular, $a_i^{[1]}(t)$ is a *convex combination* of a_{i-1} and a_i if $t_i \leq t \leq t_{i+k-1}$.

After $k - 1$ -fold iteration of this procedure, we arrive at the formula

$$s = \sum_i a_i^{[k-1]} B_{i1},$$

which shows that

$$s = a_i^{[k-1]} \quad \text{on } [t_i \dots t_{i+1}).$$

(6.12.2) Evaluation algorithm. From given constant polynomials $a_i^{[0]} := a_i, i = j - k + 1, \dots, j$, (which determine $s := \sum_i a_i B_{ik}$ on $[t_j \dots t_{j+1})$), generate polynomials $a_i^{[r]}, r = 1, \dots, k - 1$, by the recurrence

$$(6.12.3) \quad a_i^{[r+1]} := (1 - \omega_{i,k-r})a_{i-1}^{[r]} + \omega_{i,k-r}a_i^{[r]}, \quad j - k + r + 1 < i \leq j.$$

Then $s = a_j^{[k-1]}$ on $[t_j \dots t_{j+1})$. Moreover, for $t_j \leq t \leq t_{j+1}$, the weight $\omega_{i,k-r}(t)$ in (6.12.3) lies between 0 and 1. Hence the computation of $s(t) = a_j^{[k-1]}(t)$ via (6.12.3) consists of the repeated formation of convex combinations.

In the *cardinal case* (6.5.1-6.5.2), the algorithm simplifies, as follows. Now

$$s =: \sum_i N_k(\cdot - i)a_i = \sum_i N_{k-1}(\cdot - i)a_i^{[1]}/(k - 1),$$

with

$$a_i^{[1]} := (i + k - 1 - \cdot)a_{i-1} + (\cdot - i)a_i.$$

Hence

$$(6.12.3)_Z \quad s = a_j^{[k-1]}/(k - 1)! \text{ on } [j \dots j + 1), \quad \text{with}$$

$$a_i^{[r]} := (i + k - r - \cdot)a_{i-1}^{[r-1]} + (\cdot - i)a_i^{[r-1]}, \quad j - k + r < i \leq j.$$

In the *Bernstein-Bézier case* (6.5.4-6.5.5), all the nontrivial weight functions $\omega_{i,k-r}$ are the same, i.e.,

$$\omega_{i,k-r}(t) = t.$$

Thus, for

$$s = \sum_{\mu+\nu=h} a_{(\mu,\nu)} B_{(\mu,\nu)},$$

we get

$$(6.12.3)_B \quad s = a_{(0,0)} \text{ on } [0 \dots 1], \quad \text{with}$$

$$a_{(\mu,\nu)}(t) = (1 - t)a_{(\mu+1,\nu)} + ta_{(\mu,\nu+1)}, \quad \mu + \nu = r; \quad r = h - 1, \dots, 0.$$

This is **de Casteljau's algorithm** for the evaluation of the BB-form.

6.13. SPLINE FUNCTIONS VS SPLINE CURVES

So far, we have only dealt with spline *functions*, even though CAGD is mainly concerned with spline *curves*. The distinction is fundamental.

Every spline function $f = \sum_j \alpha_j B_{jk}$ gives rise to (planar) curve, namely its **graph**, i.e., the pointset

$$\{(x, f(x)) : x \in I_{k,t}\}.$$

Assuming that $\#t_j < k$ for all interior knots t_j , this is indeed a **curve** in the mathematical sense, i.e., the continuous image of an interval. Its natural parametrization is the **spline**

curve

$$(6.13.1) \quad x \mapsto (x, f(x)) = \sum_j P_j B_{jk}(x),$$

with

$$P_j := (t_{jk}^*, \alpha_j)$$

its j th **control point**, and the equality in (6.13.1) is justified by (6.6.9).

However, spline curves are not restricted to control points of this specific form. By choosing the control points P_j in (6.13.1) in any manner whatsoever as d -vectors, we obtain a spline curve in \mathbb{R}^d that smoothly follows the shape outlined by its **control polygon**, which is the broken line that connects these points P_j in order.

Note the CAGD-standard use of the term ‘spline curve’ to denote both, a curve that can be parametrized by a spline, and the (vector-valued) spline that provides this parametrization.

6.14. KNOT INSERTION

Wolfgang Böhm (see, e.g., [3]) was the first to point out that the evaluation algorithm (6.12.2) can be interpreted as repeated knot insertion. This CAGD insight into B-splines has had many wonderful repercussions.

(6.14.1) B-spline property (xi): Knot insertion. *If the knot sequence $\hat{\mathbf{t}}$ is obtained from the knot sequence \mathbf{t} by the insertion of just one term, x say, then, for any $f \in S_{k,\mathbf{t}}$, $\sum_j \alpha_j B_{j,k;\mathbf{t}} := f =: \sum_j \hat{\alpha}_j B_{j,k;\hat{\mathbf{t}}}$ with*

$$(6.14.2) \quad \hat{\alpha}_j = (1 - \hat{\omega}_{jk}(x))\alpha_{j-1} + \hat{\omega}_{jk}(x)\alpha_j, \quad \text{all } j,$$

and $\hat{\omega}_{jk} := \max\{0, \min\{1, \omega_{jk}\}\}$, i.e.,

$$(6.14.3) \quad \hat{\omega}_{jk} : x \mapsto \begin{cases} 0, & \text{for } x \leq t_j; \\ \omega_{jk}(x) = \frac{x - t_j}{t_{j+k-1} - t_j}, & \text{for } t_j < x < t_{j+k-1}; \\ 1, & \text{for } t_{j+k-1} \leq x. \end{cases}$$

Note the need here to make the dependence of a B-spline on its knot sequence explicit in the notation. This property has the following pretty geometric interpretation, in terms of the *control polygon*

$$C_{k,\mathbf{t}}f$$

of $f \in S_{k,\mathbf{t}}$.

(6.14.4) Proposition. *If $\hat{\mathbf{t}}$ is obtained from \mathbf{t} by the insertion of one additional knot, then, for any $f \in S_{k,\mathbf{t}}$, $C_{k,\hat{\mathbf{t}}}f$ interpolates, at its breakpoints, to $C_{k,\mathbf{t}}f$ (and is thereby uniquely determined).*

Figure 6.14.5 illustrates this interpretation.

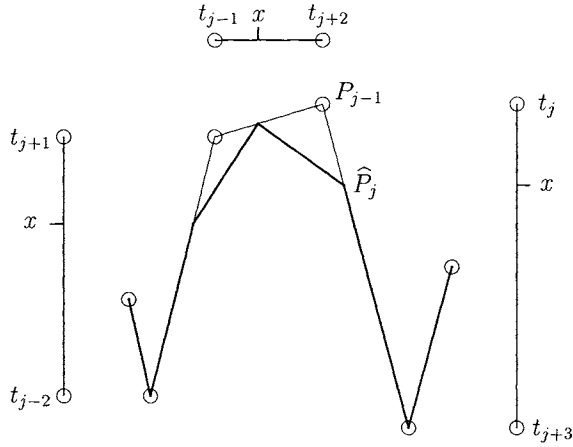


Figure 6.14.5 Insertion of $x = 2$ into the knot sequence $\mathbf{t} = (0,0,0,0,1,3,5,5,5,5)$, with $k = 4$.

By repeated insertion of the point x until its multiplicity in the resulting knot sequence $\tilde{\mathbf{t}}$ is $k - 1$, we arrive at the B-form

$$\sum_j \tilde{\alpha}_j B_{j,\tilde{\mathbf{t}}}$$

for $f = \sum_j \alpha_j B_{j,\mathbf{t}}$, in which there is exactly one B-spline $B_{j,\tilde{\mathbf{t}}}$ not zero at x . Since B-splines always sum up to 1, its coefficient must be the value of f at x . This is illustrated in Figure 6.14.6.

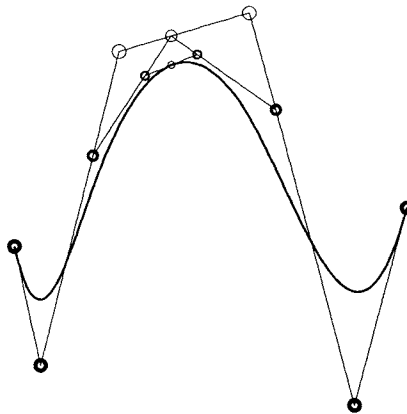


Figure 6.14.6 Three-fold insertion of the same knot provides a point on the graph of a cubic spline.

6.15. VARIATION DIMINUTION AND SHAPE PRESERVATION: SCHOENBERG'S OPERATOR

The spline $f = \sum_j \alpha_j B_{j,k,t}$ can be viewed as the result of applying to its control polygon $C_{k,t}$ Schoenberg's operator $V = V_k = V_{k,t}$, as given by

$$V_k g := \sum_j g(t_{jk}^*) B_{jk}.$$

Schoenberg's operator is **variation-diminishing**, meaning that, for any continuous function g , Vg crosses the x -axis no more often than does g . More than that, any crossing of Vg requires a 'nearby' crossing of g .

Here is a formal statement, in which $f := Vg$ and $\alpha_j := g(t_{jk}^*)$, and which follows immediately from knot insertion.

(6.15.1) B-spline property (xii): Variation diminution. *If $f = \sum_j \alpha_j B_{j,k,t}$ and $\tau_1 < \dots < \tau_r$ are such that $f(\tau_{i-1})f(\tau_i) < 0$, all i , then one can find indices $1 \leq j_1 < \dots < j_r \leq n$ so that*

$$(6.15.2) \quad \alpha_{j_i} f(\tau_i) B_{j_i}(\tau_i) > 0 \quad \text{for } i = 1, \dots, r.$$

Further, by (6.6.9),

$$V\ell = \ell, \quad \ell \in \Pi_1.$$

This implies that Vg crosses any particular straight line ℓ no more often than does g , and with the crossing of Vg closely related to the crossings of g , including the direction of the crossing.

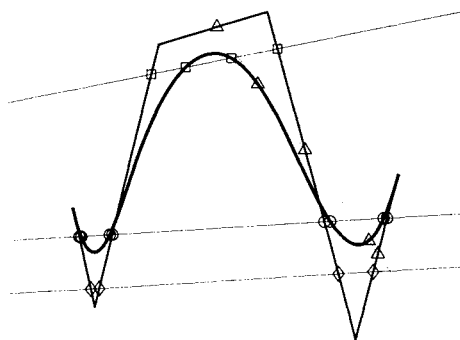


Figure 6.15.3 A cubic spline, its control polygon, and various straight lines intersecting them. The control polygon *exaggerates* the shape of the spline. The spline crossings are bracketed by the control polygon crossings.

This is illustrated in Figure 6.15.3 for Vg a spline and g its control polygon. In particular, if g is monotone, then so is Vg ; if g is convex, then so is Vg . It is in this sense that Schoenberg’s operator is **shape-preserving**.

In effect, a spline is a smoothed version of its control polygon.

6.16. ZEROS OF A SPLINE, COUNTING MULTIPLICITY

Since a spline (function) cannot cross the x -axis more often than does its control polygon, the number of sign changes in its coefficient sequence is an upper bound on the number of its zeros.

Things are a bit more subtle when one would like to include in the zero count the **multiplicity** of a zero, defined as the maximal number of distinct nearby zeros in a nearby spline (from the same spline space), and needed when considering osculatory or Hermite interpolation by splines.

Here is one relevant result. The full story is recounted in [6].

(6.16.1) Proposition. *If $f = \sum_j \alpha_j B_{j,k,t}$ is zero at $x_1 < \dots < x_r$, while $f^t := \sum_j |\alpha_j| B_{j,k,t}$ is not, then $S^-(\alpha) \geq r$, with $S^-(\alpha)$ the smallest number of sign changes in the sequence α obtainable by assigning the sign of any zero entry of α appropriately.*

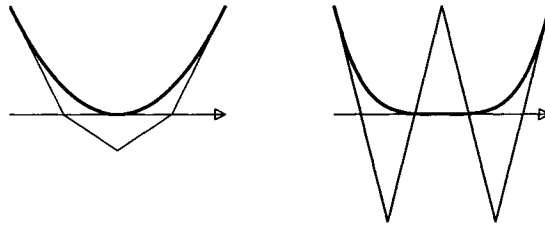


Figure 6.16.2 A double spline zero, and a quadruple spline zero, and corresponding control polygons.

6.17. SPLINE INTERPOLATION: SCHOENBERG-WHITNEY

Spline interpolation is one ready means for constructing a spline function that satisfies certain conditions. In **spline interpolation**, one seeks a spline that matches given data values y_i at given data sites $x_i, i = 1, \dots, n$. If the spline interpolant is to be a spline of order k with knot sequence \mathbf{t} , then we can write the sought-for spline in B-form, $\sum_j \alpha_j B_{jk}$, hence we are looking for a solution α to the linear system

$$(6.17.1) \quad \sum_j \alpha_j B_{jk}(x_i) = y_i, \quad i = 1, \dots, n.$$

This linear system has exactly one solution for every choice of data values y_i exactly when its coefficient matrix is invertible. This motivates the next result, which is a ready consequence of Proposition 6.16.1.

(6.17.2) Schoenberg-Whitney Theorem. Assume that all interior knots in the knot sequence $\mathbf{t} = (t_1, \dots, t_{n+k})$ have multiplicity $< k$, hence each $f \in S_{k,\mathbf{t}}$ is continuous (on its basic interval, $I_{k,\mathbf{t}}$). Let $\mathbf{x} := (x_1 < \dots < x_n)$ be a strictly increasing sequence in $I_{k,\mathbf{t}}$. Then, the collocation matrix

$$A_{\mathbf{x}} := (B_{jk}(x_i) : i, j = 1, \dots, n)$$

is invertible if and only if all its diagonal entries (namely the numbers $B_{jk}(x_j)$), are non-zero, i.e., if and only if

$$(6.17.3) \quad t_i \leq x_i \leq t_{i+k}, \quad i = 1, \dots, n,$$

with equality occurring only if the knot in question is one of the endpoints of the basic interval, $I_{k,\mathbf{t}}$.

The strict ordering of the data-site sequence \mathbf{x} not only leads to this neat characterization of the invertibility of the collocation matrix $A_{\mathbf{x}}$. It also ensures (see (6.4.2)) that $A_{\mathbf{x}}$ is a banded matrix with at most k nontrivial bands. It also ensures that $A_{\mathbf{x}}$ is **totally positive**, meaning that all its minors (i.e., determinants of submatrices) are nonnegative. This somewhat esoteric property has many consequences of practical interest. One of these is that it is numerically safe to solve the linear system (6.17.1) by Gauss elimination *without* pivoting, hence in no more storage than is required to store the banded matrix $A_{\mathbf{x}}$ to begin with.

If the knot sequence \mathbf{t} and the order k are already chosen, then the sequence $(t_i^* : i = 1, \dots, n)$ of Greville points is a good choice as data site sequence \mathbf{x} ; it certainly satisfies the **Schoenberg-Whitney conditions** (6.17.3). It also serves as a good initial guess in the iterative process for determining the **Chebyshev-Demko** sites, \mathbf{x}^* . These are optimal sites for interpolation from $S_{k,\mathbf{t}}$ in that the resulting map, $f \mapsto P_{\mathbf{x}^*}f$, is the most stable among all possible such maps $f \mapsto P_{\mathbf{x}}f$. In consequence, $P_{\mathbf{x}^*}f$ is a near-best approximation to f from $S_{k,\mathbf{t}}$ in that $\|f - P_{\mathbf{x}^*}f\| \leq \text{const}_k \text{dist}(f, S_{k,\mathbf{t}})$ for some f -independent const_k .

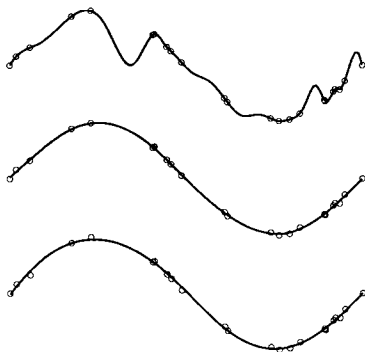


Figure 6.17.4 A spline interpolant (top) to noisy data (circled) may be unnecessarily wiggly. A smoothing spline (middle) or a least-squares approximant (bottom) to such data may be preferred.

When the data values y_i are noisy, then spline interpolation may produce a highly oscillating spline, as is illustrated in Figure (6.17.4). In such a situation, one may be willing to forego exact matching in favor of a ‘smoother’, less wiggly, approximating spline. There are two standard procedures to accomplish this: the *smoothing spline* and *least-squares spline approximant*, and Figure (6.17.4) also shows a sample of both.

6.18. SMOOTHING SPLINE

The smoothing spline is constructed as a compromise between the wish to be close to the data and the wish for a smooth approximation. Closeness of the function f to the data (x_i, y_i) is typically measured by the sum of squares of their difference:

$$E(f) := \sum_i (y_i - f(x_i))^2,$$

while roughness of f is measured by the size of some derivative of f in the mean-square norm:

$$R(f) := \int_a^b (D^m f(t))^2 dt,$$

with $[a..b]$ the interval of interest. Both measures could involve some weighting function, though this is more commonly done for E than for R .

Choosing mean-square norms for both E and F ensures that, for any positive p , the minimizer $f = f_p$ of the weighted sum

$$E(f) + pR(f)$$

is a spline, of order $2m$ and with simple knots, at the data sites, and reducing to a polynomial of degree $< m$ outside the interval $(x_1..x_n)$. As $p \rightarrow 0$, this so-called **smoothing spline** converges to the so-called ‘natural’ spline interpolant of order $2m$ to the given data. At the other extreme, as $p \rightarrow \infty$, the smoothing spline converges to the least-squares approximant to the data by polynomials of degree $< m$.

It is something of an art to choose the smoothing parameter ‘appropriately’. The most popular choice is based on *generalized cross validation*; see [12].

6.19. LEAST-SQUARES SPLINE APPROXIMATION

The perhaps somewhat vague notion behind least-squares approximation is to work with a spline with just enough degrees of freedom to fit the ‘smooth’ function underlying the noisy data, but not enough degrees of freedom to match also the noise.

In practice, this means that one must somehow choose the order, k , and the knot sequence $\mathbf{t} = (t_1, \dots, t_{N+k})$, mindful that (1) the resulting basic interval $I_{k,\mathbf{t}}$ equal the interval $[a..b]$ of interest; and (2) that $S_{k,\mathbf{t}}$ contain a unique minimizer of E . The latter is ensured exactly when some subsequence of the data sites \mathbf{x} satisfies the Schoenberg-Whitney conditions with respect to the chosen knot sequence \mathbf{t} . In that case, it is usually numerically safe to determine the B-spline coefficient vector $\boldsymbol{\alpha}$ of the least-squares spline approximant as the solution to the normal equations

$$A'_x A_x \boldsymbol{\alpha} = A'_x \mathbf{y},$$

with A_x , as before, the B-spline collocation matrix for the chosen order k and knot sequence \mathbf{t} and the given data sites \mathbf{x} .

The least-squares fit in Figure (6.17.4) is a cubic spline, with 3 equally-spaced interior knots.

When approximating a function with widely varying behavior, it is tempting to choose the location of these interior knots so as to further minimize the error, but the best one can hope for is a choice that cannot be improved upon by small local variations of the knot locations.

6.20. BACKGROUND

I have failed, except coincidentally, to supply historical comment or attribute specific results to specific authors. Nor was there any attempt to prove the results stated, not even in outline. For all these matters, consult the standard literature.

The relevant literature on (univariate) B-splines up to about 1975 is summarized in [4] which also contains hints of the most exciting developments concerning B-splines since then: knot insertion and the multivariate B-splines. Two books on splines, [5] and [11], which have appeared since 1975, cover B-splines in the traditional way. As presentations of splines from the CAGD point of view, the survey article [3] and the “Killer B’s” [1,87] are particularly recommended. The revised version, [7], of [5] develops the central part of spline theory more in the spirit of CAGD and, in particular, knot insertion. All the results mentioned here are proved there.

REFERENCES

1. R.H. Bartels, J.C. Beatty, and B.A. Barsky. An introduction to the use of splines. In *Computer Graphics*. SIGGRAPH’85, San Francisco, 1985.
2. R.H. Bartels, J.C. Beatty, and B.A. Barsky. *An Introduction to Splines for Use in Computer Graphics & Geometric Modeling*. Morgan Kaufmann, Los Altos, CA, 1987.
3. W. Böhm, G. Farin, and J. Kahmann. A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design*, 1:1–60, 1984.
4. C. de Boor. Splines as linear combinations of B-splines. In *Approximation Theory II*, G.G. Lorentz, C.K. Chui, and L.L. Schumaker, editors, pages 1–47, 1976.
5. C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, New York, 1978.
6. C. de Boor. The multiplicity of a spline zero. *Annals of Numer. Math.* 4:229–238, 1997.
7. C. de Boor. *A Practical Guide to Splines*, Revised Edition. Springer-Verlag, New York, 2001.
8. D. Lutterkort and J. Peters. Tight linear bounds on the distance between a spline and its B-spline control polygon. *Manuscript*, 1999.
9. K. Scherer and A. Yu. Shadrin. New upper bound for the B-spline basis condition number. II. A proof of de Boor’s 2^k -conjecture. *J. Approx. Theory*, 99(2), 217–229.
10. I.J. Schoenberg. Cardinal interpolation and spline functions. *J. Approximation Theory* 2:167–206, 1969.
11. L.L. Schumaker. *Spline Functions*. J. Wiley, New York, 1981.

12. G. Wahba. *Spline Models for Observational Data*. CBMS-NSF Regional Conference Series in Applied Mathematics 59, SIAM (Philadelphia, PA), 1990.

Carl de Boor
supported by ARO under Grant DAAG55-98-1-0443

Chapter 7

Curve and Surface Constructions

Dianne Hansford and Gerald Farin

This chapter introduces algorithms for the generation of curves and surfaces. The emphasis is on interpolation and approximation using Bézier and B-spline techniques.

7.1. INTRODUCTION

The goal of this chapter is to outline some of the most fundamental interpolation and approximation methods in CAGD. Wherever possible, the developments focus on Bézier and B-spline techniques because of their intuitive geometric definitions. First of all, the focus is on polynomial curve methods, including Lagrange (point) interpolation, point approximation, and Hermite (point and tangent) interpolation. Next, a piecewise polynomial scheme, C^2 cubic spline interpolation is presented. The focus then moves to surface methods. The topics include: interpolation to boundary curve data with Coons patches, interpolation to rectangular data with tensor product surfaces, approximation to large sets of data, and interpolation to point and derivative data. Mirroring the curve presentation, a piecewise polynomial surface scheme, C^2 bicubic spline interpolation, is discussed. The chapter concludes with volume deformations.

For more information on these topics, a good starting point is one of the following textbooks: [4,10,12,19,26]. The Bézier Techniques and B-spline Basics chapters 4, 6 in this handbook provide an introduction to much of the theory used here.

7.2. POLYNOMIAL CURVE METHODS

Polynomial curve interpolation is a theoretical cornerstone of CAGD. The first topic investigated in this section is the most basic formation of this problem: point data interpolation. Due to its importance, this problem is attacked from three viewpoints, and finally its weakness are examined. Next approximation is examined as a means to overcome the limits of interpolation. This section concludes with point and tangent data interpolation, an important building block for piecewise polynomial schemes.

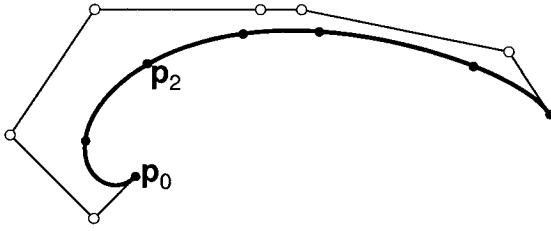


Figure 7.1. Point data interpolation: given $n + 1$ data points with parameter values, find the interpolating degree n polynomial.

7.2.1. Point Data Interpolation

Many scientific applications deal with the problem of fitting a curve to discrete point data. In other words, as illustrated in Figure 7.1, given $n + 1$ data points \mathbf{p}_i with associated parameters t_i , find a degree n polynomial curve $\mathbf{p}(t)$ such that

$$\mathbf{p}(t_i) = \mathbf{p}_i, \quad i = 0, \dots, n$$

This problem is called *point data interpolation*.

The t_i may be thought of as time increments; they indicate how much time a particle moving along the curve must spend between data points. These parameters greatly influence the fitting method, and if their values are not intrinsic to the application, they must be carefully assigned. This topic is addressed in Section 7.3.2.

A Direct Approach

Typically, Bézier curves are the preferred representation due to their stability properties (see Farouki and Rajan [13]). Thus the interpolating polynomial with respect to the Bernstein basis takes the form

$$\mathbf{p}(t) = \sum_{j=0}^n \mathbf{b}_j B_j^n(t),$$

where the \mathbf{b}_j are *Bézier points* and the $B_j^n(t)$ are the Bernstein polynomials. The “direct approach” simply means to directly apply the known relationships:

$$\begin{aligned} \mathbf{p}(t_0) = \mathbf{p}_0 &= \mathbf{b}_0 B_0^n(t_0) + \mathbf{b}_1 B_1^n(t_0) + \dots + \mathbf{b}_n B_n^n(t_0), \\ \mathbf{p}(t_1) = \mathbf{p}_1 &= \mathbf{b}_0 B_0^n(t_1) + \mathbf{b}_1 B_1^n(t_1) + \dots + \mathbf{b}_n B_n^n(t_1), \\ &\vdots \\ \mathbf{p}(t_n) = \mathbf{p}_n &= \mathbf{b}_0 B_0^n(t_n) + \mathbf{b}_1 B_1^n(t_n) + \dots + \mathbf{b}_n B_n^n(t_n), \end{aligned}$$

which are $n + 1$ equations for the $n + 1$ unknowns for each coordinate of the \mathbf{b}_j . In matrix form:

$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_n \end{bmatrix} = \begin{bmatrix} B_0^n(t_0) & B_1^n(t_0) & B_2^n(t_0) & \dots & B_n^n(t_0) \\ B_0^n(t_1) & B_1^n(t_1) & B_2^n(t_1) & \dots & B_n^n(t_1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_0^n(t_n) & B_1^n(t_n) & B_2^n(t_n) & \dots & B_n^n(t_n) \end{bmatrix} \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_n \end{bmatrix}, \quad (7.1)$$

or $\mathbf{p} = \mathbf{B}\mathbf{b}$. The matrix B is called the *generalized Vandermonde* of the interpolation problem. One can show that the determinant of B is nonzero, and therefore a unique solution to the interpolation problem exists. The direct approach results in two or three (depending on the dimensionality of the data points) linear systems with the same coefficient matrix, therefore it is best to construct the LU decomposition of B .

Instead of using the Bernstein basis, one could choose any basis. The monomials, $1, t, t^2, \dots, t^n$, have been the historical favorite (see Davis [7]); in this case, the matrix B is simply called the *Vandermonde*.

Aitken's Algorithm

If one would simply like to *evaluate* the interpolating polynomial at specific t parameters, then knowledge of the coefficients of the interpolant is not necessary. Such an evaluation method is *Aitken's algorithm*.

The basic idea behind this algorithm is that polynomials may be expressed as linear combinations of lower degree polynomials.¹ Thus Aitken's algorithm computes a point on the interpolating polynomial through a sequence of *repeated linear interpolations*. Given parameter values t_i and the data points $\mathbf{p}_i^0 = \mathbf{p}_i$, compute

$$\mathbf{p}_i^r(t) = \frac{t_{i+r} - t}{t_{i+r} - t_i} \mathbf{p}_i^{r-1}(t) + \frac{t - t_i}{t_{i+r} - t_i} \mathbf{p}_{i+1}^{r-1}(t); \quad \begin{cases} r = 1, \dots, n; \\ i = 0, \dots, n - r. \end{cases} \quad (7.2)$$

Aitken's algorithm has the following geometric interpretation: \mathbf{p}_i^r is the result of mapping t with respect to the interval $[t_i, t_{i+r}]$ onto the straight line segment through $\mathbf{p}_i^{r-1}, \mathbf{p}_{i+1}^{r-1}$. The geometry of Aitken's algorithm is illustrated in Figure 7.2 for a quadratic example. The intermediate \mathbf{p}_i^r are computed from two points from the $(r - 1)^{\text{st}}$ stage, giving the algorithm, for example for the cubic case, the following structure:

$$\begin{array}{l} \mathbf{p}_0 \\ \mathbf{p}_1 \quad \mathbf{p}_0^1 \\ \mathbf{p}_2 \quad \mathbf{p}_1^1 \quad \mathbf{p}_0^2 \\ \mathbf{p}_3 \quad \mathbf{p}_2^1 \quad \mathbf{p}_1^2 \quad \mathbf{p}_0^3 \end{array} \quad (7.3)$$

To prove that Aitken's algorithm (7.2) interpolates, suppose that the following two interpolation problems have been solved.

1. \mathbf{p}_0^{n-1} is the polynomial which interpolates to the first n data points, $\mathbf{p}_0, \dots, \mathbf{p}_{n-1}$.
2. \mathbf{p}_1^{n-1} interpolates to the last n data points, $\mathbf{p}_1, \dots, \mathbf{p}_n$.

¹Note the similarities with the de Casteljau algorithm.

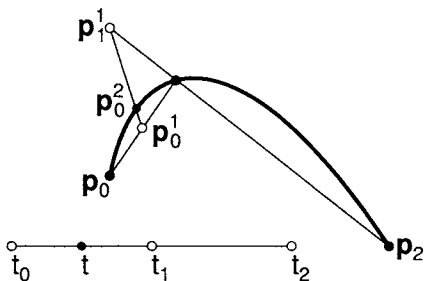


Figure 7.2. Aitken's algorithm: a point on an interpolating polynomial may be found from repeated linear interpolation.

Aitken's algorithm defines the final interpolant as

$$\mathbf{p}_0^n(t) = \frac{t_n - t}{t_n - t_0} \mathbf{p}_0^{n-1}(t) + \frac{t - t_0}{t_n - t_0} \mathbf{p}_1^{n-1}(t). \quad (7.4)$$

Figure 7.3 illustrates this form for a cubic example. One needs to verify that (7.4) does in fact interpolate to all given data points \mathbf{p}_i . For $t = t_0$ interpolation holds:

$$\mathbf{p}_0^n(t_0) = 1 * \mathbf{p}_0^{n-1}(t_0) + 0 * \mathbf{p}_1^{n-1}(t_0) = \mathbf{p}_0.$$

A similar result is derived for $t = t_n$. The initial assumption was that $\mathbf{p}_0^{n-1}(t_i) = \mathbf{p}_1^{n-1}(t_i) = \mathbf{p}_i$ for all other values of i , thus since the weights in (7.4) sum to one identically, $\mathbf{p}_0^n(t_i) = \mathbf{p}_i$.

One can infer several properties of the interpolating polynomial from Aitken's algorithm:

- *Affine invariance*: this follows since Aitken's algorithm uses only barycentric combinations.
- *Linear precision*: If all \mathbf{p}_i are uniformly distributed² on a straight line segment, all intermediate $\mathbf{p}_i^r(t)$ are identical for $r > 0$. Thus the straight line segment is reproduced.
- *No convex hull property*: the parameter t in (7.2) does not have to lie between t_i and t_{i+r} . Therefore, Aitken's algorithm does not use convex combinations only: $\mathbf{p}_0^n(t)$ is not guaranteed to lie within the convex hull of the \mathbf{p}_i . One should note, however, that no smooth curve interpolation scheme exists that has the convex hull property.
- *No variation diminishing property*: if a straight line intersects the polygon connecting the data points m times, then the line can intersect the curve more than m times. In other words, the curve wiggles more than the polygon. This follows for the same reason there is no convex hull property.

²If the points are on a straight line, but distributed unevenly, the graph of the straight line will be recaptured, but it will not be parametrized linearly.

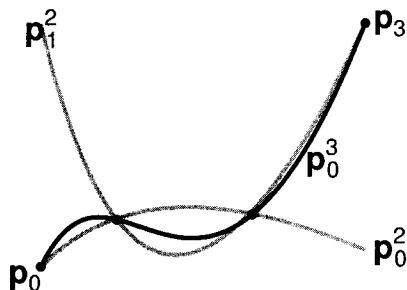


Figure 7.3. Polynomial interpolation: a cubic interpolating polynomial may be obtained as a “blend” of two quadratic interpolants.

The Cardinal Form

The *cardinal form* of a curve or surface representation is the one in which the given data appear explicitly. With regard to our interpolation problem, the cardinal form would take the form

$$\mathbf{p}(t) = \sum_{j=0}^n \mathbf{p}_j L_j^n(t). \quad (7.5)$$

The basis functions $L_j^n(t)$ can be defined by examining the properties which they must possess. Of course they must sum to one in order for (7.5) to be a barycentric combination. Additionally, the $L_j^n(t)$ must satisfy

$$L_i^n(t_j) = \delta_{i,j}, \quad (7.6)$$

with $\delta_{i,j}$ being the Kronecker delta. In other words, the i^{th} Lagrange polynomial vanishes at all knots except at the i^{th} one, where it assumes the value 1. From this information, one can conclude that the *Lagrange polynomials* L_i^n take the form

$$L_i^n(t) = \frac{\prod_{\substack{j=0 \\ j \neq i}}^n (t - t_j)}{\prod_{\substack{j=0 \\ j \neq i}}^n (t_i - t_j)}. \quad (7.7)$$

Figure 7.4 illustrates one such polynomial over a particular knot sequence.

The cardinal form is primarily used for theoretical analysis. In particular, the Lagrange polynomials are useful in answering the following questions:

- Is the interpolating polynomial unique?
- What is a closed form for the interpolating polynomial?

The interpolant in (7.5) is the only representation in which the data appear explicitly. Therefore, it is often referred to as *the* interpolating polynomial or the Lagrange interpolant even though it could be written in another basis, as illustrated in Section 7.2.1

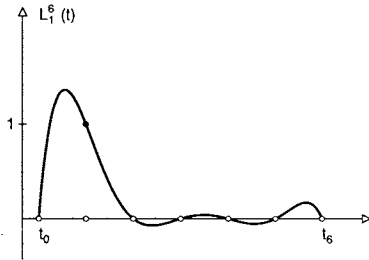


Figure 7.4. A Lagrange polynomial

Asking Too Much of a Polynomial

In practical curve fitting scenarios, it is likely that the number of data points exceeds ten. As the number of points increases, the main drawback of polynomial interpolation becomes apparent, as illustrated in Figure 7.5: polynomial interpolants may *oscillate*. The left curve in that figure is the Lagrange interpolant to 21 points read off a quarter of an ellipse. The data points were computed to a precision of six digits. Slightly changing the input data points, namely by reducing their accuracy to four digits, produces the right interpolant. This is a disturbing phenomenon: minuscule changes in the input data may result in serious changes of the result. Processes with that behavior are called *ill-conditioned*.

This tendency of polynomial interpolants to oscillate has been studied extensively in numerical analysis, where it is known as the “Runge phenomenon” [27]. Sample a small number of points and parameter values from a smooth curve for interpolation, and then gradually increase the number of points. One would expect the interpolant to converge to the underlying, smooth curve, however, this is not the case in general. For some sampled curves, the interpolant diverges. This phenomenon is not due to numerical effects; it is actually inherent in the polynomial interpolation process. An interesting observation is that this does not contradict the Weierstrass approximation theorem.

7.2.2. Point Data Approximation

The oscillatory nature of high degree polynomial interpolation, as discussed in Section 7.2.1, prompts one to search for a better solution. Approximation of given data by a low degree curve which passes “close” to the data points is a practical solution, as illustrated by Figure 7.6. The added benefit of approximation is a smoothing effect: this is favorable if an application produces data points which are noisy. The first step in building an approximation scheme is to consider a metric to measure closeness. Also, the method should produce a unique solution once given such a metric.

Least squares approximation is the most widely used approximation method; it is simple to employ and it uses a familiar metric. The given information consists of $m + 1$ data points \mathbf{p}_i with associated parameter values t_i . Find a degree n polynomial \mathbf{p} such that



Figure 7.5. Lagrange interpolation: The left and right input data only differ by the amount of accuracy: six digits after the decimal point, left; four digits, right.

the squared distances $\|\mathbf{p}_i - \mathbf{p}(t_i)\|^2$ are minimal. To make this more concrete, suppose that \mathbf{p} is a Bézier curve, thus find the \mathbf{b}_j which minimize the function

$$f(\mathbf{b}_0, \dots, \mathbf{b}_n) = \sum_{i=0}^m \left\| \mathbf{p}_i - \sum_{j=0}^n \mathbf{b}_j B_j^n(t_i) \right\|^2.$$

The minimization of f may be done componentwise, thus with a slight abuse of notation, formulate the $n + 1$ equations (for each component) as

$$\frac{\partial f}{\partial \mathbf{b}_j} = 0.$$

After differentiating, these $n + 1$ equations take the form

$$\sum_{i=0}^m \left[\mathbf{p}_i - \sum_{j=0}^n \mathbf{b}_j B_j^n(t_i) \right] B_k^n(t_i) = 0; \quad k = 0, \dots, n, \tag{7.8}$$

and are referred to as the *normal equations*.

Alternatively, one may formulate the approximation problem similarly to the direct approach from Section 7.2.1. Simply write down the given conditions!

$$\begin{aligned} \mathbf{b}_0 B_0^n(t_0) + \dots + \mathbf{b}_n B_n^n(t_0) &= \mathbf{p}_0 \\ &\vdots \end{aligned}$$

$$\mathbf{b}_0 B_0^n(t_m) + \dots + \mathbf{b}_n B_n^n(t_m) = \mathbf{p}_m.$$

This may be condensed into matrix form:

$$\begin{bmatrix} B_0^n(t_0) & \dots & B_n^n(t_0) \\ & \ddots & \\ & & \\ B_0^n(t_m) & \dots & B_n^n(t_m) \end{bmatrix} \begin{bmatrix} \mathbf{b}_0 \\ \vdots \\ \mathbf{b}_n \end{bmatrix} = \begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_m \end{bmatrix},$$

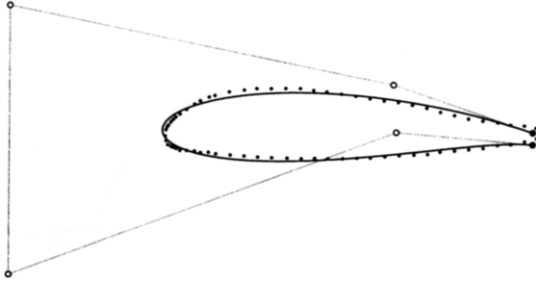


Figure 7.6. Least squares approximation: data points are sampled from the cross section of an airplane wing and a quintic Bézier curve is fitted to them.

or

$$M\mathbf{B} = \mathbf{P}. \quad (7.9)$$

Assuming the number of data points, $m + 1$, is larger than the degree n of the curve, this linear system is overdetermined. One may simply multiply both sides by M^T :

$$M^T M \mathbf{B} = M^T \mathbf{P}. \quad (7.10)$$

This is a linear system with $n + 1$ equations in $n + 1$ unknowns with a square and symmetric coefficient matrix $M^T M$. The B_i^n are assumed to be linearly independent, thus the $n + 1$ columns of M are linearly independent, and this ensures full rank of $M^T M$. Notice that (7.10) is identical to (7.8), thus this solution also minimizes the L^2 norm.

This direct approach for constructing an approximation allows for an additional approximation tool. It may be the case that even the least squares approximation produces a curve that wiggles too much, as illustrated in Figure 7.7. Another defect of the solution in this figure is the wildness of the control polygon. One benefit of the Bézier method is that the polygon often times is a good approximation of the curve's shape. To achieve this, one could impose restrictions on the control polygon, for example minimize the wiggles in the second differences:

$$\mathbf{b}_0 - 2\mathbf{b}_1 + \mathbf{b}_2 = \mathbf{0}$$

$$\vdots$$

$$\mathbf{b}_{n-2} - 2\mathbf{b}_{n-1} + \mathbf{b}_n = \mathbf{0},$$

which may be abbreviated as

$$S\mathbf{B} = \mathbf{0}. \quad (7.11)$$

Simply add these equations to the overdetermined system (7.9),

$$\begin{bmatrix} (1 - \alpha)M \\ \alpha S \end{bmatrix} \mathbf{B} = \begin{bmatrix} (1 - \alpha)\mathbf{P} \\ \mathbf{0} \end{bmatrix}. \quad (7.12)$$

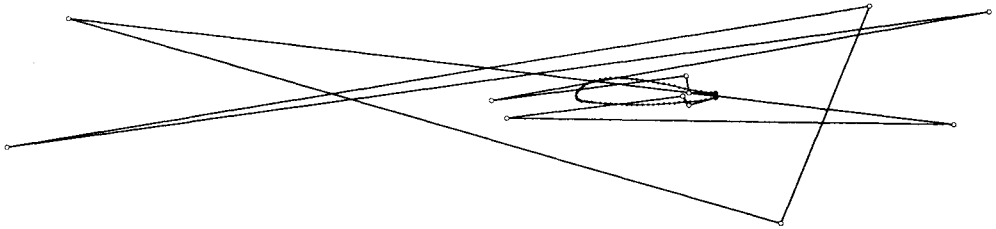


Figure 7.7. Least squares approximation: a degree 13 Bézier curve fitted to the airplane wing data set.

The factor α (restricted to $[0, 1)$) gives control over the influence of the added equations. It is solved in the same way as (7.9), that is, by forming the symmetric linear system of normal equations. The system is solvable because the coefficient matrix of (7.12) still has $n + 1$ linearly independent columns, as inherited from the initial matrix M .

The airplane wing data for Figure 7.8 was created by decimating the data from Figure 7.6. A simple least squares solution (7.9) to this data would produce a polygon that looks worse than that of Figure 7.7. However, by employing shape equations, here with $\alpha = 0.1$, results in the solution shown.

7.2.3. Point and Tangent Data Interpolation

Lagrange interpolation can wiggle unexpectedly, thus in an effort to gain more control, one may specify tangents at the data points. Then the given information consists of points \mathbf{p}_i , associated parameter values t_i , and associated tangent vectors \mathbf{m}_i . Interpolating to this data, a cubic polynomial is constructed between each \mathbf{p}_i and \mathbf{p}_{i+1} . This is called *cubic Hermite interpolation*. Figure 7.9 illustrates the result of cubic Hermite interpolation over several segments. Since adjacent segments share the same tangent vector, a globally C^1 interpolant is the result.

Consider the two points $\mathbf{p}_0, \mathbf{p}_1$, two tangent vectors $\mathbf{m}_0, \mathbf{m}_1$, and parameters t_0 and t_1 . The objective is to find a cubic polynomial curve $\mathbf{p}(t)$ that interpolates to these data:

$$\mathbf{p}(t_0) = \mathbf{p}_0, \quad \dot{\mathbf{p}}(t_0) = \mathbf{m}_0, \quad \dot{\mathbf{p}}(t_1) = \mathbf{m}_1, \quad \mathbf{p}(t_1) = \mathbf{p}_1,$$

where the dot denotes differentiation. The interpolant \mathbf{p} will be written in cubic Bézier form, and therefore it is left to determine the four Bézier points, two of them are quickly determined:

$$\mathbf{b}_0 = \mathbf{p}_0, \quad \mathbf{b}_3 = \mathbf{p}_1.$$

The endpoint derivatives for Bézier curves are

$$\dot{\mathbf{p}}(t_0) = \frac{3}{\Delta t} \Delta \mathbf{b}_0, \quad \dot{\mathbf{p}}(t_1) = \frac{3}{\Delta t} \Delta \mathbf{b}_2,$$

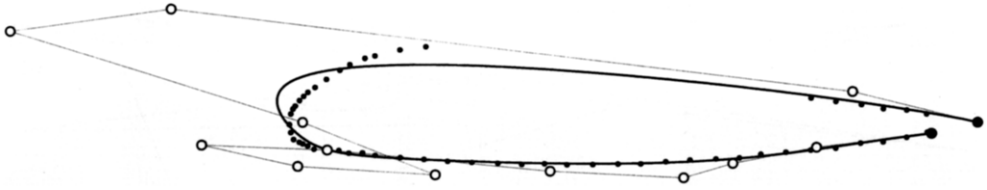


Figure 7.8. Least squares approximation with smoothing: a degree 13 Bézier curve fitted to an incomplete airplane wing data set. For this example, $\alpha = 0.1$.

where $\Delta t = t_1 - t_0$. Thus we can easily solve for \mathbf{b}_1 and \mathbf{b}_2 :

$$\mathbf{b}_1 = \mathbf{p}_0 + \frac{\Delta t}{3}\mathbf{m}_0, \quad \mathbf{b}_2 = \mathbf{p}_1 - \frac{\Delta t}{3}\mathbf{m}_1.$$

Thus the interpolant in Bézier form takes the form

$$\mathbf{p}(t) = \mathbf{p}_0 B_0^3(\hat{t}) + (\mathbf{p}_0 + \frac{\Delta t}{3}\mathbf{m}_0) B_1^3(\hat{t}) + (\mathbf{p}_1 - \frac{\Delta t}{3}\mathbf{m}_1) B_2^3(\hat{t}) + \mathbf{p}_1 B_3^3(\hat{t}) \quad (7.13)$$

for the global parameter $t \in [t_0, t_1]$ and the local parameter $\hat{t} = (t - t_0)/(t_1 - t_0)$, which lies in $[0, 1]$.

The *cardinal form* of the interpolation problem is characterized by the given data appearing *explicitly* in the equation for the interpolant. Simply rearrange (7.13):

$$\mathbf{p}(t) = \mathbf{p}_0 H_0^3(t) + \mathbf{m}_0 H_1^3(t) + \mathbf{m}_1 H_2^3(t) + \mathbf{p}_1 H_3^3(t), \quad (7.14)$$

where

$$\begin{aligned} H_0^3(t) &= B_0^3(\hat{t}) + B_1^3(\hat{t}), \\ H_1^3(t) &= \frac{\Delta t}{3} B_1^3(\hat{t}), \\ H_2^3(t) &= -\frac{\Delta t}{3} B_2^3(\hat{t}), \\ H_3^3(t) &= B_2^3(\hat{t}) + B_3^3(\hat{t}). \end{aligned} \quad (7.15)$$

The H_i^3 are called *cubic Hermite polynomials* and are shown in Figure 7.10. An additional requirement for the H_i^3 to be cardinal functions for the cubic Hermite interpolation problem is the following: They must be *cardinal* with respect to evaluation and differentiation at $t = t_0$ and $t = t_1$, which means that each of the H_i^3 equals 1 for one of these four operations and is zero for the remaining three. Another property to note is that the point coefficients must sum to one if (7.14) is to be geometrically meaningful: $H_0^3(t) + H_3^3(t) \equiv 1$. This is of course also verified by inspection of (7.15).

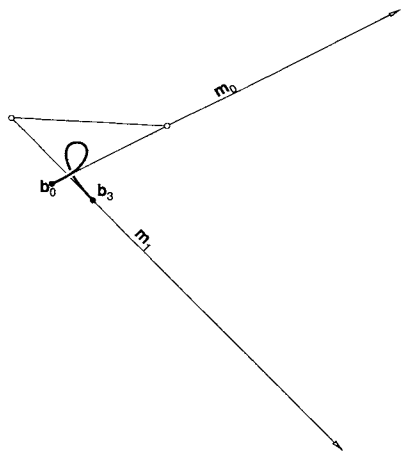


Figure 7.9. Cubic Hermite interpolation: the given point, tangent, and parameter data together with an interpolating cubic in Bézier form.

7.3. C^2 CUBIC SPLINE INTERPOLATION

C^2 cubic spline interpolation is probably the most frequently used application of B-splines. The problem statement is as follows:

Given: a set of data points $\mathbf{p}_0, \dots, \mathbf{p}_K$ and a knot sequence τ_0, \dots, τ_K and a knot multiplicity vector $3, 1, 1, \dots, 1, 1, 3$.

Find: a set of B-spline control points $\mathbf{d}_0, \dots, \mathbf{d}_L$ with $L = K + 2$ such that the resulting C^2 piecewise cubic curve $\mathbf{x}(u)$ satisfies

$$\mathbf{x}(\tau_i) = \mathbf{p}_i; \quad i = 0, \dots, K. \quad (7.16)$$

Consult Figure 7.11 for an example of the numbering scheme. The triple end knots force the curve to interpolate to the first and last data point:

$$\mathbf{d}_0 = \mathbf{p}_0 \quad \text{and} \quad \mathbf{d}_L = \mathbf{p}_K,$$

thus equations for \mathbf{d}_0 and \mathbf{d}_L may be eliminated from the list of unknowns. Even so, the above problem is underdetermined because the number of unknowns is $K + 1$, whereas the number of given data points is $K - 1$. Typically two end conditions are specified in order to have a uniquely solvable problem. To begin with, consider *clamped end conditions*; other end conditions are presented in Section 7.3.1. A clamped end condition corresponds to the prescription of two derivatives $\dot{\mathbf{x}}(\tau_0)$ and $\dot{\mathbf{x}}(\tau_K)$,

$$\dot{\mathbf{x}}(\tau_0) = \frac{3}{\tau_1 - \tau_0}[\mathbf{d}_1 - \mathbf{d}_0], \quad \dot{\mathbf{x}}(\tau_K) = \frac{3}{\tau_K - \tau_{K-1}}[\mathbf{d}_L - \mathbf{d}_{L-1}].$$

The clamped end conditions yield

$$\mathbf{d}_1 = \mathbf{d}_0 + \frac{\tau_1 - \tau_0}{3} \dot{\mathbf{x}}(\tau_0) \quad \text{and} \quad \mathbf{d}_{L-1} = \mathbf{d}_L - \frac{\tau_K - \tau_{K-1}}{3} \dot{\mathbf{x}}(\tau_K),$$

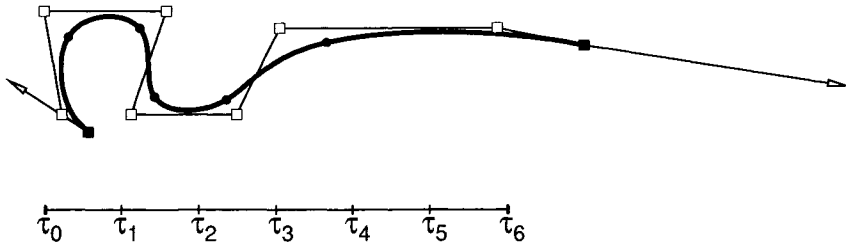


Figure 7.11. Cubic spline interpolation: the case of clamped end conditions with six intervals.

7.3.1. End Conditions

For C^2 cubic spline interpolation, the choice of end conditions is important for the shape of the interpolant near the endpoints. Clamped end conditions, as employed in the previous section, are intended to be used in situations where the end derivatives are actually known. But in most applications, one does not have this knowledge. Still, two extra equations are needed in addition to the basic interpolation conditions (7.16). Below is a list several other end condition methods.

Natural end conditions are derived from the physical analogy of a wooden beam which is clamped at some positions. Beyond the first and last clamps, such a beam assumes the shape of a straight line. A line is characterized by having a zero second derivative, and hence the end conditions

$$\ddot{\mathbf{x}}(\tau_0) = \mathbf{0}, \quad \ddot{\mathbf{x}}(\tau_K) = \mathbf{0}$$

are called “natural” end conditions. In terms of B-spline control vertices (using triple end knots), this becomes

$$\mathbf{d}_0 - 2\mathbf{d}_1 + \frac{\Delta_1}{\Delta_0 + \Delta_1}\mathbf{d}_1 + \frac{\Delta_0}{\Delta_0 + \Delta_1}\mathbf{d}_2 = \mathbf{0},$$

where $\Delta_i = \tau_{i+1} - \tau_i$. Rearrange this equation for the linear system and obtain

$$(\Delta_0 + \Delta_1)\mathbf{d}_0 - (2\Delta_0 + \Delta_1)\mathbf{d}_1 + \Delta_0\mathbf{d}_2 = \mathbf{0}. \tag{7.20}$$

A similar condition holds at the other endpoint. Unless required by a specific application, this end condition should be avoided as it forces the curve to behave linearly near the endpoints.

Bessel end conditions typically yield better results than natural end conditions. They are defined as follows: the first three data points and their parameter values determine an interpolating quadratic curve. Its first derivative at \mathbf{p}_0 is taken to be the one for the spline curve. This results in

$$\mathbf{d}_1 = \frac{2}{3}(\alpha\mathbf{p}_0 + \beta\mathbf{a}) + \frac{1}{3}\mathbf{p}_0$$

where

$$\mathbf{a} = \frac{1}{2\alpha\beta}(\mathbf{p}_1 - \alpha^2\mathbf{p}_0 - \beta^2\mathbf{p}_2), \quad \alpha = \frac{\Delta_1}{\tau_2 - \tau_0} \quad \text{and} \quad \beta = 1 - \alpha.$$

This value for \mathbf{d}_1 is then used in the clamped end condition in the previous section. The control point \mathbf{d}_{L-1} is obtained in complete analogy; it is then also used for a clamped end condition.

Quadratic end conditions require $\ddot{\mathbf{x}}(\tau_0) = \ddot{\mathbf{x}}(\tau_1)$. If all data points and parameter values were read off from a quadratic curve, then this condition would ensure that the spline interpolant reproduces that quadratic. The same is true, of course, for Bessel end conditions.

Not-a-knot end conditions work from the premise that if the first and second cubic segment are parts of one cubic, then their third derivatives at τ_1 would agree. The name is derived from the fact that the knot τ_1 does not act as a breakpoint between two distinct cubic segments.

7.3.2. Defining a Knot Sequence

The spline interpolation problem almost always assumes that parameter values τ_i are *given* along with the data points \mathbf{p}_i . These parameters dictate the amount of time an imaginary particle on the curve spends between \mathbf{p}_i and \mathbf{p}_{i+1} relative to the neighboring curve segments. If the data points are not derived from a time dependent application, then just how to assign parameter values is not entirely intuitive, yet their choice can have a significant influence upon the shape of the resulting interpolant.

The easiest way to determine the τ_i is simply to set $\tau_i = i$. This is called *uniform* or *equidistant* parametrization. This method is too simplistic to cope with most practical situations because it “ignores” the geometry of the data points by spending the same amount of time between any two adjacent data points. Drastic changes in spacing of the data can result in overshooting of the interpolant.

The *chord length* parametrization is a simple method which is a great improvement over the uniform parametrization. The knot spacing is proportional to the distances of the data points:

$$\frac{\Delta_i}{\Delta_{i+1}} = \frac{\|\Delta\mathbf{p}_i\|}{\|\Delta\mathbf{p}_{i+1}\|}, \quad (7.21)$$

where $\Delta_i = \tau_{i+1} - \tau_i$. Equation (7.21) does not uniquely define a knot sequence; rather, it defines a whole family of parametrizations that are related to each other by affine parameter transformations.

The *centripetal* parametrization [20] improves upon the chord length idea. If one sets

$$\frac{\Delta_i}{\Delta_{i+1}} = \left[\frac{\|\Delta\mathbf{p}_i\|}{\|\Delta\mathbf{p}_{i+1}\|} \right]^{1/2}, \quad (7.22)$$

the resulting motion of a point on the curve will “smooth out” variations in the centripetal force acting on it.

The uniform parametrization is the only one that is invariant under affine transformations of the data points. All other methods involve length measurements, and lengths

are not preserved under affine maps. One solution to this dilemma is the introduction of a modified length measure, as described in Nielson [24]. For more literature on parametrizations, see [6,9,15,17,18,21,25]. There is probably no “best” parametrization, since any method can be defeated by a suitably chosen data set.

7.3.3. The Minimum Property

In the early days of design, a smooth curve was manually drawn through a given set of points by placing metal weights, called “ducks,” at the data points, and then passing a thin, elastic wooden beam, a “spline,” between the ducks. The resulting curve is always very smooth and usually aesthetically pleasing. The wooden beam assumes a position that minimizes its bending energy. The mathematical model of the beam is a curve $\mathbf{x}(u)$, and its bending energy E is given by

$$E = c \int_0^l (\kappa(s))^2 ds,$$

where κ denotes the curvature of the curve, ds is the arc length of the beam, l is the length of the beam, and c is a constant determined by the material of the beam.

The integral of the curvature of most curves is difficult to work with, therefore for mathematical simplicity, one often approximates the above integral by a simpler one:

$$\hat{E} = \int [\ddot{\mathbf{x}}(u)]^2 du. \quad (7.23)$$

Note that \hat{E} is a vector; it is obtained by performing the integration on each component of \mathbf{x} . The penalty for mathematical simplicity is accuracy. The curvature of a curve is given by

$$\kappa(u) = \frac{\|\dot{\mathbf{x}} \wedge \ddot{\mathbf{x}}\|}{\|\dot{\mathbf{x}}\|^3}.$$

But it must be that $\|\dot{\mathbf{x}}\| \approx 1$ in order for $\|\ddot{\mathbf{x}}\|$ to be a good approximation to κ . This means, however, that the curve must be parametrized according to arc length. This assumption is not very realistic for cubic splines in a design environment.

While the classical spline curve merely minimizes an approximation to (7.23), methods have been developed that produce interpolants which minimize the true energy, see [22], [5]. Moreton and Séquin have suggested to minimize the functional $\int [\kappa'(t)]^2 dt$ instead, see [23].

7.4. POLYNOMIAL SURFACE METHODS

To a large part, surface methods mirror curve methods. This is apparent in the tensor product methods of this section. The Coons method presented here deviates a bit from this idea, although the flexibility it allows in a design environment makes it an important surface construction method.

7.4.1. Discrete Coons Patches

Coons patches belong to the class of surfacing methods which are capable of *transfinite interpolation*. This means that the input data are curves rather than discrete points. A

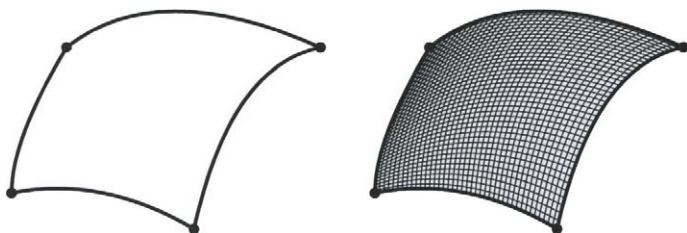


Figure 7.12. Coons patches: an example.

special case of Coons patches, which is geared toward Bézier techniques, is discussed in this section. This construction is called a *discrete Coons patch* (see [11]).

Suppose four curves with a roughly rectangular structure are given, as illustrated in the left of Figure 7.12. These curves are to be the boundary of a surface patch fit between them, as illustrated in the right of the figure. Further, assume that all four curves, with opposite boundary curves of the same degree, are in Bézier form. For $m = n = 3$, the given data takes the following form:

$$\begin{array}{cccc} \mathbf{b}_{00} & \mathbf{b}_{01} & \mathbf{b}_{02} & \mathbf{b}_{03} \\ \mathbf{b}_{10} & & & \mathbf{b}_{13} \\ \mathbf{b}_{20} & & & \mathbf{b}_{23} \\ \mathbf{b}_{30} & \mathbf{b}_{31} & \mathbf{b}_{32} & \mathbf{b}_{33} \end{array}$$

The problem: find the control net of a Bézier surface that fits between the boundary curves.

Figure 7.13 illustrates the construction of the Coons patch. In order to find the interior control points $\mathbf{b}_{i,j}$, first construct a point on each of the following ruled surfaces:

$$\mathbf{b}_{i,j}^u = \left(1 - \frac{i}{m}\right)\mathbf{b}_{0,j} + \frac{i}{m}\mathbf{b}_{m,j} \quad \text{and} \quad \mathbf{b}_{i,j}^v = \left(1 - \frac{j}{n}\right)\mathbf{b}_{i,0} + \frac{j}{n}\mathbf{b}_{i,n}.$$

Next, construct a point on the bilinear interpolant to the four corner points:

$$\mathbf{b}_{i,j}^{u,v} = \begin{bmatrix} 1 - \frac{j}{n} & \frac{j}{n} \end{bmatrix} \begin{bmatrix} \mathbf{b}_{0,0} & \mathbf{b}_{0,m} \\ \mathbf{b}_{n,0} & \mathbf{b}_{m,n} \end{bmatrix} \begin{bmatrix} 1 - \frac{i}{m} \\ \frac{i}{m} \end{bmatrix}.$$

The final control point is created as a combination of these three points:

$$\mathbf{b}_{i,j} = \mathbf{b}_{i,j}^u + \mathbf{b}_{i,j}^v - \mathbf{b}_{i,j}^{u,v}. \quad (7.24)$$

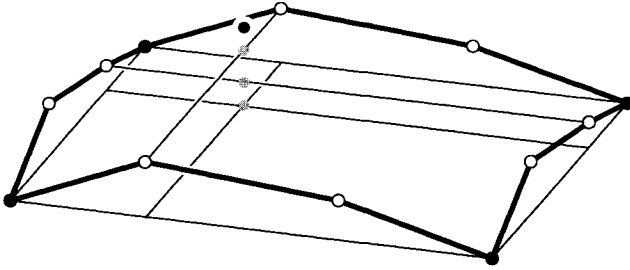


Figure 7.13. Coons patches: the construction. Gray points, from bottom: $\mathbf{b}_{1,2}^{u,v}$, $\mathbf{b}_{1,2}^u$, $\mathbf{b}_{1,2}^v$. Above them, solid black: $\mathbf{b}_{1,2}$.

7.4.2. Tensor Product Interpolation

Tensor product interpolation is suitable for data which has a rectangular structure. More precisely, the given data consist of an $(m + 1) \times (n + 1)$ array of data points \mathbf{x}_{ij} ; $0 \leq i \leq m$, $0 \leq j \leq n$, and each point has an associated parameter value (u_i, v_j) . A tensor product Bézier patch may be written in matrix form:

$$\mathbf{x}(u, v) = \begin{bmatrix} B_0^m(u) & \cdots & B_m^m(u) \end{bmatrix} \begin{bmatrix} \mathbf{b}_{00} & \cdots & \mathbf{b}_{0n} \\ \vdots & & \vdots \\ \mathbf{b}_{m0} & \cdots & \mathbf{b}_{mn} \end{bmatrix} \begin{bmatrix} B_0^n(v) \\ \vdots \\ B_n^n(v) \end{bmatrix}. \quad (7.25)$$

Interpolation requires that (7.25) hold for each pair (u_i, v_j) . This results in $(n + 1) \times (m + 1)$ equations, which may be written concisely as

$$\mathbf{X} = \mathbf{UBV}, \quad (7.26)$$

where

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_{00} & \cdots & \mathbf{x}_{0n} \\ \vdots & & \vdots \\ \mathbf{x}_{m0} & \cdots & \mathbf{x}_{mn} \end{bmatrix},$$

$$\mathbf{U} = \begin{bmatrix} B_0^m(u_0) & \cdots & B_m^m(u_0) \\ \vdots & & \vdots \\ B_0^m(u_m) & \cdots & B_m^m(u_m) \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{c}_{00} & \cdots & \mathbf{c}_{0n} \\ \vdots & & \vdots \\ \mathbf{c}_{m0} & \cdots & \mathbf{c}_{mn} \end{bmatrix},$$

$$V = \begin{bmatrix} B_0^n(v_0) & \cdots & B_0^n(v_n) \\ \vdots & & \vdots \\ B_n^n(v_0) & \cdots & B_n^n(v_n) \end{bmatrix}.$$

The matrices U and V already appeared in Section 7.2.1; they are *Vandermonde matrices*. In an interpolation context, the \mathbf{x}_{ij} are known and the coefficients \mathbf{b}_{ij} are unknown. Theoretically they may be found by setting

$$\mathbf{B} = U^{-1}\mathbf{X}V^{-1}. \quad (7.27)$$

The inverse matrices in (7.27) exist since the functions B_i^m and B_j^n are linearly independent.

The practical method for finding the control points centers on the tensor product property, as discussed in the chapter 4 on Bézier Techniques. Following the schematic diagram in Figure 7.14, the key is to break the problem down into two sets of curve interpolation problems. This is apparent by rewriting (7.26) as

$$\mathbf{X} = \mathbf{D}\mathbf{V}, \quad (7.28)$$

where

$$\mathbf{D} = \mathbf{U}\mathbf{B}. \quad (7.29)$$

Equation (7.28) should be rearranged to follow the normal linear system format, that is $\mathbf{X}^T = \mathbf{V}\mathbf{D}$.

Thus, first solve $(n+1)$ univariate degree m interpolation problems in (7.30), one for each row of \mathbf{X}^T and \mathbf{D} , where \mathbf{D} contains the unknowns. This is illustrated in the middle of the figure by the six cubic interpolants – the "rows." Next, solve $(m+1)$ univariate degree n interpolation problems in (7.29), which results in \mathbf{B} . In the figure, this corresponds to four degree five interpolation problems, schematically represented by the middle and right most diagram. It is important to note that the coefficient matrix is the same for all interpolation problems in the two stages of this algorithm.

7.4.3. Approximation with Tensor Product Patches

Often times the data do not come in a rectangular structure, as expected in the tensor product interpolation of Section 7.4.2. This is particularly true with the advent of laser digitizers. Even if the data points have a rectangular structure, it may be nontrivial to find parameter values such that the univariate curve problems produce reasonable solutions. Approximation allows for a more flexible surface construction method.

Suppose the given data consists of a set of points \mathbf{p}_k , $k = 0, \dots, K$. Also assume that each data point \mathbf{p}_k is associated with a corresponding pair of parameters $\mathbf{u}_k = (u_k, v_k)$. Approximate this data by a degree (m, n) Bézier patch. For "best" results, the number of data points should well exceed the number of points needed for interpolation: $(K+1) \gg (m+1) \times (n+1)$. To aid in the construction of the approximant, the Bézier patch will be written using a *linearized* notation:

$$\mathbf{x}(u, v) = [B_0^m(u)B_0^n(v), \dots, B_m^m(u)B_n^n(v)] \begin{bmatrix} \mathbf{b}_{0,0} \\ \vdots \\ \mathbf{b}_{m,n} \end{bmatrix}. \quad (7.31)$$

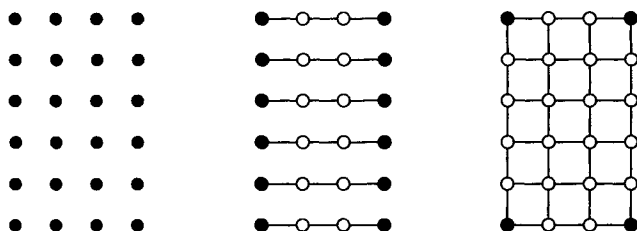


Figure 7.14. Tensor product interpolation: left, the data points; middle, interpolating all rows of data points; right, interpolating all columns from previous step.

The best approximating surface would result in each data point lying on the approximating surface. For the k -th data point \mathbf{p}_k , this becomes $\mathbf{p}_k = \mathbf{x}(\mathbf{u}_k)$ or

$$\mathbf{p}_k = [B_0^m(u_k)B_0^n(v_k), \dots, B_m^m(u_k)B_n^n(v_k)] \begin{bmatrix} \mathbf{b}_{0,0} \\ \vdots \\ \mathbf{b}_{m,n} \end{bmatrix}. \quad (7.32)$$

Combining all $K + 1$ of these equations results in

$$\begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{p}_K \end{bmatrix} = \begin{bmatrix} B_0^m(u_0)B_0^n(v_0) & \dots & B_m^m(u_0)B_n^n(v_0) \\ & \vdots & \\ & \vdots & \\ & \vdots & \\ B_0^m(u_K)B_0^n(v_K) & \dots & B_m^m(u_K)B_n^n(v_K) \end{bmatrix} \begin{bmatrix} \mathbf{b}_{0,0} \\ \vdots \\ \mathbf{b}_{m,n} \end{bmatrix}, \quad (7.33)$$

which may be abbreviated to

$$\mathbf{P} = \mathbf{M}\mathbf{B}. \quad (7.34)$$

These are $K + 1$ equations in $(m + 1)(n + 1)$ unknowns. If there are many more data points than control points, then the linear system (7.34) is *overdetermined*. A “good” approximation is found by forming

$$\mathbf{M}^T\mathbf{P} = \mathbf{M}^T\mathbf{M}\mathbf{B}. \quad (7.35)$$

Notice that this approach mimics that used in Section 7.2.2, therefore this is the *least squares solution* to the approximation problem. A note of caution: if the number of data points is very large (10^5 or more), then the normal equations become ill-conditioned and the least squares problem may become unstable.

Defining Parameter Values

In a practical setting, one would not typically be given the parameter values $\mathbf{u}_k = (u_k, v_k)$. Finding good values for the \mathbf{u}_k is not always an easy problem. Three solutions are described in this section.

If the data points can be projected into a plane then finding good parameters is not difficult. Assume they can be projected into the (x, y) -plane for simplicity. Each \mathbf{p}_k is projected by simply dropping its z -coordinate, leaving a pair (x_k, y_k) . Scale the (x_k, y_k) so that they fit into the desired domain, and then set $u_k = x_k$ and $v_k = y_k$.

If the data can be projected onto a cylinder then finding parameters is also not difficult. For example, assume they can be projected onto the cylinder

$$\mathbf{x}(\theta, z) = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ z \end{bmatrix}.$$

When each \mathbf{p}_k is projected onto the cylinder, the projected point's (θ_k, z_k) coordinate will correspond to the parameters of \mathbf{p}_k . Of course an actual projection is not necessary. Here, the value of θ is determined by a calculation in the (x, y) -plane, and z_k is directly extracted from \mathbf{p}_k . Finally, scale all (θ_k, z_k) to live within the desired domain.

For less structured data, it might be necessary to use a more sophisticated method. First, obtain a *triangulation* of the data points. This scenario is realistic for data obtained using a laser digitizer. Assuming that the triangulation is isomorphic to the unit square, we can construct a triangulation in the unit square with the same connectivity as the given one in 3D. The following method is due to Floater [14]. First, a convex polygon is built in the (u, v) unit square with as many vertices as the 3D mesh has boundary vertices. This polygon is somewhat arbitrary; a circle or the boundary of the unit square are good candidates for forming it. In this way, we assign 2D parameters to the 3D mesh boundary points. Next, consider any interior point \mathbf{u} of the 2D mesh with n neighbors. These neighbors are labeled $\mathbf{u}_1, \dots, \mathbf{u}_n$. For a “nice” triangulation, the following condition should be satisfied for each interior \mathbf{u} :

$$\mathbf{u} = \frac{1}{n} \sum_{i=1}^n \mathbf{u}_i. \tag{7.36}$$

We now observe that there are as many equations (7.36) as there are (unknown) interior points in the mesh. Some of these equations involve boundary points, others will not. This system is always solvable.

Shape Equations

One of the points of using Bézier techniques is the benefit of a polygon that closely resembles the shape of the underlying curve or surface. However, the solution to a least squares problem in Section 7.4.3 may be close to the data points, yet the control net might “behave badly,” similar to the curve case as illustrated in Figure 7.7. As with curve approximation, a way to combat such behavior is to invoke *shape equations*. These are conditions that a “good” control net would satisfy. A translational surface, which is characterized by the fact that its twist vanishes everywhere, has a very well-behaved

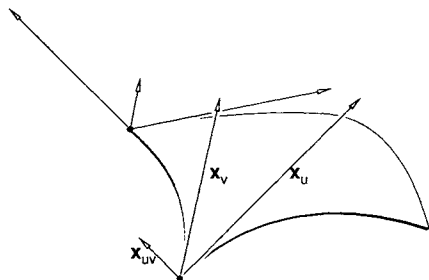


Figure 7.15. Bicubic Hermite patches: some of the data points and vectors.

polygon. In terms of the control net, this means that

$$\Delta^{1,1}\mathbf{b}_{i,j} = \mathbf{0}; \quad i = 0, \dots, m - 1, \quad j = 0, \dots, n - 1.$$

When these equations are added to the overdetermined linear system (7.34), the result will be less faithful to the data points, but it will achieve a control net with better shape. In practice, one would weigh the shape equations, just as was done for the curve case.

7.4.4. Bicubic Hermite Patches

Many surface schemes are generalizations of curve schemes. Bicubic Hermite patches follow that rule, as they are a generalization of cubic Hermite interpolation from Section 7.2.3. Again, to gain more control over the interpolant, derivatives are introduced to the given information. As illustrated in Figure 7.15, the given data for this interpolation problem include points, partials, and mixed partials at each corner:

$$[\mathbf{h}_{i,j}] = \begin{bmatrix} \mathbf{x}(u_0, v_0) & \mathbf{x}_v(u_0, v_0) & \mathbf{x}_v(u_0, v_1) & \mathbf{x}(u_0, v_1) \\ \mathbf{x}_u(u_0, v_0) & \mathbf{x}_{uv}(u_0, v_0) & \mathbf{x}_{uv}(u_0, v_1) & \mathbf{x}_u(u_0, v_1) \\ \mathbf{x}_u(u_1, v_0) & \mathbf{x}_{uv}(u_1, v_0) & \mathbf{x}_{uv}(u_1, v_1) & \mathbf{x}_u(u_1, v_1) \\ \mathbf{x}(u_1, v_0) & \mathbf{x}_v(u_1, v_0) & \mathbf{x}_v(u_1, v_1) & \mathbf{x}(u_1, v_1) \end{bmatrix}. \quad (7.37)$$

Note how the coefficients in the matrix are grouped into four 2×2 partitions, each holding the data pertaining to one corner. Additionally, the parameter space must be given, for instance, define the patch over $u_0 \leq u \leq u_1$ and $v_0 \leq v \leq v_1$.

Employing knowledge of the Hermite basis functions from Section 7.2.3, the interpolating bicubic Hermite patch takes the following form:

$$\mathbf{x}(u, v) = \sum_{i=0}^3 \sum_{j=0}^3 \mathbf{h}_{i,j} H_i^3(u) H_j^3(v); \quad \begin{cases} u_0 \leq u \leq u_1, \\ v_0 \leq v \leq v_1 \end{cases} \quad (7.38)$$

Keep in mind that the H_i^3 must incorporate the parameter interval, as defined in (7.15).

In order to define the bicubic Bézier representation of this patch,

$$\mathbf{x}(u, v) = \sum_{i=0}^3 \sum_{j=0}^3 \mathbf{b}_{i,j} B_i^3(s) B_j^3(t); \quad \begin{cases} u_0 \leq u \leq u_1, & s = \frac{u_0 - u}{u_1 - u_0} \\ v_0 \leq v \leq v_1, & t = \frac{v_0 - v}{v_1 - v_0} \end{cases}$$

the control points must be defined. First, the corner points are simple assignments:

$$\mathbf{b}_{0,0} = \mathbf{x}(u_0, v_0), \quad \mathbf{b}_{30} = \mathbf{x}(u_1, v_0), \quad \mathbf{b}_{0,3} = \mathbf{x}(u_0, v_1), \quad \mathbf{b}_{33} = \mathbf{x}(u_1, v_1).$$

The boundary control points are computed using the curve algorithm from Section 7.2.3, for example

$$\mathbf{b}_{1,0} = \mathbf{x}(u_0, v_0) + \frac{(u_1 - u_0)}{3} \mathbf{x}_u(u_0, v_0), \quad \mathbf{b}_{2,0} = \mathbf{x}(u_1, v_0) - \frac{(u_1 - u_0)}{3} \mathbf{x}_u(u_1, v_0).$$

Recall from the Bézier Techniques chapter 4 that the mixed partials at the corners of a Bézier patch take a very simple form, for example at one corner of the bicubic patch,

$$\mathbf{x}(u_0, v_0) = \frac{9}{\Delta u \Delta v} \Delta^{1,1} \mathbf{b}_{0,0},$$

where $\Delta u = u_1 - u_0$ and $\Delta v = v_1 - v_0$. Therefore, the middle, or *twist* control points are assigned as follows

$$\begin{aligned} \mathbf{b}_{1,1} &= \frac{\Delta u \Delta v}{9} \mathbf{x}_{uv}(u_0, v_0) + \mathbf{b}_{0,1} + \mathbf{b}_{1,0} - \mathbf{b}_{0,0} \\ \mathbf{b}_{2,1} &= -\frac{\Delta u \Delta v}{9} \mathbf{x}_{uv}(u_1, v_0) + \mathbf{b}_{3,1} - \mathbf{b}_{3,0} + \mathbf{b}_{2,0} \\ \mathbf{b}_{1,2} &= -\frac{\Delta u \Delta v}{9} \mathbf{x}_{uv}(u_0, v_1) + \mathbf{b}_{1,3} - \mathbf{b}_{0,3} + \mathbf{b}_{0,2} \\ \mathbf{b}_{2,2} &= \frac{\Delta u \Delta v}{9} \mathbf{x}_{uv}(u_1, v_1) - \mathbf{b}_{3,3} + \mathbf{b}_{2,3} + \mathbf{b}_{3,2}. \end{aligned}$$

Thus the bicubic Bézier patch, which interpolates to Hermite data, has been completely defined.

7.5. C^2 BICUBIC SPLINE INTERPOLATION

In an interpolation context, if given point data come in a rectangular structure, often times there will be too many points to realistically use one polynomial patch, as was done in Section 7.4.2. Higher degree patches have a tendency to oscillate. An alternative approach is to employ bicubic Hermite patches, however generation of the necessary derivative data is not trivially done in a meaningful manner. The most popular solution to this problem are tensor product bicubic B-spline surfaces. The principles from Section 7.4.2 apply here too.

Suppose we are given $(K + 1) \times (L + 1)$ data points \mathbf{x}_{IJ} and two knot sequences u_0, \dots, u_K and v_0, \dots, v_L . This interpolation method will employ a bicubic tensor product B-spline surface,

$$\mathbf{x}(u, v) = \sum_{i=0}^M \sum_{j=0}^N \mathbf{d}_{ij} N_i^3(u) N_j^3(v), \quad (7.39)$$

which has triple knots at each end of the two knot sequences and simple knots elsewhere. This special requirement results in the relationships $M = K + 2$ and $N = L + 2$, that is, the final B-spline control net has two more rows and columns than the given data point

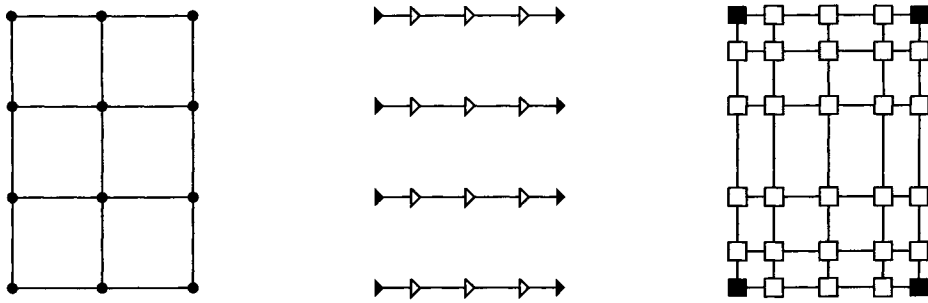


Figure 7.16. Tensor product bicubic spline interpolation: the solution is obtained in a two-step process.

array. The given knot sequences correspond to the unique knots in the knot sequences of the B-spline surface. The solution to this interpolation problem constitutes finding the d_{ij} .

Figure 7.16 illustrates the steps necessary to define the interpolating B-spline surface. For each row of data points, prescribe two end conditions, such as Bessel tangents, and solve the univariate B-spline interpolation problem as described in Section 7.3. Since all interpolation problems use the same knot sequence, each problem has the same tridiagonal coefficient matrix, thus an LU decomposition technique should be applied. The points marked by triangles in Figure 7.16 have thus been constructed. Now take every column of points denoted by triangles, and perform univariate B-spline interpolation on it, again by prescribing end conditions. The resulting control points constitute the desired B-spline control net. An example is shown in Figure 7.17. An alternative approach would be to interpolate first to the columns of data points. This would produce the same result, however the computation count for the two processes are not identical.

7.5.1. Finding Knot Sequences

One obstacle to a good interpolant is the generation of *one* set of parameter values for *all* isoparametric curves in the u -direction; the same holds for the v -direction. When the data points significantly deviate from a regular grid, the problem of finding an appropriate parametrization can be quite difficult. As discussed in Section 7.3.2, a poor choice in parameters can cause an isoparametric curve to unnaturally wiggle, and this defect will be reflected in the surface. One possibility for finding a reasonable parametrization is the following. Create a good parametrization for each isoparametric curve using one of the methods from Section 7.3.2. Average each of these parametrizations to yield one. This approach will only produce acceptable results if all our isoparametric curves essentially yield the same parametrization. It is not difficult to find an example for which this will not help. Figure 7.18 illustrates from a schematic point of view, the type of distribution

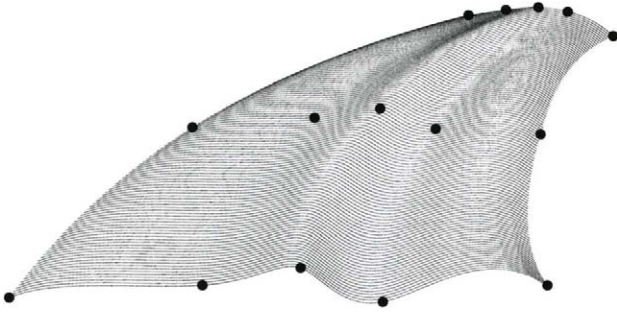


Figure 7.17. Tensor product bicubic spline interpolation: the given data, solid circles, and the solution, using Bessel end conditions and uniform parametrizations.

of data that will cause this method to fail.

7.6. VOLUME DEFORMATIONS

Once an initial curve or surface is designed, sometimes a deformation of the shape is called for; this would be a bending or stretching of the shape. P. Bézier [1–3] devised an intuitive method to deform a Bézier patch which eliminated the need to tediously move control points. His method also applicable to B-spline surfaces. A more graphics-oriented version of this principle was presented by Sederberg and Parry [28], see also [16].

To illustrate the principle, consider the 2D case first. Let $\mathbf{x}(t)$ be a planar curve (Bézier, B-spline, rational B-spline, etc.), which is, without loss of generality, located within the (u, v) unit square. Next, cover the square with a regular grid of points

$$\mathbf{b}_{i,j} = [i/m, j/n]^T, \quad \begin{cases} i = 0, \dots, m; \\ j = 0, \dots, n. \end{cases}$$

Every point (u, v) may be written as

$$(u, v) = \sum_{i=0}^m \sum_{j=0}^n \mathbf{b}_{i,j} B_i^m(u) B_j^n(v).$$

This follows from the linear precision property of Bernstein polynomials. Now, distort the grid of $\mathbf{b}_{i,j}$ into a grid $\hat{\mathbf{b}}_{i,j}$; the point (u, v) will be mapped to a point (\hat{u}, \hat{v}) :

$$(\hat{u}, \hat{v}) = \sum_{i=0}^m \sum_{j=0}^n \hat{\mathbf{b}}_{i,j} B_i^m(u) B_j^n(v), \quad (7.40)$$

which is a mapping of \mathbb{E}^2 to \mathbb{E}^2 . In particular, the control vertices of the curve $\mathbf{x}(t)$ will be mapped to new control vertices, which in turn determine a new curve $\mathbf{y}(t)$. Note that \mathbf{y} is only an approximation to the image of \mathbf{x} under (7.40).³ This is highlighted by

³An exact procedure is described by T. DeRose [8].

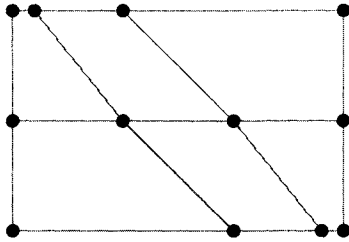


Figure 7.18. Finding parameter values: this 4×3 array of data points are distributed so that it will be very difficult to find a good parametrization in the “horizontal” direction.

the fact that the image of \mathbf{x} 's control polygon under (7.40) would be a collection of curve arcs, not another piecewise linear polygon. Figure 7.19 gives an example of the use of this global design technique. This technique may be generalized. For instance, we may replace the Bézier distortion (7.40) by an analogous tensor product B-spline distortion. This would reintroduce some form of local control into our design scheme.

The next level of generalization is to \mathbb{E}^3 , and requires the introduction of a *trivariate* Bézier patch,

$$(\hat{u}, \hat{v}, \hat{w}) = \sum_{i=0}^m \sum_{j=0}^n \sum_{k=0}^l \hat{\mathbf{b}}_{i,j,k} B_i^m(u) B_j^n(v) B_k^l(w), \tag{7.41}$$

which constitutes a deformation of 3D space. Similar to the planar deformation, the control net in (7.41) is used to deform the control net of a surface embedded in the unit cube. Again, the use of a Bézier patch for the distortion is immaterial; trivariate B-splines, for example could have been used too.

A practical example of volume deformations, as illustrated in Figure 7.20⁴, is in brain imaging. In comparative studies, many MRI brain scans have to be compared. Different people have differently shaped brains; in order to carry out a meaningful comparison, they have to be aligned and then they have to be deformed for a closer match – see [30] or [29]. While volume deformations take 3D objects to other 3D objects, it is convenient to visualize the results by a sequence of 2D slices, as shown in Figure 7.20.

REFERENCES

1. P. Bézier. *Numerical Control: Mathematics and Applications*. Wiley, 1972. translated from the French by A.R. Forrest.
2. P. Bézier. General distortion of an ensemble of biparametric patches. *Computer-Aided Design*, 10(2):116–120, 1978.

⁴Courtesy S. Xie and the Arizona Alzheimer Research Center.

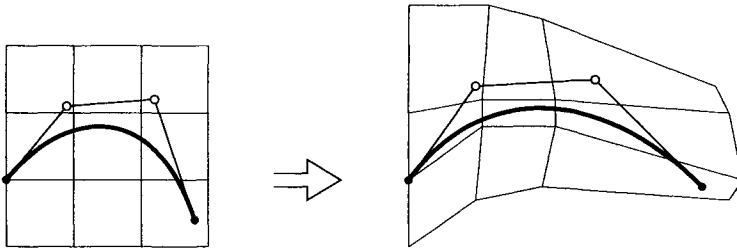


Figure 7.19. Curve deformation: a Bézier polygon is distorted into another polygon, resulting in a deformation of the initial curve.

3. P. Bézier. *The Mathematical Basis of the UNISURF CAD System*. Butterworths, London, 1986.
4. W. Boehm and H. Prautzsch. *Geometric Foundations of Geometric Design*. AK Peters, Boston, 1992.
5. H. Burchardt, J. Ayers, W. Frey, and N. Sapidis. Approximation with aesthetic constraints. In N. Sapidis, editor, *Designing Fair Curves and Surfaces*, pages 3–28. SIAM, Philadelphia, 1994.
6. E. Cohen and C. O'Dell. A data dependent parametrization for spline approximation. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 155–166. Academic Press, 1989.
7. P. Davis. *Interpolation and Approximation*. Dover, New York, 1975. first edition 1963.
8. T. DeRose. Composing Bézier simplices. *ACM Transactions on Graphics*, 7(3):198–221, 1988.
9. M. Epstein. On the influence of parametrization in parametric interpolation. *SIAM J Numer. Analysis*, 13(2):261–268, 1976.
10. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Morgan-Kaufmann, 2001. fifth edition.
11. G. Farin and D. Hansford. Discrete Coons patches. *Computer Aided Geometric Design*, 16(7):691–700, 1999.
12. G. Farin and D. Hansford. *The Essentials of CAGD*. AK Peters, 2000.
13. R. Farouki and V. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, 1987.
14. M. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–271, 1997.
15. T. Foley. Interpolation with interval and point tension controls using cubic weighted ν - splines. *ACM Trans. on Math. Software*, 13(1):68–96, 1987.
16. J. Gomes, L. Darsa, B. Costa, and L. Velho, editors. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann, 1999.

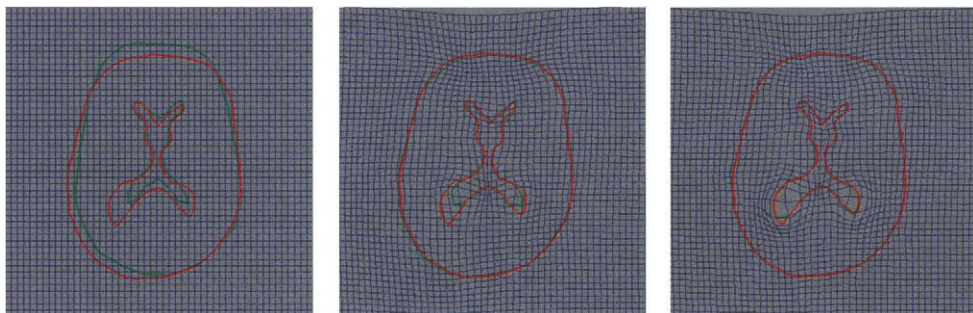


Figure 7.20. Deformation of brain scans: the original two scans, superimposed, are shown on the left. A series of deformations deforms the light colored brain contour to the dark colored one.

17. P. Hartley and C. Judd. Parametrization of Bézier-type B-spline curves. *Computer-Aided Design*, 10(2):130–134, 1978.
18. P. Hartley and C. Judd. Parametrization and shape of B-spline curves. *Computer-Aided Design*, 12(5):235–238, 1980.
19. J. Hoschek and D. Lasser. *Grundlagen der Geometrischen Datenverarbeitung*. B.G. Teubner, Stuttgart, 1989. English translation: *Fundamentals of Computer Aided Geometric Design*, AK Peters, 1993.
20. E. Lee. Choosing nodes in parametric curve interpolation. *Computer-Aided Design*, 21(6), 1989. Presented at the SIAM Applied Geometry meeting, Albany, N.Y., 1987.
21. D. McConalogue. A quasi-intrinsic scheme for passing a smooth curve through a discrete set of points. *The Computer J*, 13:392–396, 1970.
22. E. Mehlum. Nonlinear splines. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 173–208. North-Holland, 1974.
23. H. Moreton and C. Sequin. Minimum variation curves and surfaces for cagd. In N. Sapidis, editor, *Designing Fair Curves and Surfaces*, pages 123–160. SIAM, Philadelphia, 1994.
24. G. Nielson. Coordinate free scattered data interpolation. In L. Schumaker, editor, *Topics in Multivariate Approximation*. Academic Press, 1987.
25. G. Nielson and T. Foley. A survey of applications of an affine invariant norm. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in CAGD*, pages 445–467. Academic Press, 1989.
26. L. Piegl and W. Tiller. *The NURBS Book*. Springer Verlag, 1997. second edition.
27. C. Runge. Ueber empirische Funktionen und die Interpolation zwischen aequidistanten Ordinaten. *ZAMM*, 46:224–243, 1901.
28. T. Sederberg and S. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4):151–160, 1986. SIGGRAPH proceedings.
29. A. Toga, editor. *Brain Warping*. Academic Press, 1999.
30. Z. Xie and G. Farin. Hierarchical B-spline deformation with an application to brain

imaging. In L. Schumaker and T. Lyche, editors, *Mathematical Methods in CAGD*. Vanderbilt University Press, 2001.

Chapter 8

Geometric Continuity

Jörg Peters

This chapter covers geometric continuity with emphasis on a constructive definition for piecewise parametrized surfaces. Section 8.1 gives examples that show the need for a notion of continuity different from matching of Taylor expansions as in the case of functions. Section 8.2 defines geometric continuity for parametric curves, and for surfaces, first along edges, then around points, and finally for a whole complex of patches which is called a G^k free-form surface spline. Here G^k characterizes a relation between specific maps while C^k continuity is a property of the resulting surface. The composition constraint on reparametrizations and the vertex-enclosure constraints are highlighted. Section 8.3 covers alternative definitions and approaches to generating free-form surface splines, and briefly discusses geometric continuity in the context of implicit representations and of generalized subdivision. Section 8.4 explains the generic construction of G^k free-form surface splines and points to some low degree constructions. The chapter closes with pointers to additional literature.

8.1. MOTIVATING EXAMPLES

This section points out the difference between geometric continuity for curves and surfaces and the continuity of functions. The examples are formulated in Bézier representation (Chapter 4 on Bézier Techniques).

Two C^k function pieces join smoothly at a boundary to form a joint C^k function if, at all common points, their κ th derivatives agree for $\kappa = 0, 1, \dots, k$. Since the x , y and z components of curves and surfaces are functions, it is tempting to declare that curve or surface pieces join smoothly if and only if the derivatives of the component functions agree. However, as the following four examples illustrate, this criterion is neither sufficient nor necessary for characterizing smooth curves or smooth surfaces and therefore motivates the definitions in Section 8.2.

The first two examples illustrate the inadequacy of the standard notion of smoothness for functions when applied to *curves*. In Figure 8.1 the V of VC is parametrized by the

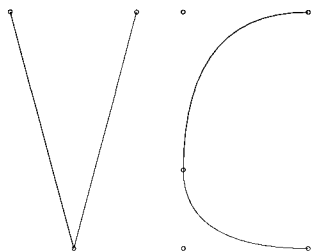


Figure 8.1. Matching derivatives of the component functions and geometric (visual) continuity are not the same: the V of VC is parametrized by two parabolic arcs with equal derivatives at the tip, but the V shape is not geometrically continuous; the C of VC is parametrized by two parabolic arcs with unequal derivatives at their common point, but the C shape is geometrically continuous.

two quadratic pieces, $u, v \in [0, 1]$,

$$\mathbf{q}_1(u) = \begin{bmatrix} 0 \\ -1 \end{bmatrix} (1-u)^2 + \begin{bmatrix} 0 \\ 0 \end{bmatrix} 2(1-u)u + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u^2$$

and

$$\mathbf{q}_2(v) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} (1-v)^2 + \begin{bmatrix} 0 \\ 0 \end{bmatrix} 2(1-v)v + \begin{bmatrix} 1 \\ 1 \end{bmatrix} v^2.$$

Evidently, at the common point $\mathbf{q}_1(1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \mathbf{q}_2(0)$ the derivatives agree:

$$(D\mathbf{q}_1)(1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = (D\mathbf{q}_2)(0).$$

However, even with suitably cushioned end points, the V should not be handed over to boys or girls under the age of 1 for fear of injury from the sharp corner. Matching derivatives clearly do not always imply smoothness. Conversely, smoothness does not imply matching derivatives. The C of VC is parametrized by the two quadratic pieces, $u, v \in [0, 1]$,

$$\mathbf{q}_3(u) = \begin{bmatrix} 3 \\ 2 \end{bmatrix} (1-u)^2 + \begin{bmatrix} 0 \\ 2 \end{bmatrix} 2(1-u)u + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u^2$$

and

$$\mathbf{q}_4(v) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} (1-v)^2 + \begin{bmatrix} 0 \\ -1 \end{bmatrix} 2(1-v)v + \begin{bmatrix} 3 \\ -1 \end{bmatrix} v^2.$$

The C is visually (and geometrically) smooth at the common point $\mathbf{q}_3(1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ since the two pieces have the same vertical tangent line but the derivatives do not agree:

$$(D\mathbf{q}_3)(1) = \begin{bmatrix} 0 \\ -4 \end{bmatrix} \neq \begin{bmatrix} 0 \\ -2 \end{bmatrix} = (D\mathbf{q}_4)(0).$$

Both examples could be made consistent with our notion of continuity for functions if we ruled out parametrizations with zero derivative and substituted $v \rightarrow 2v$ in \mathbf{q}_4 . In the case of *surfaces*, the distinction between higher-order continuity of the component functions and actual (geometric) continuity of the surface is more subtle.

In two variables, we contrast the smoothness criteria for surfaces with the concept valid for functions by looking at two examples involving polynomial pieces in total degree Bézier form, i.e. de Casteljau’s triangles (Chapter 4 on Bézier Techniques). A necessary and sufficient geometric criterion for two polynomial pieces $p_1, p_2 : \mathbb{R}^2 \mapsto \mathbb{R}$ to join C^1 along a common boundary, is the ‘coplanarity condition’ (Chapter 4 on Bézier Techniques), illustrated in Figure 8.3, *left*; the function pieces p_1 and p_2 join C^1 if all subtriangles of the control net are coplanar that share two boundary points. Since the coplanarity of the edge-adjacent triangles of the control net is a geometric criterion it is tempting to use it as a definition of smoothness for surfaces consisting of the 3-sided patches. However, the criterion is neither sufficient nor necessary.

To see that coplanarity of the edge-adjacent triangles of the control net does not imply tangent continuity of the surface consider the eight, degree 2, triangular, polynomial patches (Figure 8.2) whose control nets are obtained by chopping off the eight corners of a cube down to the midpoint of each edge. The edge midpoints and face centers of the cube serve as the control points of 8 quadratic 3-sided Bézier patches. For example, the patch in the positive octant (with thick control lines in Figure 8.2, *left*) has the coefficients

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

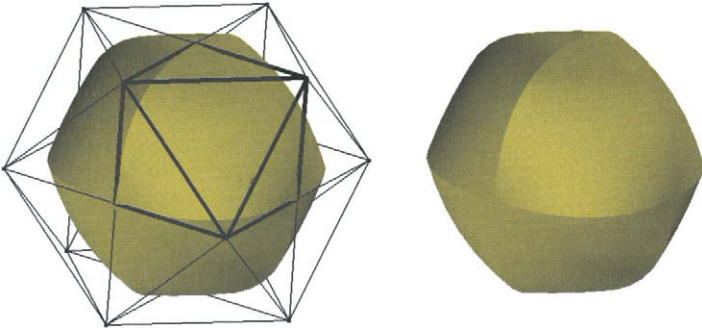


Figure 8.2. (*left*) The 6-point control net of one degree 2 patch in Bézier form is drawn in *thick lines*. The two subtriangles in the control net that include the end points of a boundary of the patch define the derivative along that boundary. For two edge-adjacent patches these subtriangles are mirror images and coplanar with their counterparts in the other patch. Still the surface defined by the patches is not tangent continuous as the creases in the surface demonstrate. (The creases are visible in the silhouette and in the change in surface shading, *right*).

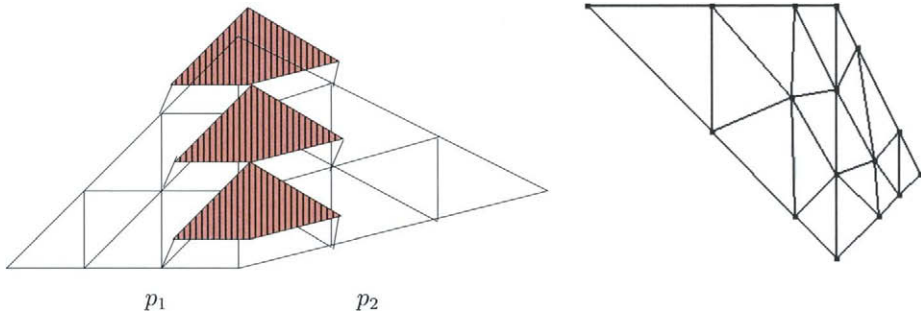


Figure 8.3. (left) Two polynomial pieces p_1 and p_2 join to form a C^1 function if all subtriangles of the control net that share two boundary points (striped) are coplanar (Farin's C^1 condition). (right) Even though the middle cross-boundary subtriangle pair (where the patch labels \mathbf{p} and \mathbf{q} are placed, right) are not coplanar the two Bézier patches $\mathbf{p}(\Delta)$ and $\mathbf{q}(\Delta)$ join to form a tangent continuous surface.

Figure 8.2, right shows that the patches join with a sharp crease at the middle of their common parabolic boundaries. Indeed, the normal at the midpoint $\begin{bmatrix} .75 \\ .75 \\ 0 \end{bmatrix}$ of the equatorial boundary of the positive octant patch is $\begin{bmatrix} 2/3 \\ 2/3 \\ 1/3 \end{bmatrix}$, but to match its counterpart in the lower hemisphere, by symmetry, the z -component would have to be zero. Upper and lower hemisphere therefore do not meet with a continuous normal.

Conversely, the geometric coplanarity criterion is not necessary for a smooth join. The two cubic pieces \mathbf{p}, \mathbf{q} with coefficients (c.f. Figure 8.3)

$$\mathbf{p} : \begin{bmatrix} 72 \\ 72 \\ 6 \\ 36 \\ 36 \\ 0 \\ 12 \\ 12 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 72 \\ 36 \\ 12 \\ 46 \\ 13 \\ 0 \\ 24 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 72 \\ 12 \\ 12 \\ 48 \\ 9 \\ 12 \end{bmatrix} \quad \begin{bmatrix} 72 \\ 0 \\ 12 \end{bmatrix}$$

$$\mathbf{q} : \begin{bmatrix} 0 \\ 0 \\ 0 \\ 12 \\ -12 \\ 0 \\ 18 \\ -18 \\ 0 \\ 24 \\ -24 \\ 6 \end{bmatrix} \quad \begin{bmatrix} 24 \\ 0 \\ 0 \\ -11 \\ 12 \\ 36 \\ -18 \\ 12 \end{bmatrix} \quad \begin{bmatrix} 48 \\ 0 \\ 12 \\ 60 \\ -6 \\ 12 \end{bmatrix} \quad \begin{bmatrix} 72 \\ 0 \\ 12 \end{bmatrix}$$

have the partial derivatives $D_1\mathbf{p} = D_2\mathbf{q}$ along and $D_2\mathbf{p}$, respectively $D_1\mathbf{q}$ across the

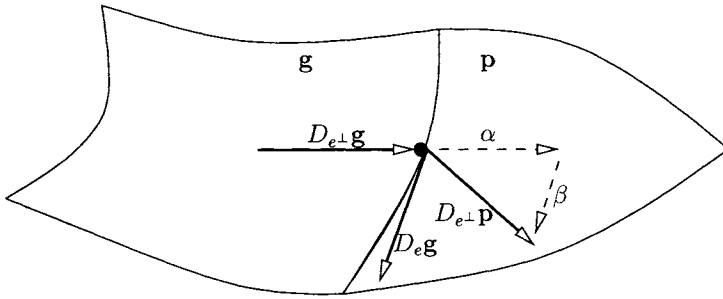


Figure 8.4. If the patches meet with tangent continuity, the transversal derivative $D_{e^\perp}\mathbf{p}$ of \mathbf{p} must be a linear combination of the versal derivative vector $D_e\mathbf{g}$ in the direction e along the preimage E of common boundary $\mathbf{p}(E)$ and the transversal derivative $D_{e^\perp}\mathbf{g}$ in the direction e^\perp perpendicular to e : $D_{e^\perp}\mathbf{p} = \alpha D_e\mathbf{g} + \beta D_{e^\perp}\mathbf{g}$.

common boundary:

$$\begin{aligned} (D_1\mathbf{p})(t, 0) &= \begin{bmatrix} 72 \\ 0 \\ 0 \end{bmatrix} (1-t)^2 + \begin{bmatrix} 72 \\ 0 \\ 36 \end{bmatrix} 2(1-t)t + \begin{bmatrix} 72 \\ 0 \\ 0 \end{bmatrix} t^2 = (D_2\mathbf{q})(0, t), \\ (D_2\mathbf{p})(t, 0) &= \begin{bmatrix} 36 \\ 36 \\ 0 \end{bmatrix} (1-t)^2 + \begin{bmatrix} 66 \\ 39 \\ 0 \end{bmatrix} 2(1-t)t + \begin{bmatrix} 72 \\ 36 \\ 0 \end{bmatrix} t^2, \\ (D_1\mathbf{q})(0, t) &= \begin{bmatrix} 36 \\ -36 \\ 0 \end{bmatrix} (1-t)^2 + \begin{bmatrix} 12 \\ -33 \\ 36 \end{bmatrix} 2(1-t)t + \begin{bmatrix} 36 \\ -18 \\ 0 \end{bmatrix} t^2. \end{aligned}$$

With the help of Maple, we can check that the partial derivatives are coplanar at every point of the boundary, i.e. $\det(D_1\mathbf{p}(t, 0), D_2\mathbf{p}(t, 0), D_1\mathbf{q}(0, t)) = 0$, the zero polynomial in t . Since the surface pieces neither form a cusp nor have vanishing derivatives along the boundary, the normal direction varies continuously across (cf. Lemma 8.3.1, page 212). On the other hand, for the control point differences of the middle pair of subtriangles, labeled p and q in Figure 8.3,

$$\det\left(\begin{bmatrix} 72 \\ 0 \\ 36 \end{bmatrix}, \begin{bmatrix} 66 \\ 39 \\ 0 \end{bmatrix}, \begin{bmatrix} 12 \\ -33 \\ 36 \end{bmatrix}\right) = 5832 \neq 0.$$

This shows that, in contrast to a C^1 match between two functions, edge-adjacent subtriangle pairs need not each be coplanar to obtain a tangent continuous surface.

8.1.1. Differentiation and evaluation

Even though derivatives of the component functions by themselves do not yield a correct picture of curve and surface continuity, the definition of geometric continuity relies on derivatives. And since we work with functions in several variables, some clarification of notation is in order.

First, it is at times clearer to denote evaluation at a point Q by $f|_Q$ rather than $f(Q)$, evaluation on points along a curve segment E by $f|_E$ and to use the symbol \circ for composition, i.e. $\mathbf{g} \circ \mathbf{r} = \mathbf{g}(\mathbf{r})$. We use bold font for vector-valued functions and, somewhat inconsistently but ink-saving, regular font for directions of differentiation e and points of

evaluation, say Q or 0 , the zero vector in \mathbb{R}^n . The notation D^κ for the κ th derivative in one variable is consistent with the notation in two variables from [106]:

Definition 8.1.1 (Differentiation) *The differentials $D^\kappa \mathbf{p}$ of a map $\mathbf{p} : \mathbb{R}^2 \mapsto \mathbb{R}^3$ with x -, y - and z -components $\mathbf{p}^{[x]}$, $\mathbf{p}^{[y]}$, $\mathbf{p}^{[z]}$ and the domain spanned by the unit vectors $\mathbf{e}_1 \perp \mathbf{e}_2$ are defined recursively as*

$$D_{\mathbf{e}_i} \mathbf{p}^{[x]}|_Q := \lim_{t \downarrow 0} \frac{\mathbf{p}^{[x]}(Q + t\mathbf{e}_i) - \mathbf{p}^{[x]}(Q)}{t}, \quad D_{\mathbf{e}_1} \mathbf{p} := \begin{bmatrix} D_{\mathbf{e}_1} \mathbf{p}^{[x]} \\ D_{\mathbf{e}_1} \mathbf{p}^{[y]} \\ D_{\mathbf{e}_1} \mathbf{p}^{[z]} \end{bmatrix}, \quad D\mathbf{p} := [D_{\mathbf{e}_1} \mathbf{p} \ D_{\mathbf{e}_2} \mathbf{p}],$$

$$D^\kappa := DD^{\kappa-1}, \quad \text{e.g. } D^2 \mathbf{p} = DD\mathbf{p} = \begin{bmatrix} D_{\mathbf{e}_1} D_{\mathbf{e}_1} \mathbf{p} & D_{\mathbf{e}_2} D_{\mathbf{e}_1} \mathbf{p} \\ D_{\mathbf{e}_1} D_{\mathbf{e}_2} \mathbf{p} & D_{\mathbf{e}_2} D_{\mathbf{e}_2} \mathbf{p} \end{bmatrix}.$$

If the Jacobian $D\mathbf{p}$ is of full rank 2, \mathbf{p} is called regular. We often abbreviate $D_i \mathbf{p} = D_{\mathbf{e}_i} \mathbf{p}$.

In one variable (see e.g. [17])

$$D^\kappa (\mathbf{g} \circ \rho) = \sum_{j=1}^{\kappa} \sum_{K(j)} c_{K(j)} \left((D^j \mathbf{g}) \circ \rho \right) \cdot (D^1 \rho)^{k_1} \cdot \dots \cdot (D^\kappa \rho)^{k_\kappa}.$$

This combination of the chain rule and the product rule is called *Faá di Bruno's Law* and the bookkeeping is hidden in the index set

$$K(j) := \{k_i \geq 0, i = 1, \dots, \kappa, \sum_{i=1}^{\kappa} k_i = j, \sum_{i=1}^{\kappa} i k_i = \kappa\}, \quad c_{K(j)} := \frac{\kappa!}{k_1! (1!)^{k_1} \dots k_\kappa! (\kappa!)^{k_\kappa}},$$

In two variables $D^\kappa \mathbf{g}$ (no subscript) is a κ -linear map acting on $\mathbb{R}^{2 \times \kappa}$ (κ terms). Its component with index $(i_1, i_2, \dots, i_\kappa) \in \{1, 2\}^\kappa$ is $D_{i_1} D_{i_2} \dots D_{i_\kappa} \mathbf{g}$. The arguments of $D^\kappa \mathbf{g}$ are surrounded by $\langle \cdot \rangle$ and $\langle \mathbf{a}, \mathbf{a}, \dots, \mathbf{a}, \mathbf{b}, \dots, \mathbf{b} \rangle$ with $\mathbf{a} \in \mathbb{R}^2$ repeated i times and $\mathbf{b} \in \mathbb{R}^2$ repeated j times is abbreviated as $\langle (\mathbf{a})^i, (\mathbf{b})^j \rangle$. We can then write the bivariate *Faá di Bruno's Law* as

$$D_1^\kappa (\mathbf{g} \circ \mathbf{r}) = \sum_{j=1}^{\kappa} \sum_{K(j)} c_{K(j)} \left((D^j \mathbf{g}) \circ \mathbf{r} \right) \langle (D_1^1 \mathbf{r})^{k_1}, \dots, (D_1^{\kappa} \mathbf{r})^{k_\kappa} \rangle.$$

For example

$$\begin{aligned} D^2 f(\mathbf{a}, \mathbf{b}) &= [\mathbf{a}^{[1]} \ \mathbf{a}^{[2]}] \begin{bmatrix} D_1^2 f & D_1 D_2 f \\ D_1 D_2 f & D_2^2 f \end{bmatrix} \begin{bmatrix} \mathbf{b}^{[1]} \\ \mathbf{b}^{[2]} \end{bmatrix} \\ &= \mathbf{a}^{[1]} \mathbf{b}^{[1]} D_1^2 f + (\mathbf{a}^{[1]} \mathbf{b}^{[2]} + \mathbf{a}^{[2]} \mathbf{b}^{[1]}) D_1 D_2 f + \mathbf{a}^{[2]} \mathbf{b}^{[2]} D_2^2 f. \end{aligned}$$

8.2. GEOMETRIC CONTINUITY OF PARAMETRIC CURVES AND SURFACES

This section defines k th order geometric continuity, short G^k continuity, as agreement of derivatives after suitable reparametrization, i.e. paraphrasing [58], 'geometric continuity is a relaxation of parametrization, and not a relaxation of smoothness'. Section 3 will show that G^1 and G^2 are equivalent notions to continuity of the tangent and curvature(s).

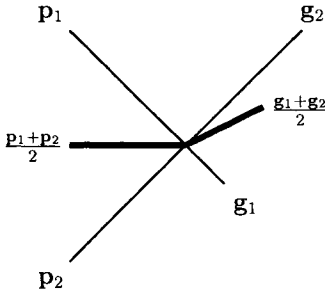


Figure 8.5. The average (bold lines) of two curves, whose pieces \mathbf{p}_i and \mathbf{g}_i join G^∞ , can be tangent discontinuous, i.e. its pieces do not even join G^1 .

8.2.1. Joining parametric curve pieces

Definition 8.2.1 (G^k join) Two C^k curve segments $\mathbf{q} : [a..b] \rightarrow \mathbb{R}$ and $\mathbf{p} : [0..c] \rightarrow \mathbb{R}$ join at $\mathbf{p}(0)$ with geometric continuity G^k via the C^k map $\rho : \mathbb{R} \rightarrow \mathbb{R}$, $\rho(0) = b$, if

$$D^\kappa(\mathbf{g} \circ \rho) |_0 = D^\kappa \mathbf{p} |_0 \quad \kappa = 0, \dots, k, \quad D\rho |_0 > 0, D\mathbf{p} |_0 \neq 0.$$

The map ρ is called reparametrization. If ρ is a rigid transformation then \mathbf{p} and \mathbf{g} are said to join parametrically C^k and if $\rho = \text{id}$, the identity map, then \mathbf{p} and \mathbf{g} form a C^k map.

The constraint $D\rho |_0 > 0$ rules out cusps and other singularities.

With the abbreviation $\mathbf{j}^k \mathbf{p} |_0 = [\mathbf{p} |_0, D\mathbf{p} |_0, \dots, D^k \mathbf{p} |_0]^T \in \mathbb{R}^{(k+1) \times n}$ for $\mathbf{p} \in \mathbb{R}^n$, Faa di Bruno’s law applied to $\mathbf{j}^k \mathbf{p} |_0 = \mathbf{j}^k(\mathbf{g} \circ \rho) |_0$ yields

$$\mathbf{j}^k \mathbf{p} |_0 = \mathbf{A}(\mathbf{j}^k \mathbf{g}) |_{\rho(0)}, \quad \mathbf{A} = \begin{bmatrix} 1 & & & & & \\ & D\rho & & & & \\ & D^2\rho & (D\rho)^2 & & & \\ & D^3\rho & \alpha & (D\rho)^3 & & \\ & \vdots & \vdots & \vdots & \ddots & \\ & D^k\rho & \dots & \dots & \dots & (D\rho)^k \end{bmatrix} |_0, \quad \alpha = 3D\rho D^2\rho.$$

The matrix \mathbf{A} of derivatives of ρ is called G^k connection matrix [13], [113] or β -matrix [7] and $\mathbf{j}^k \mathbf{p}$ is the k -jet of \mathbf{p} . In one variable, two regular maps \mathbf{p} and \mathbf{q} can both be reparametrized so that $\mathbf{p}(\rho_p)$ and $\mathbf{q}(\rho_q)$ have the preferred arclength parametrization (Chapter 2 on Geometric Fundamentals), i.e. unit increments in the parameter correspond to unit increments in the length of the curve. Then $\mathbf{j}^k(\mathbf{p} \circ \rho_p) |_0 = \mathbf{j}^k(\mathbf{q} \circ \rho_q) |_0$.

G^k splines with different connection matrices do not form a linear vector space; in particular the average of two curves that join G^k is not necessarily G^k as illustrated in Figure 8.5: if \mathbf{p}_1 and \mathbf{q}_1 join G^k via ρ_1 at $\mathbf{p}_1(0)$ and \mathbf{p}_2 and \mathbf{q}_2 join G^k via ρ_2 at $\mathbf{p}_2(0) = \mathbf{p}_1(0)$ then, in general, there does not exist a reparametrization ρ so that $(1 - \sigma)\mathbf{p}_1 + \sigma\mathbf{p}_2$ joins G^k with $(1 - \sigma)\mathbf{g}_1 + \sigma\mathbf{g}_2$ at $\mathbf{p}_2(0) = \mathbf{p}_1(0)$. That is, there does not generally exist a connection matrix \mathbf{A} such that

$$\mathbf{A}_1(1 - \sigma)\mathbf{j}^k \mathbf{g}_1 + \mathbf{A}_2\sigma\mathbf{j}^k \mathbf{g}_2 = \mathbf{A}((1 - \sigma)\mathbf{j}^k \mathbf{g}_1 + \sigma\mathbf{j}^k \mathbf{g}_2).$$

In the example shown in Figure 8.5,

$$\mathbf{j}^1 \mathbf{p}_1 = \begin{bmatrix} 0 & -1 \\ 0 & -1 \end{bmatrix}, A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}, \mathbf{j}^1 \mathbf{g}_1 = \begin{bmatrix} 0 & -1/3 \\ 0 & -1/3 \end{bmatrix}, \quad \mathbf{j}^1 \mathbf{g}_2 = \mathbf{j}^1 \mathbf{p}_2 = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$$

but $\mathbf{j}^1(\mathbf{p}_1 + \mathbf{p}_2)/2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ while $\mathbf{j}^1(\mathbf{g}_1 + \mathbf{g}_2)/2 = \begin{bmatrix} 0 & 2/3 \\ 0 & 1/3 \end{bmatrix}$ and there does not exist a G^1 connection matrix $A = \begin{bmatrix} 1 & 0 \\ 0 & D_\rho \end{bmatrix}$ such that $\mathbf{j}^1(\mathbf{p}_1 + \mathbf{p}_2)/2 = A\mathbf{j}^1(\mathbf{g}_1 + \mathbf{g}_2)/2$.

However, if we fix a $(\kappa_i + 1) \times (\kappa_i + 1)$ connection matrix at the i th breakpoint, we can construct a space of degree k splines with prescribed G^{κ_i} joints and knots of order $k - \kappa_i$. Such a spline space can be analyzed as the affine image of a ‘universal spline’ whose control points are in general position [113].

Conversely, any given polygon can be interpreted as the control polygon of a G^k spline: by iterated linear interpolation (corner cutting), the polygon is refined into one whose vertices, when interpreted as Bézier coefficients, define curve pieces that join G^k , e.g. [11] for $k = 2$, [38] for Frénet frame continuity (see Section 8.3.1) and [113], [114], [115] for the general case.

There are *degree-optimal constructions* for this conversion, i.e. constructions that maximise the smoothness of the spline for a given number of corner cuts, i.e. polynomial degree. Via the notion of order of contact (see Section 8.3.1) smoothness is closely related to the ability to interpolate, say the data of a previous spline segment. Following the pioneering paper [20] where it was observed that a cubic segment can often interpolate position, tangent and curvature at both end points (see also [64],[22]), Koch and Höllig [61] conjectured that, under suitable assumptions, “splines of degree $\leq n$ can interpolate points on a smooth curve in \mathbb{R}^m with order of contact $k - 1 = n - 1 + \lfloor (n - 1)/(m - 1) \rfloor$ at every n^{th} knot. Moreover, this geometric Hermite interpolant has the optimal approximation order $k + 1$ ” (see also [101]).

8.2.2. Geometric continuity of edge-adjacent patches

We now turn to a constructive characterization of the smoothness of *surfaces* assembled from standard pieces used in CAGD, such as 3- or 4-sided Bézier patches, or tensor-product b-spline patches.

Definition 8.2.2 (Domain, reparametrization, geometry map, patch)

- A domain is a simple, closed subset Δ of \mathbb{R}^2 , bounded by a finite number of possibly curved edges E_j . Edges are not collinear where they meet.
- Let E_1 be an edge of the domain Δ_1 and E_2 an edge of the domain Δ_2 . Denote an open neighborhood of a set X by $\mathcal{N}(X)$. Then $\mathbf{r} : \mathcal{N}(E_1) \rightarrow \mathcal{N}(E_2)$ is a C^k reparametrization between Δ_1 and Δ_2 if it (1) maps E_1 to E_2 (2) maps points exterior to Δ_1 to points interior of Δ_2 , and (3) is C^k continuous and invertible.
- A C^k geometry map is a map $\mathbf{g} : \Delta \rightarrow \mathbb{R}^3$ such that $D^k \mathbf{g}, \kappa = 0, \dots, k$ is continuous and $\det(D\mathbf{g}) \neq 0$; $\mathbf{g}(\Delta) \subset \mathbb{R}^3$ is called a C^k patch.

Regularity of the geometry map $\det(D\mathbf{g}) \neq 0$, rules out geometric singularities, such as cusps or ridges, and avoids special cases – but it off-hand also rules out singular maps

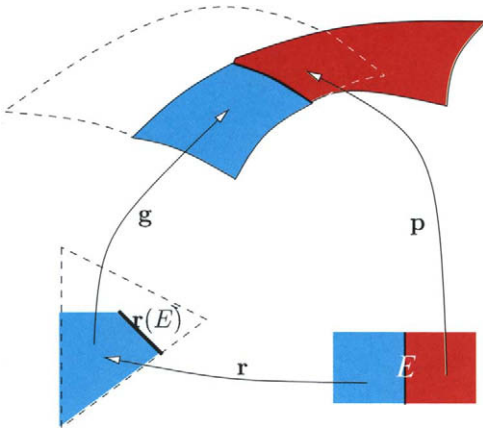


Figure 8.6. Reparametrization $\mathbf{r} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ and geometry maps $\mathbf{p}, \mathbf{g} : \mathbb{R}^2 \rightarrow \mathbb{R}^3$. For a G^k join via \mathbf{r} , the traversal derivatives $D_{e^\perp}^\kappa(\mathbf{g} \circ \mathbf{r})$ and $D_{e^\perp}^\kappa \mathbf{p}$ have to agree along the common boundary $\mathbf{p}(E)$ for $\kappa = 0, \dots, k$. The dashed lines indicate that $\mathbf{r}(E)$ need not be a boundary edge of the standard domain of \mathbf{g} .

that generate perfectly smooth surfaces ([84], [79], [15], [102]). These constructions are shown to be smooth by a local change of variable that removes the singularity. Defining the domain boundary to consist of a few edges is specific to CAGD usage: we could have a fractal boundary separating two pieces of the same smooth surface.

The map \mathbf{g} is called geometry map to emphasize that the local shape (but not the extent) of the surface is defined by \mathbf{g} . The image in \mathbb{R}^3 of \mathbf{g} restricted to its domain is the patch. The reparametrization \mathbf{r} maps outside points to inside points to prevent the surface from folding back onto itself in a 180°-turn.

Next we join two pieces (c.f. Figure 8.6).

Definition 8.2.3 (G^k join) Two C^k geometry maps \mathbf{p} and \mathbf{g} join along $\mathbf{p}(E)$ with geometric continuity G^k via the C^k reparametrization \mathbf{r} if

$$D^\kappa \mathbf{p} |_E = D^\kappa(\mathbf{g} \circ \mathbf{r}) |_E, \quad \kappa = 0, \dots, k.$$

If \mathbf{r} is a rigid transformation then \mathbf{p} and \mathbf{g} are said to join parametrically C^k and if $\mathbf{r} = \text{id}$, the identity map, then the restrictions of \mathbf{p} and \mathbf{g} to their abutting domains form a C^k map.

Since \mathbf{p} , \mathbf{g} and \mathbf{r} are C^k maps, G^k continuity along $\mathbf{p}(E)$ with C^k reparametrization \mathbf{r} is equivalent to just $k + 1$ univariate polynomial equalities corresponding to differentiation in the direction e^\perp perpendicular to the edge E :

$$D_{e^\perp}^\kappa \mathbf{p} |_E = D_{e^\perp}^\kappa(\mathbf{g} \circ \mathbf{r}) |_E, \quad \kappa = 0, \dots, k.$$

That is, as one might expect and can check by Faá di Bruno's law, k th order continuity between two geometry maps depends on the Taylor expansion of \mathbf{r} , \mathbf{p} perpendicular to the edge E only up to k th order. In particular to the dege E only up to k th order. In particular, it is not necessary to know \mathbf{r} completely.

Example Consider two C^2 geometry maps \mathbf{p} and \mathbf{g} , and a C^2 reparametrization $\mathbf{r} : \mathbf{r}(t, 0) = (0, t)$. $E = \{(t, 0) : t \in [0, 1]\}$ and $e^\perp = (0, -1)$. An alternative parametrization of E is $\{(t^2, 0) : t \in [0, 1]\}$. Such a definition would make the subtle point that G^0 and C^0 can differ as well, since the reparametrization $\mathbf{r}(t, 0) = (0, \sqrt{t})$ is required to equate the derivatives along the boundary. (If \mathbf{p} and \mathbf{g} are polynomials of the same least degree then \mathbf{r} can only be linear and \mathbf{p} and \mathbf{g} share the same parametrization along the edge). We write the conditions for \mathbf{p} and \mathbf{g} joining G^2 via \mathbf{r} along $\mathbf{p}(E)$ and translate them into a commonly used, abbreviated notation where $\mathbf{g}_{uv} := D_{12}\mathbf{g} = D_{e_1, e_2}\mathbf{g}$.

$$\begin{aligned} \mathbf{p} |_{(t,0)} &= \mathbf{g} \circ \mathbf{r} |_{(t,0)}, \\ D_{e^\perp} \mathbf{p} |_{(t,0)} &= D\mathbf{g} |_{\mathbf{r}(t,0)} \langle (D_{e^\perp} \mathbf{r}) |_{(t,0)} \rangle \\ &= D_{e_1} \mathbf{g} |_{(0,t)} (D_{e^\perp} \mathbf{r})^{[1]} |_{(t,0)} + D_{e_2} \mathbf{g} |_{(0,t)} (D_{e^\perp} \mathbf{r})^{[2]} |_{(t,0)}, \quad (D_{e^\perp} \mathbf{r})^{[2]} > 0, \\ &= \mathbf{g}_u(0, t) \alpha(t) + \mathbf{g}_v(0, t) \beta(t), \quad \beta > 0, \\ D_{e^\perp}^2 \mathbf{p} |_{(t,0)} &= (D^2 \mathbf{g} |_{\mathbf{r}(t,0)}) \langle (D_{e^\perp} \mathbf{r}) |_{(t,0)}, (D_{e^\perp} \mathbf{r}) |_{(t,0)} \rangle + D\mathbf{g} |_{\mathbf{r}(t,0)} D_{e^\perp}^2 \mathbf{r} |_{(t,0)} \\ &= \dots + D_2 \mathbf{g} |_{(0,t)} (D_{e^\perp}^2 \mathbf{r})^{[2]} |_{(t,0)} \\ &= \mathbf{g}_{uu}(0, t) \alpha^2(t) + 2\mathbf{g}_{uv}(0, t) \alpha(t) \beta(t) + \mathbf{g}_{vv}(0, t) \beta^2(t) \\ &\quad + \mathbf{g}_u(0, t) \sigma(t) + \mathbf{g}_v(0, t) \tau(t). \end{aligned}$$

In particular, for \mathbf{p} and \mathbf{q} defined on page 196,

$$\text{we compute } D_{e_1} \mathbf{q} |_{(t,0)} = D\mathbf{p} |_{(t,0)} \cdot \left[\frac{1}{3-u} \right]. \quad \diamond$$

The example illustrates that it is convenient and shorter to give separate names, $\alpha, \beta, \sigma, \tau$, to the partial derivatives of \mathbf{r} evaluated on the edge E . We can in fact specify just the partial derivatives rather than all of \mathbf{r} : if we group the two components of each derivative into a vector we can define \mathbf{r} in terms of C^{k-j} -vector fields along $\mathbf{r}(E)$ (Lemma 3.2 of [53]). Provided the derivatives are sufficiently differentiable in the direction e^\perp perpendicular to E we thereby prescribe the Taylor expansion of \mathbf{r} (by the Whitney-Stein Theorem).

8.2.3. Geometric continuity at a vertex

We extend our new notion of geometric continuity to n patches meeting at a common point, e.g. at a point of the global boundary where the patches may meet without necessarily enclosing the point (c.f. Figure 8.7).

Definition 8.2.4 (G^k enclosure) *The C^k geometry maps $\mathbf{g}_i : \Delta_i \rightarrow \mathbb{R}^3, i = 1, \dots, n$, meet G^k via $\mathbf{r}_{i,i+1}, i = 1, \dots, n-1$ with corner $Q \in \mathbb{R}^3$ if*

- \mathbf{g}_i and \mathbf{g}_{i+1} join G^k via $\mathbf{r}_{i,i+1}$ along $\mathbf{g}_{i+1}(E_{i+1})$,
- $\mathbf{g}_i(E_i(0)) = Q$,

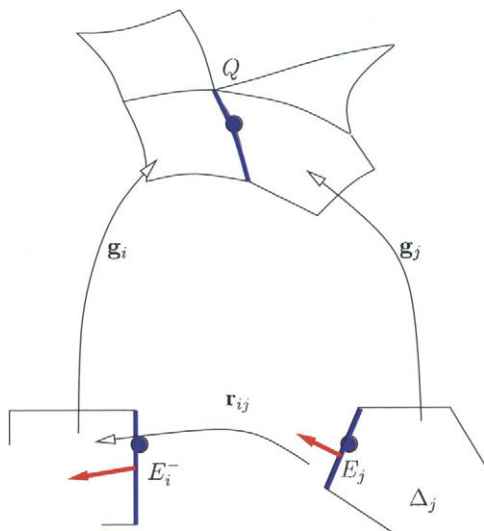


Figure 8.7. Patches meeting at a corner $Q = \mathbf{g}_i(E_i(0)) = \mathbf{g}_i(E_i^-(1)) = \mathbf{g}_j(E_j(0))$, $j = (i \bmod n) + 1$.

- the normalized tangent vectors of each \mathbf{g}_i sweep out a sector of a disk and these tangent sectors lie in a common plane and may touch but do not overlap.

The C^k geometry maps form a G^k enclosure of the vertex Q if additionally \mathbf{g}_n and \mathbf{g}_1 join G^k via $\mathbf{r}_{n,1}$ along $\mathbf{g}_1(E_1)$.

The regularity of the C^k geometry maps implies that each tangent sector is the 1 to 1 image of a corner formed by the non-collinear edges E^- and E of the domain. Moreover, the geometry maps do not wrap around the corner more than once. The common plane referred to above is therefore the tangent plane and, by the implicit function theorem we can expand the geometry maps as a C^k functions at Q .

Where a point is enclosed by three or more patches, additional constraints on \mathbf{r} and \mathbf{g} arise because patches join in a cycle. If one were to start with one patch and added one patch at a time, the last patch would have to match pairwise smoothness constraints across two of its edges. More generally, if all patches are determined simultaneously, a circular interdependence among the smoothness constraints around the vertex results. This circular dependence implies composition constraints on admissible \mathbf{r} and vertex enclosure constraints, on the \mathbf{g}_i . The latter imply for example the important practical fact that it is not always possible to interpolate a given network of C^1 curves by a smooth, regularly parametrized tangent-plane continuous surface with one polynomial patch per mesh facet [85]. A characterization, of when a curve network can be embedded into a curvature continuous surface can be found in [54].

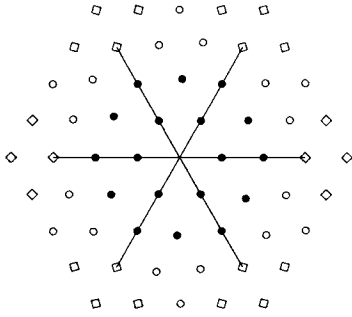


Figure 8.8. The derivative $D_1^m D_2^n \mathbf{p}_i$ of a geometry map \mathbf{p}_i at the central vertex is represented symbolically as a \bullet , \circ , or \diamond placed m units into the direction of the first edge and n into the second. Elements of the total-degree 2-jet \mathbf{j}^2 are marked \bullet , elements of the coordinate-degree 2-jet \mathbf{J}^2 are marked \bullet or \circ , elements of \mathbf{j}^4 are marked \bullet , \circ , or \diamond . The higher-order derivatives H_i^k appearing on the right hand side of the vertex-enclosure constraint system are marked by diamonds \diamond .

To discuss the details, the k -jet notation (c.f. page 199) is helpful:

Definition 8.2.5 The coordinate-degree k -jet, $\mathbf{J}^k \mathbf{p}$, is a vector of directional derivatives $D_1^i D_2^j \mathbf{p}$, $i, j \in \{0, 1, \dots, k\}$ sorted first with key $i + j$, then, within each group, with key i . The total-degree k -jet, $\mathbf{j}^k \mathbf{p}$, consists of the first $\binom{k}{2}$ entries of the coordinate-degree k -jet.

For example, as illustrated in Figure 8.8 (see also [59], p.61, [53]),

$$\begin{aligned} \mathbf{j}^2 \mathbf{p} &:= (\mathbf{p}, D_1 \mathbf{p}, D_2 \mathbf{p}, D_1^2 \mathbf{p}, D_1 D_2 \mathbf{p}, D_2^2 \mathbf{p}), \\ \mathbf{J}^2 \mathbf{p} &:= (\mathbf{p}, D_1 \mathbf{p}, D_2 \mathbf{p}, D_1^2 \mathbf{p}, D_1 D_2 \mathbf{p}, D_2^2 \mathbf{p}, D_1^2 D_2 \mathbf{p}, D_1 D_2^2 \mathbf{p}, D_1^2 D_2^2 \mathbf{p}). \end{aligned}$$

The composition of k -jets, $\mathbf{j}^k \mathbf{g} |_{\mathbf{r}(E)} \circ \mathbf{j}^k \mathbf{r} |_E := \mathbf{j}^k(\mathbf{g} \circ \mathbf{r}) |_E$, is associative and has the identity map id as its neutral element. In k -jet notation the conditions for geometric continuity are

$$\mathbf{j}^k \mathbf{p} |_E = \mathbf{j}^k \mathbf{g} |_{\mathbf{r}(E)} \circ \mathbf{j}^k \mathbf{r} |_E.$$

Composition constraint on reparametrization maps

Assume now that the C^k geometry maps $\mathbf{g}_i, i = 1, \dots, n$, meet G^k via $\mathbf{r}_{i,i+1}$ with corner $\mathbf{g}_i(0), 0 \in \mathbb{R}^2$, i.e. $\mathbf{r}_{i,i+1}(0) = 0$ and

$$\begin{aligned} \mathbf{j}^k \mathbf{g}_1 |_0 &= \mathbf{j}^k(\mathbf{g}_n \circ \mathbf{r}_{n,1}) |_0 = \dots = \mathbf{j}^k(\mathbf{g}_2 \circ \mathbf{r}_{2,3} \circ \dots \circ \mathbf{r}_{n,1}) |_0 \\ &= \mathbf{j}^k(\mathbf{g}_1 \circ \mathbf{r}_{1,2} \circ \dots \circ \mathbf{r}_{n,1}) |_0 \end{aligned}$$

By the implicit function theorem, since \mathbf{g}_1 is regular, $D\mathbf{g}_1$ has a left inverse in the neighborhood of 0 and that implies the *Composition Constraint*

$$\mathbf{j}^k(\mathbf{r}_{1,2} \circ \dots \circ \mathbf{r}_{n,1}) |_0 = \mathbf{j}^k \text{id} |_0,$$

i.e. the Taylor expansion up to k th order of the composition of all reparametrizations must agree with the expansion of the identity map.

Example For $k = 1$ and $n = 3$ and with $\mathbf{r}_{i,j}(0, t) = (t, 0)$ we have

$$\begin{aligned} \mathbf{r}_{1,2} \circ \mathbf{r}_{2,3} \circ \mathbf{r}_{3,1} \Big|_0 &= 0, \\ D\mathbf{r}_{1,2} D\mathbf{r}_{2,3} D\mathbf{r}_{3,1} \Big|_0 &= D \text{ id} \Big|_0. \end{aligned}$$

With scalars λ_i and μ_i , the second equation is equivalent to the matrix product

$$\begin{bmatrix} \lambda_1 & 1 \\ \mu_1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_2 & 1 \\ \mu_2 & 0 \end{bmatrix} \begin{bmatrix} \lambda_3 & 1 \\ \mu_3 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

which is in turn equivalent to

$$\mu_1 \mu_2 \mu_3 = -1, \quad \lambda_j \mu_i = 1, \quad i = 1, 2, 3, j = (i \bmod 3) + 1.$$

In general, the G^1 constraints at 0 imply $\prod_{i=1}^n \mu_i = (-1)^n$. An Expansion of the nonlinear constraints for $k = 2$ is shown in Section 7.2 of [53]. ◊

Lemma 8.2.1 *A symmetric reparametrization $\mathbf{r}_{ij} = \mathbf{r}$ that satisfies the Composition Constraint for a given n is defined by*

$$\mathbf{r}(0) = 0, \quad D\mathbf{r} = \begin{bmatrix} 2\cos(\alpha) & 1 \\ -1 & 0 \end{bmatrix}, \quad \alpha = \frac{2\pi}{n}, \quad D^\kappa \mathbf{r} = 0, \kappa > 1.$$

Proof The eigenvalues of $D\mathbf{r}$ are the n th unit roots $e^{\pm\sqrt{-1}\alpha}$. Therefore $D\mathbf{r}_{1,2} D\mathbf{r}_{2,3} \dots D\mathbf{r}_{n,1} = (D\mathbf{r})^n = \text{Did}$. Since, by Faa di Bruno’s law, at least one factor of the expansion of $D^\kappa(\mathbf{r}_{1,2} \circ \mathbf{r}_{2,3} \circ \dots \circ \mathbf{r}_{n,1})$ is a higher derivative of \mathbf{r} , $D^\kappa(\mathbf{r}_{1,2} \circ \mathbf{r}_{2,3} \circ \dots \circ \mathbf{r}_{n,1}) = 0$, for $\kappa > 1$. ⊗

Vertex enclosure constraints

Another set of constraints applies to geometry maps. Since the G^k constraints of two edge-adjacent patches have support on the first k layers of derivatives counting from each edge, the constraints across two consecutive edges of a geometry map share as variables the derivatives $D_1^m D_2^n$ with $m \leq k$ and $n \leq k$ at the vertex, i.e. overlap on the coordinate-degree k -jet of the geometry map at the vertex (markers \bullet and \circ in Figure 8.8).

If n is the number of patches surrounding the vertex, then there are $n(k+1)^2$ overlapping continuity constraints and an equal number of variables in the form of derivatives in the corresponding coordinate degree k -jets $\mathbf{J}^k \mathbf{p}_i$. Can the constraints can always be enforced by choosing $\mathbf{J}^k \mathbf{p}_i$ appropriately? Already for $k = 1$, the resulting $4n$ by $4n$ constraint matrix \mathcal{M} is not invertible if n is even but it is invertible for n odd. For $k > 1$, more complex rank-deficiencies arise while the right hand side is in general not in the span of the constraint matrix: unlike the univariate case, where we consider only the first k derivatives for G^k joins, the G^k vertex-enclosure constraints involve derivatives of up to order $2k$!

Depending on the data and the construction scheme, some of the higher derivatives are fixed. For example, prescribing boundary curves pins down $D_1^i D_2^0 \mathbf{p}$ for all i . Even when the goal is to just identify degrees of freedom of a free-form spline space [37],[65],

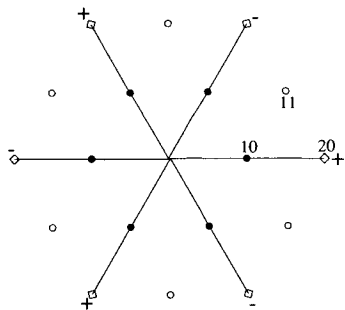


Figure 8.9. The total-degree 1-jet, illustrated as ‘•’, represents the same linear function for all patches. The $n = 6$ constraints involving the ‘11’ derivatives $D_1^1 D_2^1$, \circ , in the coordinate-degree 1-jet but not the total-degree 1-jet give rise to a 6×6 matrix M_c that is rank deficient by 1, i.e. of rank 5. This (vertex-enclosure) constraint can only be solved if the right hand side, defined by the (component normal to the tangent plane of the) ‘20’ derivatives $D_1^2 D_2^0$, \diamond , lies in the span of the constraint matrix. If all reparametrizations are the same, this is the case exactly when the alternating sum of the ‘20’ derivatives is zero, i.e. if the average of the elements marked + equals the average of the elements marked - .

that is, the total-degree k -jets represent a single polynomial expansion up to total degree k at the vertex, a characterization that is also known as the $n + 1$ -Tangent Theorem [81], [56].

Each submatrix $M_{c,i}$ corresponds to the remaining $k(k + 1)/2$ constraints that involve derivatives of total degree greater than k (the diamonds \diamond in Figure 8.8 and 8.9). By blockwise elimination, the rank deficiency of \mathcal{M} equals the rank deficiency of $I - \prod M_i$ and the solvability for arbitrary right hand side depends, after removal of the homogeneous constraints, only on the rank of $I - \prod M_{c,i}$. Each submatrix $M_{c,i}$ decomposes further into skew upper triangular matrices $M_{k,\ell,i}$ of size $\ell \times \ell$ that are grouped along the diagonal.

Example For $k = 1$ we have the constraints at 0 (c.f. Figure 8.9) and $\mathbf{r}_{ab} := D_1^a D_2^b \mathbf{r} |_0$

$$\begin{aligned}
 00 : \quad D_1^0 D_2^0 \mathbf{p}_i &= D_1^0 D_2^0 \mathbf{p}_{i+1} \\
 01 : \quad D_1^0 D_2^1 \mathbf{p}_i &= D_1^1 D_2^0 \mathbf{p}_{i+1} \\
 10 : \quad D_1^1 D_2^0 \mathbf{p}_i &= \lambda D_1^1 D_2^0 \mathbf{p}_{i+1} + \mu D_1^0 D_2^1 \mathbf{p}_{i+1} \\
 11 : \quad D_1^1 D_2^1 \mathbf{p}_i &= \mathbf{r}_{11}^{[1]} D_1^1 D_2^0 \mathbf{p}_{i+1} + \mathbf{r}_{11}^{[1]} D_1^0 D_2^1 \mathbf{p}_{i+1} + \lambda D_1^1 D_2^1 \mathbf{p}_{i+1} + \mu D_1^2 D_2^0 \mathbf{p}_{i+1}.
 \end{aligned}$$

That is, dropping the subscript i for simplicity, each matrix-block $[MN]$ of G^1 constraints

has the form

$$\begin{array}{cccccc}
 & 00 & 10 & 01 & 11 & 20 \\
 00 & 1 & & & & \\
 10 & & \lambda & \mu & & \\
 01 & & 1 & 0 & & \\
 11 & & \mathbf{r}_{11}^{[1]} & \mathbf{r}_{11}^{[2]} & \mu & \lambda
 \end{array}$$

Here N is the last column, below ‘00’. The entries mn to the left of the matrix indicate that the row corresponds to the constraint $D_1^m D_2^n (\mathbf{p}_{i-1} - (\mathbf{p}_i \circ \mathbf{r}_i)) = 0$ while the entries on top indicate the derivatives $D_1^m D_2^n \mathbf{p}_i$ that enter the constraint as variables. For example, the column ‘20’ corresponds to the variable $D_1^2 \mathbf{p}_i$. The constraint rows labeled ‘00’, ‘01’, and ‘10’ correspond to the total-degree 1-jet and are solvable leaving $\mathbf{j}^1 \mathbf{p}_1$ free to determine the tangent plane by its three variables $D_1^0 D_2^0 \mathbf{p}_1|_0 = \mathbf{p}_1(0)$, $D_1 \mathbf{p}_1(0)$ and $D_2 \mathbf{p}_1(0)$ with normal $\mathbf{n} = D_1 \mathbf{p}_1(0) \wedge D_2 \mathbf{p}_1(0)$. The (more interesting) constraint matrix M_c corresponds to the constraint row and column ‘11’. With $\mathbf{p}_i^{11} = \mathbf{n} \cdot D_1^1 D_2^1 \mathbf{p}_i$ and $\mathbf{p}_i^{20} = \mathbf{n} \cdot D_1^2 \mathbf{p}_i$

$$\begin{bmatrix}
 1 & -\mu_1 & & & & \\
 & 1 & -\mu_2 & & & \\
 & & & \ddots & \ddots & \\
 -\mu_n & & & & & 1
 \end{bmatrix}
 \begin{bmatrix}
 \mathbf{p}_1^{11} \\
 \mathbf{p}_2^{11} \\
 \vdots \\
 \mathbf{p}_n^{11}
 \end{bmatrix}
 =
 \begin{bmatrix}
 \lambda_1 \mathbf{p}_1^{20} \\
 \lambda_2 \mathbf{p}_2^{20} \\
 \vdots \\
 \lambda_n \mathbf{p}_n^{20}
 \end{bmatrix}$$

By the Composition Constraint on page 205, $\prod_{i=1}^n \mu_i = (-1)^n$. Therefore the rank of the matrix is $n - 1$ if n is even and n if n is odd [111], [112], [28], [128], [84], [29]. Moreover, if we assume symmetry, i.e. $\mu_i = -1$ and $\lambda_i = \lambda$ for $i = 1, \dots, n$, and if n is even then the vector \mathbf{v} with $\mathbf{v}(i) = (-1)^i$ spans the null space of M_c and therefore the *Alternating Sum Constraint* has to hold for the system to be solvable (c.f. Figure 8.9): if $\lambda \neq 0$ then

$$0 = \sum_{i=1}^n (-1)^i \mathbf{p}_i^{20}$$

For $k = 2$, $[MN]$ has the form

$$\begin{array}{cccccccccccc}
 & 00 & 10 & 01 & 20 & 11 & 02 & 21 & 12 & 22 & 30 & 31 & 40 \\
 00 & 1 & & & & & & & & & & & \\
 10 & & \lambda & \mu & & & & & & & & & \\
 01 & & 1 & 0 & & & & & & & & & \\
 20 & & \mathbf{r}_{20}^{[1]} & \mathbf{r}_{20}^{[2]} & \lambda^2 & 2\lambda\mu & \mu^2 & & & & & & \\
 11 & & \mathbf{r}_{11}^{[1]} & \mathbf{r}_{11}^{[2]} & \lambda & \mu & & & & & & & \\
 02 & & & & 1 & & & & & & & & \\
 21 & & \mathbf{r}_{21}^{[1]} & \mathbf{r}_{21}^{[2]} & A & B & C & 2\lambda\mu & \mu^2 & \lambda^2 & & & \\
 12 & & \mathbf{r}_{12}^{[1]} & \mathbf{r}_{12}^{[2]} & D & E & & \mu & & \lambda & & & \\
 22 & & \mathbf{r}_{22}^{[1]} & \mathbf{r}_{22}^{[2]} & G & H & I & J & K & \mu^2 & L & 2\lambda\mu & \lambda^2
 \end{array}$$

where

$$\begin{aligned}
 A &:= \lambda D + \mathbf{r}_{20}^{[1]}, & B &:= \mu D + \lambda E + \mathbf{r}_{20}^{[2]}, & C &:= \mu E, & D &:= 2\mathbf{r}_{11}^{[1]}, & E &:= 2\mathbf{r}_{11}^{[2]}, \\
 G &:= D^2/2 + 2\lambda\mathbf{r}_{12}^{[1]} + 2\mathbf{r}_{21}^{[1]}, & H &:= DE + 2\mu\mathbf{r}_{12}^{[1]} + 2\lambda\mathbf{r}_{12}^{[2]} + 2\mathbf{r}_{21}^{[2]}, & I &:= E^2/2 + 2\mu\mathbf{r}_{12}^{[2]}, \\
 J &:= 2\mu D + 2\lambda E, & K &:= 2\mu E, & L &:= 2\lambda D + \mathbf{r}_{20}^{[1]} + 2\lambda\mu.
 \end{aligned}$$

[MN] decomposes into the upper left 6×6 block M_t and, from columns ‘21’, ‘12’ and ‘22’,

$$M_c = \begin{bmatrix} 2\lambda\mu & \mu^2 \\ \mu & \\ J & K & \mu^2 \end{bmatrix}, \quad M_{2,1} = \mu^2, \quad M_{2,2} = \begin{bmatrix} 2\lambda\mu & \mu^2 \\ \mu & \end{bmatrix}, \quad N_c = \begin{bmatrix} \lambda^2 \\ \lambda \\ L & 2\lambda\mu & \lambda^2 \end{bmatrix}.$$

◇

Remark: C , J and K above depend directly on D and E in the C^2 reparametrization matrix. To define a weaker notion of continuity in the spirit of Frénet-frame continuity for curves of Section 8.3.1 one would choose C , J and K independently.

For the remainder of the discussion we assume that all \mathbf{r}_{ij} are linear and equal to \mathbf{r} , as in Lemma 8.2.1 (see [86] for a more general analysis and [30] and [126] for a discussion of the case $k = 2$ in terms of Bézier coefficients). Such equal reparametrization is the natural choice for filling ‘ n -sided holes’ (Chapter N -sided Patches), and does not force symmetry of the patches: the tangent vectors, for example, need not span a regular n -gon (but span the affine image of a regular n -gon). If $n = 4$ then $\text{rank}(I - (M_{k,\ell})^n) = 0$. That is, in the tensor-product case, since $N_c = 0$, one full coordinate-jet $\mathbf{J}^k \mathbf{p}_1$ can be chosen freely and $\mathbf{J}^k \mathbf{p}_2$, $\mathbf{J}^k \mathbf{p}_3$ and $\mathbf{J}^k \mathbf{p}_4$ are determined uniquely by the continuity constraints. For general n , the rank deficiencies of $I - M_c^n$ for $k = 1, 2, 3$ are listed in the following table. The results for larger k are summarized in a conjecture in [86].

n	k	1	2	3
3		0	2	2
4		1	3	6
6		1	2	4
even > 6		1	1	2
odd > 3		0	1	0

Since only the Taylor expansion is of interest, the vertex enclosure constraints are independent of the particular representation of the surrounding geometry maps. In particular, the vertex enclosure constraints apply to rational geometry maps in the same fashion as to polynomial geometry maps unless the denominator vanishes. The four known techniques for enforcing the vertex-enclosure constraints are listed in Section 8.4, page 218.

8.2.4. Free-form surface splines

One interpretation of the two types of maps defining the G^k free-form surface spline is that the reparametrizations \mathbf{r} define, by gluing together domains, an *abstract manifold* whose concrete *immersion* into \mathbb{R}^3 is defined by the geometry maps, e.g. Figure 8.10. Free-form surface splines have a bivariate control net with possibly n -sided facets and m -valent nodes. Alternative names are G-splines [62] and geometric continuous patch complexes [53]. Geometric continuous patch complexes differ in their characterization by requiring additionally a *connecting relation* that identifies (glues together) domain edges [53], [50], [104]. This connecting relation is needed when G^k continuity is defined in terms of the *existence* of reparametrizations rather than by explicitly identifying the (first $k + 1$ Taylor terms of the) reparametrization.



Figure 8.10. A free-form spline surface.

Definition 8.2.6 A G^k free-form surface spline is a collection of C^k geometry maps and reparametrizations such that

- at most one reparametrization is associated with any domain edge;
- if a reparametrization \mathbf{r}_{ij} exists between the edge E of the domain Δ_i of \mathbf{g}_i and an edge of the domain of \mathbf{g}_j then \mathbf{g}_i and \mathbf{g}_j join with geometric continuity G^k via \mathbf{r}_{ij} along $\mathbf{g}_i(E)$ and \mathbf{r}_{ij} is C^k ;
- any sequence of C^k geometry maps $\mathbf{g}_i : \Delta_i \mapsto \mathbb{R}^3, i = 1, \dots, n$, such that \mathbf{g}_i and \mathbf{g}_{i+1} join G^k via $\mathbf{r}_{i,i+1}$ along $\mathbf{g}_{i+1}(E_{i+1})$, and $\mathbf{g}_i(E_i(0)) = Q$, meet G^k with corner $Q \in \mathbb{R}^3$.

Free-form surface splines with different reparametrizations do not form a linear vector space. This follows directly from the same statement for G^k continuous curves. For example, we can replace lines with planes in the example shown in Figure 8.5. However, if all reparametrizations agree then we can form an average free-form surface spline and the average inherits the continuity by linearity of differentiation. Section 8.4 outlines constructions.

8.3. EQUIVALENT AND ALTERNATIVE DEFINITIONS

8.3.1. Matching intrinsic curve properties

In [13], Boehm argues that there are (only) two types of geometric continuity: contact of order k , a notion equivalent to G^k continuity, and, secondly, continuity of geometric invariants (but not necessarily of their derivatives).

Two abutting curve segments have *contact of order k* if they are the respective limit of two curve sequences whose corresponding elements intersect in $k + 1$ points and these points coalesce in the limit. In particular, for a space curve $\mathbf{x} : \mathbb{R} \rightarrow \mathbb{R}^3$ with Frénet frame (Chapter 2 on Geometric Fundamentals) spanned by the tangent vector \mathbf{t} , the normal

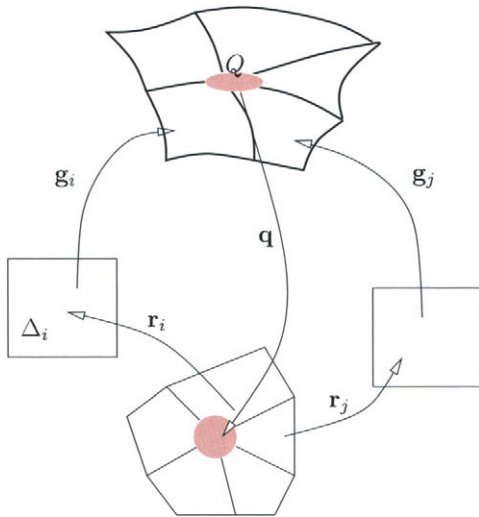


Figure 8.11. In the neighborhood of Q , a G^k free-form surface spline with reparametrizations $\mathbf{r} = \mathbf{r}_i \circ \mathbf{r}_j^{-1}$ can be viewed as a manifold with chart consisting of pieces $\mathbf{q}_i = \mathbf{r}_i^{-1} \circ \mathbf{g}_i^{-1}$, $i = 1, \dots, n$.

vector \mathbf{m} and the binormal \mathbf{b} and $'$ denoting the derivative with respect to *arc length*

$$\begin{aligned} \mathbf{x}'' &= \mathbf{t}' = \kappa \mathbf{m}, & \kappa &= \text{vol}_2[\mathbf{x}', \mathbf{x}''], \\ \mathbf{x}''' &= \mathbf{t}'' = -\kappa^2 \mathbf{t} + \kappa' \mathbf{m} + \kappa \tau \mathbf{b} & \tau \kappa^2 &= \text{vol}_3[\mathbf{x}', \mathbf{x}'', \mathbf{x}'''], \end{aligned}$$

contact of order 2 implies that $\mathbf{x}' = \mathbf{t}$ and \mathbf{x}'' are continuous and therefore that tangent, normal and curvature are continuous. Contact of order 3 implies continuity of \mathbf{x}' , \mathbf{x}'' and \mathbf{x}''' and therefore continuity of Frénet frame, curvature and torsion. Moreover, the *derivative of the curvature must be continuous* tying the entry labeled α in the connection matrix displayed on page 199 in Section 8.2.1 to quantities already listed in the matrix. Similarly, contact of order k in \mathbb{R}^3 requires $\kappa \in C^{k-2}$ and $\tau \in C^{k-3}$ and therefore further dependencies among the entries [42].

By contrast, continuity of the k th geometric invariant, also called *kth order Frénet frame continuity* [31], [38], and abbreviated F^k , does not require that the α -entry (or, more generally, any subdiagonal entry) depend on other entries in the connection matrix. Frénet frame continuity requires that the frame of the two curve pieces agrees and only makes sense in \mathbb{R}^d , for $d \geq k$. Boehm [13] shows that while geometric continuity is projectively invariant, Frénet frame continuity is not. For surfaces, an analogous notion of continuity in terms of fewer restrictions on the connection matrix entries, is pointed out on page 209.

8.3.2. C^k manifolds

Differential geometry has a well-established notion of continuity for a point set: to verify k th order continuity, we must find, for every point Q in the point set, an invertible C^k map (chart) that maps an open surface-neighborhood of Q into an open set in \mathbb{R}^2 . If two surface-neighborhoods, with charts \mathbf{q}_1 and \mathbf{q}_2 respectively, overlap then $\mathbf{q}_2 \circ \mathbf{q}_1^{-1} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ must be a C^k function. This notion of continuity is not constructive: while it defines when a point set can be given the structure of a C^k manifold, say a C^k surface, it neither provides tools to build a C^k surface nor a mechanism suitable for verification by computer.

However, geometric continuity and the continuity of manifolds are closely related: every point in the union of the patches of a G^k free-form surface spline admits local parametrization by C^k charts if the surface does not self-intersect: the union is an immersed C^k surface with piecewise C^k boundary. We face two types of obstacles in establishing this fact. First, the geometry maps should not have geometric singularities on their respective domains since these would prevent invertibility of the charts, and the spline should not self-intersect so that we can map a neighborhood of the point in \mathbb{R}^3 to the plane in a 1-1 fashion. Establishing regularity and non-self-intersection requires potentially expensive intersection testing (Chapter 25 on Intersection Problems). The second apparent obstacle is that the patches that make up the surface are *closed sets that join without overlap*. Therefore the geometry maps cannot directly be used as charts. However, as illustrated in Figure 8.11, we can think of the charts as piecewise maps composed of n maps $\mathbf{q}_i = \mathbf{r}_i^{-1} \circ \mathbf{g}_i^{-1}$ that map open neighborhoods in \mathbb{R}^3 (grey oval) to open neighborhoods in \mathbb{R}^2 (grey disk). The constructions [50,78,19] explicitly start by constructing the union of neighborhoods and connecting charts and then compose these with (rational) spline basis functions.

8.3.3. Tangent and normal continuity

A number of alternative characterizations exist to test a given G^1 free-form spline complex for tangent continuity or to derive G^1 free-form splines. The criteria consist of an equality constraint establishing coplanarity of the first partial derivatives at each point of the common boundary of two patches (c.f. Figure 8.4), and an inequality constraint on the reparametrization that prevents a 180°-flip of the normal for the regular geometry maps. In the following lemma, e is the direction along the preimage E of the common boundary $\mathbf{p}(E)$ and, as is appropriate for least degree polynomial representations (see page 202), \mathbf{p} and \mathbf{g} have the same parametrization along $\mathbf{p}(E)$.

Lemma 8.3.1 (Tangent continuity) *Let \mathbf{p} and \mathbf{g} be regular parametrizations and $\mathbf{p}|_E = \mathbf{g}|_E$. The maps λ , μ and ν are univariate scalar-valued functions and*

$$\begin{aligned} \mathbf{c} &:= D_e \mathbf{p}|_E = D_e \mathbf{g}|_E, \\ \mathbf{n} &:= \mathbf{c} \wedge D_{e^\perp} \mathbf{g}|_E \\ \bar{\mathbf{n}} &:= -\mathbf{c} \wedge D_{e^\perp} \mathbf{p}|_E, \\ \mathbf{t} &:= \mathbf{n} \wedge \mathbf{c} \end{aligned}$$

are functions restricted to an edge E mapping into \mathbb{R}^3 . The following characterizations

of G^1 continuity are equivalent.

$$(1e) \quad D_{e^\perp} \mathbf{p} = \alpha_{\mathbf{p}} \mathbf{t} + \beta_{\mathbf{p}} \mathbf{c}, \quad D_{e^\perp} \mathbf{g} = \alpha_{\mathbf{g}} \mathbf{t} + \beta_{\mathbf{g}} \mathbf{c} \quad \text{and } \alpha_{\mathbf{p}} \alpha_{\mathbf{g}} < 0, \quad (1i)$$

$$(2e) \quad \lambda \mathbf{c} = \mu D_{e^\perp} \mathbf{g} \mid_E + \nu D_{e^\perp} \mathbf{p} \mid_E, \quad \text{and } \mu \nu > 0, \quad (2i)$$

$$(3e) \quad \det [\mathbf{c}, D_{e^\perp} \mathbf{g} \mid_E, D_{e^\perp} \mathbf{p} \mid_E] = \mathbf{n} \cdot D_{e^\perp} \mathbf{p} \mid_E = 0, \quad \text{and } \mathbf{n} \cdot \bar{\mathbf{n}} > 0, \quad (3i)$$

$$(4e) \quad \frac{\bar{\mathbf{n}}}{\|\bar{\mathbf{n}}\|} = \frac{\mathbf{n}}{\|\mathbf{n}\|},$$

With $D\mathbf{r} \mid_E = \begin{bmatrix} \lambda & \mu \\ \nu & \nu \end{bmatrix}$, (2) is the definition of a G^1 join in Definition 8.2.3. Figure 8.4 illustrates the geometric meaning of (2).

Proof Regularity implies $\mathbf{n}(t) \neq 0$ for all t on E .

(1) \implies (2): Adding the two equalities (1e) after multiplication with $\nu = -\alpha_{\mathbf{g}}$ and $\mu = \alpha_{\mathbf{p}}$ respectively (2e) holds in the form $\alpha_{\mathbf{p}} D_{e^\perp} \mathbf{g} - \alpha_{\mathbf{g}} D_{e^\perp} \mathbf{p} = (\alpha_{\mathbf{p}} \beta_{\mathbf{g}} - \alpha_{\mathbf{g}} \beta_{\mathbf{p}}) \mathbf{c}$.

(2) \implies (3): The inner product of both sides of (2e) with \mathbf{n} yields (3e). The cross product \wedge of (2e) with \mathbf{c} followed by the inner product \cdot with $\mu \mathbf{n}$ yields $0 = \mu^2 \|\mathbf{n}\|^2 - \mu \nu \mathbf{n} \cdot \bar{\mathbf{n}}$. Then (2i) implies (3i).

(3) \implies (4): From (3e) we have $\mathbf{n} \perp D_{e^\perp} \mathbf{p}$ and, by definition, $\mathbf{n} \perp \mathbf{c}$. By regularity $\mathbf{n}/\|\mathbf{n}\| = \pm \bar{\mathbf{n}}/\|\bar{\mathbf{n}}\|$ and (3i) decides the sign.

(4) \implies (1): Regularity and (4) imply (1e) that the partial derivatives $D_{e^\perp} \mathbf{p}$ and $D_{e^\perp} \mathbf{g}$ can be expressed in the same (orthogonal) coordinate system-spanned by \mathbf{t} and \mathbf{c} . The cross product of each equality with \mathbf{c} yields $-\bar{\mathbf{n}} = \alpha_{\mathbf{p}} \mathbf{c} \wedge \mathbf{t}$ and $\mathbf{n} = \alpha_{\mathbf{g}} \mathbf{c} \wedge \mathbf{t}$ and by sign comparison (1i). \(\square\)

Formulation (4), comparison of normals, can be turned into a practical tool for quantifying tangent discontinuity. While (1), (2) and (3) are unique only up to scaling, and therefore ‘ ϵ -discontinuity’ measured as $\epsilon > \|\alpha_{\mathbf{p}} \mathbf{t} + \beta_{\mathbf{p}} \mathbf{c} - D_{e^\perp} \mathbf{p}\|$ or $\epsilon > \|\mu D_{e^\perp} \mathbf{g} + \nu D_{e^\perp} \mathbf{p}, -\lambda D_{e^\perp} \mathbf{g}\|$ or $\epsilon > |\mathbf{n} \cdot D_{e^\perp} \mathbf{p}|$ is not well-defined, the angle between the two normals is scale-invariant.

The symmetric characterization (1) asserts the existence of a Taylor expansion along the boundary that is matched by \mathbf{g} and \mathbf{p} . This has been used for constructions [16], [83], [105]. The direct equivalence of (1) and (2) for polynomials is proven in [23] and [55] generalizes this Taylor-expansion approach to k th order.

If \mathbf{p} and \mathbf{g} are *rational* maps, i.e. quotients of polynomials, the continuity conditions can be discussed in terms of polynomials in *homogeneous coordinates* keeping in mind that we may scale freely by a scalar-valued function $\sigma(u, v): D^\kappa \mathbf{p} \mid_E = D^\kappa(\sigma \mathbf{g} \circ \mathbf{r}) \mid_E, \quad \kappa = 0, \dots, k$ [127].

If \mathbf{p} and \mathbf{g} are *polynomials* then, up to a common factor, so are the scalar functions, λ, μ and ν in (1). In fact, after removal of common factors, the degree of the functions is bounded by the degree of \mathbf{p} and \mathbf{g} . This comes in handy when looking for possible reparametrizations \mathbf{r} between two geometry maps.

Lemma 8.3.2 *If \mathbf{p} and \mathbf{g} are polynomials, then, up to a common factor, λ, μ and ν in (1) are polynomials of degree no larger than respectively*

$$\begin{aligned} & \text{degree}(D_{e^\perp} \mathbf{g}) + \text{degree}(D_{e^\perp} \mathbf{p}), \quad \text{degree}(D_{e^\perp} \mathbf{g}) + \text{degree}(D_{e^\perp} \mathbf{p}) \text{ and} \\ & \text{degree}(D_{e^\perp} \mathbf{g}) + \text{degree}(D_{e^\perp} \mathbf{g}). \end{aligned}$$



Figure 8.12. A global periodic parametrization.

Proof Due to regularity of \mathbf{g} along the boundary, the pre-image of the boundary is covered by overlapping intervals U such that for each U there are two components $i, j \in \{x, y, z\}$ with $\det M_{ij} \neq 0$, $M_{ij} := \begin{bmatrix} D_e \mathbf{g}^{[i]} & D_{e \perp} \mathbf{g}^{[i]} \\ D_e \mathbf{g}^{[j]} & D_{e \perp} \mathbf{g}^{[j]} \end{bmatrix}$, $\mathbf{g}^{[j]}$ the j th component of \mathbf{g} . Therefore we can apply Cramer's rule to

$$M_{ij} \begin{bmatrix} \lambda \\ -\mu \end{bmatrix} = \nu \begin{bmatrix} D_{e \perp} \mathbf{p}^{[i]} \\ D_{e \perp} \mathbf{p}^{[j]} \end{bmatrix}$$

and obtain

$$\begin{bmatrix} \lambda \\ -\mu \\ \nu \end{bmatrix} = \frac{\nu}{\det M_{ij}} \begin{bmatrix} -\det \begin{bmatrix} D_{e \perp} \mathbf{p}^{[i]} & D_{e \perp} \mathbf{g}^{[i]} \\ D_{e \perp} \mathbf{p}^{[j]} & D_{e \perp} \mathbf{g}^{[j]} \end{bmatrix} \\ \det \begin{bmatrix} D_e \mathbf{g}^{[i]} & D_{e \perp} \mathbf{p}^{[i]} \\ D_e \mathbf{g}^{[j]} & D_{e \perp} \mathbf{p}^{[j]} \end{bmatrix} \\ \det \begin{bmatrix} D_{e \perp} \mathbf{g}^{[i]} & D_e \mathbf{g}^{[i]} \\ D_{e \perp} \mathbf{g}^{[j]} & D_e \mathbf{g}^{[j]} \end{bmatrix} \end{bmatrix}.$$

The degree of λ , μ and ν is bounded by the degrees of the determinants since the common factor $\nu / \det M_{ij}$ can be eliminated in the constraints. Since $\det M_{ij}$ vanishes at most at isolated points, the degree bound can be extended from U to the whole interval. \square

The characterizations of geometric continuity in terms of geometric invariants (tangents, curvatures) are characterizations of continuity by covariant derivatives [53].

Lemma 8.3.3 *Two C^k geometry maps \mathbf{p} and \mathbf{g} join with geometric continuity G^k via the C^k reparametrization \mathbf{r} along $\mathbf{p}(E)$ if there exist normal vector fields $\mathbf{n}_{\mathbf{p}}$ and $\mathbf{n}_{\mathbf{g}}$ of \mathbf{p} and \mathbf{g} respectively such that*

$$D^\kappa \mathbf{n}_{\mathbf{p}}|_E = D^\kappa \mathbf{n}_{\mathbf{g}}|_{\mathbf{r}(E)}, \kappa = 0, \dots, k-1.$$

In particular, $D\mathbf{n}$ represents the shape operator [68], principal curvatures and directions [120], [58] or the Dupin indicatrix [67]. [58] shows in particular equivalence of G^2 continuity with curvature continuity based on sharing surface normal, principal curvatures and principle curvature directions in \mathbb{R}^3 .

8.3.4. Global and regional reparametrization

Often we can view a free-form surface as a *function* over a domain with the same topological genus, e.g. an isosurface of the electric field surrounding the earth may be computed as a function over a sphere [1]. More generally, we can assemble an object of the appropriate topological genus by identifying edges of a planar domain and then define a standard spline space over the planar domain, mapping into \mathbb{R}^3 with additional periodic boundary conditions and creating 'orbifolds' [35], [123]. This approach circumvents the need for

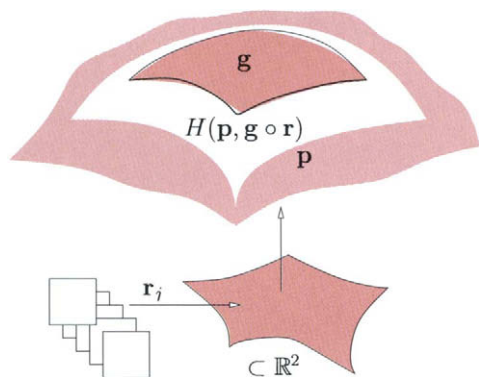


Figure 8.13. Regional parametrization: an n -sided cap g is Hermite-extended via $H(p, g \circ r)$ to match a given spline complex p .

relating many individual domains via reparametrizations, since there is only one *global* domain (modulo periodicity). Basis functions with local support in the domain yield local control. For practical use one has to consider three points. First, the genus of the object has to be fixed before the detailed design process can begin – so one cannot smoothly attach an additional handle later on. Second, the spline functions have to be placed with a density that anticipates for example, an ornate protrusion of the surface where more detail control is required. Third, the ‘hairy ball theorem’, *the hair on a tennis ball cannot be smoothly combed down without leaving a bald spot or making a parting*, implies that the global mapping from subsets of the plane to, say, the sphere has a singularity. The theorem, a consequence of the Borsuk-Ulam theorem, states more formally “If $f : S^2 \rightarrow S^2$ is a continuous map from the sphere to itself then there exists a point where x and $f(x)$ are not orthogonal as vectors in \mathbb{R}^3 ” and in particular, with x a point on the sphere S^2 and $f(x)$ a corresponding unit tangent, “the tangent field on a sphere in \mathbb{R}^3 has to have a singularity”. The singularity is nicely illustrated in [57].

Prautzsch [99], and co-workers [76],[100], and Reif [102], [104] developed the idea of filling n -sided holes by building a *regional parametrization* r for a neighborhood of a point Q where n patches meet (see Figure 8.13). The regional parametrization stands in contrast to the local reparametrizations along an edge used to define free-form surface splines, and the global parametrization discussed earlier. The regional parametrization is composed with a single map g , for example a quadratic polynomial. This approach considerably simplifies reasoning about the resulting surfaces. By separating issues of geometric shape from valence and local topology, verification of smoothness of the resulting surfaces at Q (which could be a major effort of symbolic computing) reduces to showing that r is smooth since smoothness is preserved under composition with an (infinitely smooth) polynomial geometry map. Reparametrization gives $g \circ r$ the structure of a collection of standard (tensor-product or total-degree) patches that can then be connected to a surrounding ring of spline patches p via Hermite interpolation $H(p, g \circ r)$ of degree (degree of g times

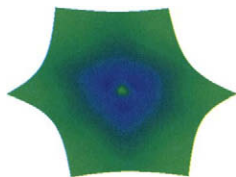


Figure 8.14. Gauss curvature at a higher-order saddle point: the central point must have zero curvature.

degree of \mathbf{r}). By fixing the degree d of the polynomial beforehand, independent of the smoothness k to be achieved, and choosing the C^k parametrization in the domain to be of bidegree $(k+1)$, the regional schemes were the first to claim a construction of C^k surfaces of a degree linear in k , namely of degree $d(k+1)$. In particular, Prautzsch and Reif proposed to choose polynomials of degree $d=2$ which yields G^2 free-form constructions of degree bi-6.

Fixing the degree of \mathbf{g} comes at a cost. While quadratics come in a large number of shapes [93] they are not able to model, for example, higher-order saddle points. A *higher-order saddle point* is a point on a C^2 surface with three or more extremal curvature directions; hence it has zero curvature as illustrated in Figure 8.14. Since quadratics that have a point of zero curvature must be linear this yields flat patches rather than a flat point. For the particular example of a 3-fold saddle point, we could address this shortcoming by increasing the degree of the polynomial to three and the overall degree to bi-9 [104]. But this does not address the underlying problem, namely the mismatch between n patches meeting at Q and the fixed number of coefficients of any single polynomial of fixed degree d . In [89] it was therefore suggested to replace \mathbf{g} by a (total-degree cubic) spline. This yields at least n degrees of freedom.



Figure 8.15. (left) Zero set of $x^2 + y^3 = 0$ and (right) tangent sectors of two possibly smooth patches whose relative position does not allow the reduction to smoothness of a curve by cutting with a transversal plane (dashed line).

8.3.5. Implicit representation

Under suitable monotonicity and regularity constraints the zero set of a trivariate polynomials in BB-form over a unit simplex defines a *single-sheeted, singly connected* piece of surface that we can also call a patch (Chapter 15 on Algebraic Methods). If $p, q : \mathbb{R}^3 \rightarrow \mathbb{R}$ are two trivariate polynomials in BB-form that join C^k as functions at a common point $E \in \mathbb{R}^3$ or across a common curve $E \in \mathbb{R}^3$ then generically (see caveat below) the corresponding patches join with *contact of order k at E*: E is the limit of $k + 1$ intersection points (curves) E_j of functions p_j and q_j converging respectively $E_j \mapsto E, p_j \mapsto p$ and $q_j \mapsto q$. Just as in the parametric case, we have to make sure that the trivariate polynomials are regular at E , as the following example demonstrates: Let $p(x, y, z) = x^2 + y^3, x \leq 0, q(x, y, z) = x^2 + y^3, x \geq 0$. Both p and q are polynomial pieces, have single-sheeted, singly connected zero sets and join C^k for any k ; but the zero set is not smooth at the intersection (see Figure 8.15 left for a cross-section.)

Since we are only concerned with the zero set of the polynomial, we do not actually need that the polynomials join smoothly but can scale the joining pieces by scalar functions a and b ([125], [124], [72]):

Definition 8.3.1 *Two trivariate polynomials p and q join with G^k continuity at Q if*

$$j^k(p \cdot a) |_Q = j^k(q \cdot b) |_Q, \quad a(Q) \neq 0, b(Q) \neq 0.$$

Two trivariate polynomials p and q join with G^k continuity along an irreducible curve $E = (q = 0) \cap (h = 0)$ if

$$j^k p = j^k (aq + bh^{k+1}), \quad a(E) \neq 0, b(E) \neq 0.$$

One could intersect E with a transversal plane to obtain curves meeting in a point and avoid a separate definition for continuity along E (The analogous technique for parametric surfaces is called Linkage Curve Theorem [81], [56].) This approach, however, runs into technical problems, since implicit patches have corners and there may be just one point of intersection with a transversal plane. Similarly, the trivariate k -jet $j^k p$ cannot just be replaced by a k -jet along a line [36] since the domains may lie in the same half space and thus there is no plane that intersects both *restricted* domains in more than the common point (see Figure 8.15, right.)

8.3.6. Generalized subdivision

Near n -valent mesh nodes, uniform generalized subdivision surfaces consist of an infinite sequence of ever smaller, concentric rings that are internally parametrically C^k . The rings also join one another parametrically C^k . Yet, by the Borsuk-Ulam theorem (Section 8.3.4) there cannot be a parametrically C^k mapping from the plane to objects of arbitrary genus without a singularity. In a sense the (effect of the) necessary reparametrization is therefore concentrated near the limit points of n -valent mesh nodes. Correspondingly, the analysis of smoothness of generalized subdivision surfaces has focused on the limits of the n -valent mesh nodes (see e.g. [3] [94], [95]). At present it appears that uniform generalized subdivision cannot generate curvature continuous surfaces unless the control net is in special position.

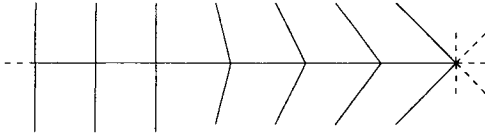


Figure 8.16. Generic ‘Tannenbaum’ layout of Bézier coefficients along a boundary (horizontal line) between two patches that transition from a 4-valent point (left) to an n -valent point (right).

8.4. CONSTRUCTIONS

An algorithm for constructing C^k surfaces subject to data, such as a control net or a prescribed network of curves, is a specification of the C^k reparametrizations \mathbf{r} up to k th order and of the geometry maps \mathbf{g} (see also n -sided hole filling (Chapter N -sided Patches)).

The *generic approach* of stitching together individual spline patches (after possibly manipulation or refinement of the control structure) consists of

- choosing a consistent reparametrization \mathbf{r} at the n -valent points and deriving a reparametrization for two edge-adjacent patches as a Hermite interpolant to the reparametrizations at the end points of the edge;
- solving the vertex-enclosure problem at each point and deriving the geometry maps of abutting patches as a Hermite interpolant to the Taylor expansions at the vertices.

The *generic construction* is as follows.

(1) If n_0 patches join at one boundary endpoint, corresponding to $s = 0, t = 0$, and n_1 patches join the other with $s = 1, t = 0$ then a Hermite interpolant (in s) to the linear symmetric reparametrization of Lemma 8.2.1 at the endpoints, up to k th order is given by

$$\mathbf{r}(t, s) = (s + 2th(s), -t), \quad h(s) = \alpha(s) \cos \frac{2\pi}{n_0} - \beta(s) \cos \frac{2\pi}{n_1},$$

where $\alpha(s)$ and $\beta(s)$ are C^k functions such that $\alpha(s), \beta(s) \geq 0$,

$$\alpha(0) = \beta(1) = 1, \alpha(1) = \beta(0) = 0, \quad D^\kappa \alpha(i) = D^\kappa \beta(i) = 0, i \in \{0, 1\}, \kappa > 1.$$

The interpolant is constructed so that the Taylor expansions of the reparametrization at either endpoint do not interfere, i.e. $D_2^\kappa \mathbf{r}(i) = 0$ for $i \in \{0, 1\}, \kappa > 1$ [47]. For the special, tensor-product transition case of $n_1 = 4$, we obtain the ‘Tannenbaum’ layout of Bézier coefficients shown in Figure 8.16.

(2) The vertex-enclosure problem (for example for G^1, n even) can be solved by one the following four techniques [84]:

1. Choosing H_i^k (e.g. the curve mesh) in the span of the constraint matrix \mathcal{M} ;

2. Splitting patches whose boundaries are prescribed into two or more pieces so that the boundary curves of the split patches can be freely chosen in the span of the constraint matrix (e.g. [32], [33], [96]);
3. Using rational patches to introduce second-order poles at the vertices (e.g. [43], [16] [54]);
4. Using a non-regular parametrization [84], [79], [102].

Thus, if we are not concerned about the degree, it is straightforward to create G^k free-form surface splines for any k . The focus over the past decade has been to reduce the degree of the surface representation, and to obtain better surface shapes (Chapter on Surface Fairing). For example, while the degree of curvature continuous surfaces prior to [99] and [102] was at least bi-9 (100 coefficients per patch) [131], [52] newest results achieve curvature continuity with at most 24 coefficients per patch [90].

Some special techniques, in particular for tangent continuity (c.f. Lemma 8.3.1), are as follows.

- Given the common boundary with derivative $\mathbf{c} = D_e \mathbf{p} |_E = D_e \mathbf{g} |_E$, [16], [83], [23] use a *symmetric* construction, picking \mathbf{t} as minimal Hermite interpolant to the transversal derivative data at the endpoints and

$$D_{e^\perp} \mathbf{p} = \alpha_p \mathbf{t} + \beta_p \mathbf{c}, \quad D_{e^\perp} \mathbf{g} = \alpha_g \mathbf{t} + \beta_g \mathbf{c}$$

with $\mathbf{t} \wedge \mathbf{c} \neq 0$ and $\alpha_p \neq 0 \neq \alpha_g$, cf. Lemma 8.3.1.

- As illustrated in Figure 8.5 the average of two G^1 joined curves or surfaces is generally not G^1 . But if all transversal derivatives of a patch along a boundary $E(t)$ are *collinear* with a vector \mathbf{v} , i.e.

$$D_{e^\perp} \mathbf{p} |_E = p(t) \mathbf{v} \text{ and } D_{e^\perp} \mathbf{q} |_E = q(t) \mathbf{v}$$

for scalar functions p and q with $q/p < 0$ then $\mathbf{n} = N/\|N\|$ where $N(t) := D_e \mathbf{p} |_E \wedge \mathbf{v}$ is a normal common to both patches along the boundary [103].

- Sabin [109] and [82] use formulation (2) of Lemma 8.3.1, $\mathbf{n} \wedge D \mathbf{p} = 0$, to determine versal and transversal derivatives of \mathbf{p} , for given \mathbf{n} – thereby *isolating* the construction of a patch from its neighbor.
- [73] and [83] list a number of choices for reparametrizations for particular constructions.

8.4.1. Free-form surface splines of low degree

Goodman [37] introduced G^1 splines of degree *bi-2* for special control meshes that consist of quadrilateral facets and vertices of valence 3 or 4. The prototype meshes, that can be modified by quadrilateral refinement, the cube mesh and the dual of the cube with lopped off corners, are sphere-like shapes that, when symmetric, are curvature continuous (see [92]). Splitting each original quadrilateral facet 1 to 4, [103] derives a bi-2 G^1 free-form surface spline. Splitting each quadrilateral facet into four triangles, [88] obtains a G^1 construction of total degree 3 that also satisfies the local convex hull property, i.e. the surface points are guaranteed to be an average of the local control mesh. All constructions with *quadratic boundary curve*, however, suffer from a slight *shape defect* when they are to model a higher-order saddle point : due to the Alternating Sum Constraint on page 8.2.3 the quadratic boundary curves must lie on a straight line. If the construction is additionally based on tensor-product patches then the shape defect is very noticeable [49,91] at higher-order saddle points.

The G^k free-form spline constructions of Prautzsch [99] and Reif [102] are of degree $2(k+1)$ if flat regions at higher-order saddle points are acceptable – and of degree $d(k+1)$ if modeling of the local geometry requires a polynomial of degree d . [49] shows that a G^2 degree bi-5 construction is possible; as stated the construction has a shape defect due to the quadratic boundary curve. At least algebraically, we can now model C^2 free-form surfaces of unrestricted patch layout from patches of maximal degree $(d+2, 3)$, $d > 0$ with the flexibility of degree d , C^2 splines at extraordinary points [90]. This approach generalizes to C^k surfaces of piecewise bidegree $k+1, 2k+d-2$.

8.5. ADDITIONAL LITERATURE

Every paper on smooth surfacing defines some, possibly specialized, notion of geometric continuity. Some of the early characterizations can be found in [10] [9],[8] [24] [28] [32] [34] [70] [73] [75] [80] [108,107] [110], [111], [112], [118], [116], [117], [46], , [85], [96], [121], [119] [120] [130] [120] [129] and characterizations for curves in [6], [7], [51],[21], [31].

A number of publications specifically aim at clarifying the notion of geometric continuity. Kahmann discusses curvature and the chain rule [67], DeRose [24] reconciles continuity after reparametrization with the smoothness of manifolds (see also [26], [27]). Liu [74] characterizes C^1 constraints in the form (1) of Lemma 8.3.1. Particularly well-illustrated is Boehm's treatment of geometric and 'visual' continuity [11], [12] [14], [13]. Herron [58] shows directly the equivalence of first and second order geometric continuity with tangent and curvature continuity of surfaces. Further characterizations can be found in [63], [98,97],[23], [55], [25], [126], [122].

Hahn's treatment of geometric continuity [53] (see also [42]) served as a blueprint for Section 8.2 but differs in that he defines a G^k join in terms of the existence of a reparametrization, rather than making the reparametrization part of the definition.

Warren's thesis [125] looks at geometric continuity of implicit representations and [36] is a tour de force of conversions of notions of geometric continuity between two patches.

I am indebted to *Tamas Hermann* for closely reading the article and making numerous suggestions. Kestas Karicauskas introduced me to the term 'Tannenbaum' configuration (Christmas tree configuration).

REFERENCES

1. P. Alfeld, M. Neamtu, and L.L. Schumaker. Bernstein-Bézier polynomials on spheres and sphere-like surfaces. *Computer Aided Geometric Design*, 13(4):333–349, 1996.
2. P. Alfeld and L.L. Schumaker. On the dimension of bivariate spline spaces of smoothness r and degree $d = 3r + 1$. *Numerische Mathematik*, 57(6/7):651–661, July 1990.
3. A.A. Ball and D.J.T. Storry. Conditions for tangent plane continuity over recursively generated B-spline surfaces. *ACM Transactions on Graphics*, 7(2):83–102, 1988.
4. R. Barnhill and J. Gregory. Compatible smooth interpolation in triangles. *J of Approx. Theory*, 15(3):214–225, 1975.
5. P.J. Barry, N. Dyn, R.N. Goldman, and C.A. Micchelli. Identities for piecewise polynomial spaces determined by connection matrices. *Aequationes Math.*, 42(2-3):123–136, 1991.
6. B.A. Barsky. *The Beta-spline: A Local Representation Based on Shape Parameters and Fundamental Geometric Measures*. PhD thesis, University of Utah, December 1981.
7. B.A. Barsky and T.D. DeRose. Geometric continuity of parametric curves: Three equivalent characterizations. *IEEE Computer Graphics and Applications*, 9(6):60–69, November 1989.
8. E. Beeker. Smoothing of shapes designed with free-form surfaces. *Computer-Aided Design*, 18(4):224–232, May 1986.
9. P. Bézier. *Numerical Control: Mathematics and Applications*. Wiley, 1972. translated by R. Forrest.
10. P. Bézier. *Essai de définition numérique des courbes et des surfaces expérimentales*. PhD thesis, Université Pierre et Marie Curie, February 1977.
11. W. Boehm. Curvature continuous curves and surfaces. *Computer-Aided Design*, 18(2):105–106, March 1986.
12. W. Boehm. Smooth curves and surfaces. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 175–184. SIAM, Philadelphia, 1987.
13. W. Boehm. On the definition of geometric continuity. *Computer-Aided Design*, 20(7):370–372, 1988. Letter to the Editor.
14. W. Boehm. Visual continuity. *Computer-Aided Design*, 20(6):307–311, 1988.
15. H. Bohl and U. Reif. Degenerate Bézier patches with continuous curvature. *Computer Aided Geometric Design*, 14(8):749–761, 1997.
16. H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, 17(3):289–298, July 1983.
17. G.M. Constantine and T.H. Savits. A multivariate Faà di Bruno formula with applications. *Trans. Amer. Math. Soc.*, 348(2):503–520, 1996.
18. S.A. Coons. Surfaces for computer-aided design of space forms. Technical Report MIT/LCS/TR-41, Massachusetts Institute of Technology, June 1967.
19. J.C. Navau and N.P. Garcia. Modelling surfaces from planar irregular meshes. *Computer Aided Geometric Design*, 17(1):1–15, 2000.
20. C. de Boor, K. Höllig, and M. Sabin. High accuracy geometric hermite interpolation. *Computer Aided Geometric Design*, 4(4):269–278, December 1987.
21. W.L.F. Degen. Some remarks on Bézier curves. *Computer Aided Geometric Design*,

- 5(3):259–268, August 1988.
22. W.L.F. Degen. High accurate rational approximation of parametric curves. *Computer Aided Geometric Design*, 10(3):293–314, August 1993.
 23. W.L.F. Degen. Explicit continuity conditions for adjacent Bézier surface patches. *Computer Aided Geometric Design*, 7(1-4):181–189, June 1990.
 24. T. DeRose. *Geometric Continuity: A Parametrization Independent Measure of Continuity for Computer Aided Design*. PhD thesis, UC Berkeley, California, 1985.
 25. T.D. DeRose. Necessary and sufficient conditions for tangent plane continuity of Bézier surfaces. *Computer Aided Geometric Design*, 7(1-4):165–179, June 1990.
 26. T.D. DeRose and B.A. Barsky. An intuitive approach to geometric continuity for parametric curves and surfaces. In M. Wein and E.M. Kidd, editors, *Graphics Interface '85 Proceedings*, pages 343–351. Canadian Inf. Process. Soc., 1985.
 27. T.D. DeRose and B.A. Barsky. Geometric continuity, shape parameters, and geometric constructions for Catmull-Rom splines. *ACM Transactions on Graphics*, 7(1):1–41, January 1988.
 28. W.-H. Du and F.J.M. Schmitt. New results for the smooth connection between tensor product bezier patches. In N. Magnenat-Thalmann and D. Thalmann, editors, *New Trends in Computer Graphics (Proceedings of CG International '88)*, pages 351–363. Springer-Verlag, 1988.
 29. W.-H. Du and J.M. Schmitt. G^1 smooth connection between rectangular and triangular Bézier patches at a common corner. In P.J. Laurent, A. LeMéhauté, and L.L. Schumaker, editors, *Curves and Surfaces*, pages 165–168. Academic Press, 1991.
 30. W.-H. Du and J.M. Schmitt. On the G^2 continuity of piecewise parametric surfaces. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 197–207. Academic Press, 1991.
 31. N. Dyn and C.A. Micchelli. Piecewise polynomial spaces and geometric continuity of curves. *Numer. Math.*, 54(3):319–337, 1988.
 32. G. Farin. Smooth interpolation to scattered 3D data. In R.E. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 43–63. North-Holland, 1983.
 33. G. Farin. Triangular Bernstein–Bézier patches. *Computer Aided Geometric Design*, 3(2):83–127, 1986.
 34. I.D. Faux and M.J. Pratt. *Computational Geometry for Design and Manufacture*. Ellis Horwood, Chichester, 1979.
 35. H. Ferguson, A. Rockwood, and J. Cox. Topological design of sculptured surfaces. In E.E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):149–156, July 1992.
 36. T. Garrity and J. Warren. Geometric continuity. *Computer Aided Geometric Design*, 8(1):51–66, February 1991.
 37. T.N.T. Goodman. Closed surfaces defined from biquadratic splines. *Constructive Approximation*, 7(2):149–160, 1991.
 38. T.N.T. Goodman. Construction piecewise rational curves with frenet frame continuity. *Computer Aided Geometric Design*, 7(1-4):15–31, June 1990.
 39. T.N.T. Goodman. Joining rational curves smoothly. *Computer Aided Geometric Design*, 8(6):443–464, December 1991.

40. W. Gordon. Free-form surface interpolation through curve networks. Technical Report GMR-921, General Motors Research Laboratories, 1969.
41. W. Gordon. Spline-blended surface interpolation through curve networks. *J of Math and Mechanics*, 18(10):931–952, 1969.
42. J. Gregory. Geometric continuity. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 353–372. Academic Press, 1989.
43. J.A. Gregory. *Smooth Interpolation Without Twist Constraints*, pages 71–88. Academic Press, 1974.
44. J.A. Gregory and K.H. Lau. High order continuous polygonal patches. In G. Farin, H. Hagen, H. Noltemeier, and W. Knödel, editors, *Geometric modelling*, volume 8 of *Computing. Supplementum*, pages 117–132. Springer, Wien / New York, 1993.
45. J.A. Gregory, V.K.H. Lau, and J.M. Hahn. High order continuous polygonal patches. In *Geometric Modelling*, pages 117–132. Springer, Vienna, 1993.
46. J.A. Gregory and J.M. Hahn. Geometric continuity and convex combination patches. *Computer Aided Geometric Design*, 4(1-2):79–89, 1987. Special issue on topics in computer aided geometric design (Wolfenbüttel, 1986).
47. J.A. Gregory and J.M. Hahn. A C^2 polygonal surface patch. *Computer Aided Geometric Design*, 6(1):69–75, 1989.
48. J.A. Gregory and J. Zhou. Filling polygonal holes with bicubic patches. *Computer Aided Geometric Design*, 11(4):391–410, 1994.
49. J.A. Gregory and J. Zhou. Irregular C^2 surface construction using bi-polynomial rectangular patches. *Computer Aided Geometric Design*, 16(5):423–435, 1999.
50. C.M. Grimm and J.F. Hughes. Modeling surfaces of arbitrary topology using manifolds. *Computer Graphics*, (Annual Conference Series), 29:359–368, 1995.
51. H. Hagen. Bézier-curves with curvature and torsion continuity. *Rocky Mtn. J of Math.*, 16(3):629–638, 1986.
52. J. Hahn. Filling polygonal holes with rectangular patches. In *Geometric Modeling*, Blaubeuren, Germany, 1988.
53. J.M. Hahn. Geometric continuous patch complexes. *Computer Aided Geometric Design*, 6(1):55–67, 1989.
54. T. Hermann. G^2 interpolation of free-form curve networks by biquintic Gregory patches. *Computer Aided Geometric Design*, 13:873–893, 1996.
55. T. Hermann and G. Lukács. A new insight into the G^n continuity of polynomial surfaces. *Computer Aided Geometric Design*, 13(8):697–707, 1996.
56. T. Hermann, G. Lukács, and F. Wolter. Geometrical criteria on the higher order smoothness of composite surfaces. *Computer Aided Geometric Design*, 16(9):907–911, 1999.
57. G. Herron. Smooth closed surfaces with discrete triangular interpolants. *Computer Aided Geometric Design*, 2(4):297–306, 1985.
58. G. Herron. *Techniques for Visual Continuity*, pages 163–174. SIAM, 1987.
59. M.W. Hirsch. *Differential Topology*. Springer-Verlag, New York, 1994. Corrected reprint of the 1976 original.
60. K. Höllig and J. Koch. Geometric hermite interpolation. *Computer Aided Geometric Design*, 12(6):567–580, 1995.

61. K. Höllig and J. Koch. Geometric hermite interpolation with maximal order and smoothness. *Computer Aided Geometric Design*, 13(8):681–695, 1996.
62. K. Höllig and H. Mögerle. G-splines. *Computer Aided Geometric Design*, 7(1-4):197–207, June 1990.
63. K. Höllig. Geometric continuity of spline curves and surfaces. TR 645, Computer Sciences Department, University of Wisconsin, Madison, WI, June 1986.
64. K. Höllig. Algorithms for rational spline curves. In *Transactions of the Fifth Army Conference on Applied Mathematics and Computing (West Point, NY, 1987)*, pages 287–300. U.S. Army Res. Office, Research Triangle Park, NC, 1988.
65. K. Höllig and H. Mögerle. G-splines. *Computer Aided Geometric Design*, 7(1-4):197–207, 1990. Curves and surfaces in CAGD '89 (Oberwolfach, 1989).
66. J. Hoschek. Free-form curves and free-form surfaces. *Computer Aided Geometric Design*, 10(3):173–174, August 1993.
67. J. Kahmann. *Continuity of Curvature Between Adjacent Bézier Patches*, pages 65–75. North-Holland Publishing Company, Amsterdam, 1983.
68. T. Jensen. Assembling triangular and rectangular patches and multivariate splines. In G. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 203–220. SIAM, Philadelphia, 1987.
69. T.W. Jensen, C.S. Petersen, and M.A. Watkins. Practical curves and surfaces for a geometric modeler. *Computer Aided Geometric Design*, 8(5):357–370, November 1991.
70. A.K. Jones. Nonrectangular surface patches with curvature continuity. *Computer-Aided Design*, 20(6):325–335, 1988.
71. B. Jüttler and P. Wassum. Some remarks on geometric continuity of rational surface patches. *Computer Aided Geometric Design*, 9(2):143–158, June 1992.
72. J. Li, J. Hoschek, and E. Hartmann. G^{n-1} -functional splines for interpolation and approximation of curves, surfaces and solids. *Computer Aided Geometric Design*, 7(1-4):209–220, June 1990.
73. D. Liu and J. Hoschek. GC^1 continuity conditions between adjacent rectangular and triangular Bézier surface patches. *Computer-Aided Design*, 21(4):194–200, 1989.
74. D. Liu. A geometric condition for smoothness between adjacent bézier surface patches. *Acta Mathematicae Applicatae Sinica*, 9(4), 1986.
75. J. Manning. Continuity conditions for spline curves. *The Computer J*, 17(2):181–186, 1974.
76. T. Müller and H. Prautzsch. *A Free Form Spline Software Package*. Universität Karlsruhe, 1996.
77. L.J. Nachman. Matching NURBS surfaces G^1 and G^2 . *The mathematics of surfaces, VI (Uxbridge, 1994)*, pages 475–515. Oxford Univ. Press, 1996.
78. J.C. Navau and N.P. Garcia. Modeling surfaces from meshes of arbitrary topology. *Computer Aided Geometric Design*, 17(7):643–671, 2000.
79. M. Neamtu and P.R. Pfluger. Degenerate polynomial patches of degree 4 and 5 used for geometrically smooth interpolation in R^3 . *Computer Aided Geometric Design*, 11(4):451–474, 1994.
80. G. Nielson. *Some Piecewise Polynomial Alternatives to Splines Under Tension*, pages 209–235. Academic Press, 1974.

81. J. Pegna and F. Wolter. Geometric criteria to guarantee curvature continuity of blend surfaces. *ASME Transactions, J. of Mech. Design*, 114, 1992.
82. J. Peters. Local cubic and bicubic C^1 surface interpolation with linearly varying boundary normal. *Computer Aided Geometric Design*, 7:499–515, 1990.
83. J. Peters. Smooth mesh interpolation with cubic patches. *Computer-Aided Design*, 22(2):109–120, 1990.
84. J. Peters. Parametrizing singularly to enclose vertices by a smooth parametric surface. In S. MacKay and E. M. Kidd, editors, *Graphics Interface '91, Calgary, Alberta, 3–7 June 1991: proceedings*, pages 1–7, 243 College St, 5th Floor, Toronto, Ontario M5T 2Y1, Canada, 1991. Canadian Information Processing Society.
85. J. Peters. Smooth interpolation of a mesh of curves. *Constructive Approximation*, 7:221–247, 1991. Winner of SIAM Student Paper Competition 1989.
86. J. Peters. Joining smooth patches at a vertex to form a C^k surface. *Computer Aided Geometric Design*, 9:387–411, 1992.
87. J. Peters. Surfaces of arbitrary topology constructed from biquadratics and bicubics. In N.S. Sapidis, editor, *Designing Fair Curves and Surfaces: Shape Quality in Geometric Modeling and Computer-Aided Design*, pages 277–293. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1994.
88. J. Peters. C^1 -surface splines. *SIAM Journal on Numerical Analysis*, 32(2):645–666, 1995.
89. J. Peters. Curvature continuous surfacing, 1998. Presentation at the Oberwolfach seminar on Curves and Surfaces.
90. J. Peters. Curvature continuous free-form surfaces of degree 5,3. Technical Report 02, Dept CISE, University of Florida, 2001.
91. J. Peters. Modifications of PCCM. Technical Report 01, Dept CISE, University of Florida, 2001.
92. J. Peters and L. Kobbelt. The platonic spheroids. Technical Report 97-052, Dept of Computer Sciences, Purdue University, 1998.
93. J. Peters and U. Reif. The 42 equivalence classes of quadratic surfaces in affine n -space. *Computer Aided Geometric Design*, 15:459–473, 1998.
94. J. Peters and U. Reif. Analysis of generalized B-spline subdivision algorithms. *SIAM Journal on Numerical Analysis*, 35(2):728–748, April 1998.
95. J. Peters and G. Umlauf. Gaussian and mean curvature of subdivision surfaces. In R. Cipolla and R. Martin, editors, *The Mathematics of Surfaces IX*, pages 59–69. Springer, 2000.
96. B.R. Piper. Visually smooth interpolation with triangular Bézier patches. In G. Farin, editor, *Geometric Modeling : Algorithms and New Trends*, pages 221–233. SIAM, Philadelphia, 1987.
97. H. Pottmann. A projectively invariant characterization of G^2 continuity for rational curves. In G. Farin, editor, *NURBS for Curve and Surface Design*, pages 141–148. SIAM, Philadelphia, 1991.
98. H. Pottmann. Projectively invariant classes of geometric continuity for CAGD. *Computer Aided Geometric Design*, 6(4):307–321, 1989.
99. H. Prautzsch. Freeform splines. *Computer Aided Geometric Design*, 14(3):201–206, 1997.

100. H. Prautzsch and G. Umlauf. Triangular G^2 splines. In L.L. Schumaker P.J. Laurent, and A. LeMéhauté, editors, *Curve and Surface Design*, pages 335–342. Vanderbilt University Press, 2000.
101. A. Rababah. High order approximation method for curves,. *Computer Aided Geometric Design*, 12(1):89–102, 1995.
102. U. Reif. TURBS—topologically unrestricted rational B -splines. *Constr. Approx.*, 14(1):57–77, 1998.
103. U. Reif. Biquadratic G -spline surfaces. *Computer Aided Geometric Design*, 12(2):193–205, 1995.
104. U. Reif. *Analyse und Konstruktion von Subdivisionsalgorithmen für Freiformflächen beliebiger Topologie*. Shaker Verlag, Aachen, 1999.
105. G. Renner. Polynomial n -sided patches. *Curves and Surfaces*, pages xx–xx, 1990.
106. W. Rudin. *Principles of Mathematical Analysis*, 3rd Edition. McGraw-Hill, Tokyo, 1976.
107. M.A. Sabin. Conditions for continuity of surface normals between adjacent parametric surfaces. Technical Report VTO/MS/151, British Aircraft Corporation, 1968.
108. M.A. Sabin. Parametric splines in tension. Technical Report VTO/MS/160, British Aircraft Corporation, 1970.
109. M.A. Sabin. Conditions for continuity of surface normal between adjacent parametric surfaces. Technical Report VTO/MS/151, Tech. Rep., British Aircraft Corporation Ltd., 1968.
110. M.A. Sabin. *The Use of Piecewise Forms for the Numerical Representation of Shape*. PhD thesis, Computer and Automation Institute, 1977.
111. R. Sarraga. G^1 interpolation of generally unrestricted cubic Bézier curves. *Computer Aided Geometric Design*, 4(1-2):23–40, 1987.
112. R. Sarraga. Errata: G^1 interpolation of generally unrestricted cubic Bézier curves. *Computer Aided Geometric Design*, 6(2):167–172, 1989.
113. H.-P. Seidel. Geometric constructions and knot insertion for geometrically continuous spline curves of arbitrary degree. Technical Report 90-024, Dept of Computer Sciences, University of Waterloo, 1990.
114. H.-P. Seidel. New algorithms and techniques for computing with geometrically continuous spline curves of arbitrary degree. *Modélisation mathématique et Analyse numérique (M^2AN)*, 26:149–176, 1992.
115. H.-P. Seidel. Polar forms for geometrically continuous spline curves of arbitrary degree. *ACM Transactions on Graphics*, 12(1):1–34, January 1993.
116. L.A. Shirman and C.H. Séquin. Local surface interpolation with Bézier patches: Errata and improvements. *Computer Aided Geometric Design*, 8(3):217–222, August 1991.
117. L.A. Shirman. *Construction of Smooth Curves and Surfaces From Polyhedral Models*. PhD thesis, Computer Science Division (EECS), University of California, Berkeley, 1990.
118. L.A. Shirman and C.H. Sequin. Local surface interpolation with bezier patches. *Computer Aided Geometric Design*, 4(4):279–295, December 1987.
119. J. van Wijk. Bicubic patches for approximating non-rectangular control-point meshes. *Computer Aided Geometric Design*, 3(1):1–13, 1986.

120. M. Veron, R. Riss, and J.-P. Musse. Continuity of biparametric surface patches. *Computer-Aided Design*, 8:267–273, October 1976.
121. A. Vinacua and P. Brunet. A construction for VC^1 continuity for rational Bézier patches. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 601–611. Academic Press, 1989.
122. Du W.-H. *Etude sur la représentation d surfaces complexes: application à la reconstruction de surfaces échantillonnes*. PhD thesis, Ecole National Supérieure des télé-communications, October 1988.
123. J. Wallner and H. Pottmann. Spline orbifolds. In *Curves and surfaces with applications in CAGD (Chamonix-Mont-Blanc, 1996)*, pages 445–464. Vanderbilt Univ. Press, 1997.
124. J. Warren. Blending algebraic surfaces. *ACM Transactions on Graphics*, 8(4):263–278, October 1989.
125. J.D. Warren. On algebraic surfaces meeting with geometric continuity. Technical Report TR86-770, Cornell University, Computer Science Department, August 1986.
126. P. Wassum. *Bedingungen und Konstruktionen zur geometrischen Stetigkeit und Anwendungen auf approximative Basistransformationen*. PhD thesis, TH Darmstadt, 1991.
127. P. Wassum. Geometric continuity between adjacent rational Bézier surface patches. In G. Farin, H. Hagen, H. Noltemeier, and W. Knödel, editors, *Geometric modelling*, volume 8 of *Computing. Supplementum*, pages 291–316. Springer, Wien/New York, 1993.
128. M. Watkins. Problems in geometric continuity. *Computer-Aided Design*, 20(8):499–502, 1988.
129. J. Weber. Constructing a boolean-sum curvature-continuous surface. In F.L. Krause and H. Jansen, editors, *Advanced Geometric Modelling for Engineering Applications*, pages 103–116, 19xx.
130. F. Yamaguchi. *Curves and Surfaces in Computer-Aided Geometric Design*. Springer, Berlin, 1988.
131. X. Ye. Curvature continuous interpolation of curve meshes. *Computer Aided Geometric Design*, 14(2):169–190, 1997.

Chapter 9

Splines on Surfaces

Marian Neamtu

This chapter addresses the area of spline theory concerned with the construction of functions defined on manifolds in three-dimensional Euclidean space. For the most part, the mathematical aspects of this discipline are in their infancy and therefore the presentation will have an exploratory character.

9.1. INTRODUCTION

Thus far in this book we have mostly encountered spline curves and surfaces whose parameter domains are subsets of the real line or the Euclidean plane. In particular, in several chapters of this book we became accustomed to the idea that a spline surface is the graph of a bivariate real-valued function or, alternatively, a parametric surface, which is the image of a planar domain under a vector function, or a collection of such functions. The parametric or free-form surfaces that are typically considered in the CAGD literature are composite surfaces consisting of a collection of individual surface patches of the form $f_i(\mathbf{S}_i)$, where

$$f_i : \mathbf{S}_i \rightarrow \mathbb{R}^3, \quad i = 1, \dots, N, \quad (9.1)$$

is a three-component vector function whose domain \mathbf{S}_i is a “simple” planar region, such as the standard triangle or the unit square.

The functions f_i are usually chosen to be polynomials of a fixed “low” degree, *e.g.*, Bézier triangles or tensor products of univariate Bernstein polynomials, or rational functions. Moreover, the patches $f_i(\mathbf{S}_i)$ “fit together” so that they form a globally continuous, or even smooth, surface (see Figure 9.1). We refer the reader to Chapters 5, 7, and 8, for a discussion on constructing smooth free-form surfaces with polynomial and rational surface patches.

The popularity of polynomial and rational composite surfaces in the CAGD community can be explained by the fact that they can be used to represent, in a robust way, a great variety of shapes and surfaces of arbitrary topological genus. However, in spite of the

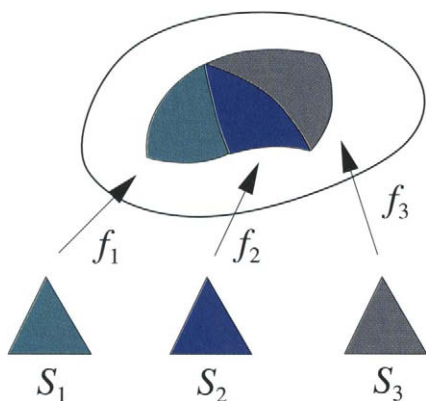


Figure 9.1. Composite parametric surface.

high flexibility and other attractive properties of such surfaces (see Chapters 4, 5, and 8), in many situations of practical interest it is desirable to extend the above concept of a spline surface a step farther. Namely, it makes sense, at least in principle, to consider the more general case in which the domains S_i in (9.1) are *non-planar* and hence are themselves surfaces. In particular, consider the problem of constructing scalar or vector-valued functions

$$f : S \rightarrow \mathbb{R}^k \quad (k = 1, 2, \dots), \quad (9.2)$$

whose domain S is a general surface in \mathbb{R}^3 . There are a host of situations in applied sciences where such functions are likely to be useful. Let us briefly mention a few applications. The reader will undoubtedly be able to come up with many other examples on his/her own.

- *Scalar Fields.* The function f can describe a scalar field associated with the surface S . For example, S could be the surface of a combustion engine part and f could be the temperature as a function of location on the surface. Or, S could be the surface of the earth and f one of the following quantities measured at various locations on S : altitude above the sea level, temperature, intensity of the magnetic field, etc. Another example: given values of f measured on S , such as discrete pressures on an aircraft wing, an objective might be to determine the pressure at arbitrary locations on the wing. Yet another application, of interest in CAGD, is when f stands for a scalar field that reflects an aspect of the visual quality of S , such as the Gaussian curvature.
- *Vector Fields.* A rich source of examples of vector fields defined on surfaces is fluid dynamics. Here the objective is to model/reconstruct velocity fields and other flow quantities that are governed by the Navier-Stokes equations. In particular, in meteorology and global atmospheric modeling, the fluid flow on the surface of the earth is described in terms of the horizontal velocity vector field (with longitudinal and

latitudinal components), together with atmospheric pressure [15,34,79]. Another example comes from the so-called moving boundary problems in partial differential equations, in which the surface \mathbf{S} varies with time and represents a moving interface between solid and liquid phases of a given material substance, ice and water, say. Typically, the vector field associated with \mathbf{S} has several components, for example the normal vectors to the surface \mathbf{S} , its mean curvature, and the velocity of the moving boundary [45].

- *Surface Design/Reconstruction.* Some surfaces are often closely related to a given “reference” surface \mathbf{S} or are even directly determined by this surface. For example, any scalar field f on \mathbf{S} can be thought of or visualized as a surface in its own right, *e.g.*, the surface

$$\mathbf{S}' := \{\mathbf{s} + f(\mathbf{s})\mathbf{n}_{\mathbf{s}}, \mathbf{s} \in \mathbf{S}\}, \quad (9.3)$$

where $\mathbf{n}_{\mathbf{s}}$ is the unit surface normal at the point \mathbf{s} . This is the reason why surfaces of this type are also sometimes referred to as “surfaces on surfaces” [11]. For example, \mathbf{S}' could represent the true surface of the earth, with \mathbf{S} being the reference sphere/ellipsoid and f the height above sea level. Or, \mathbf{S}' could be an offset surface to \mathbf{S} , *i.e.*, a surface whose distance to \mathbf{S} is a fixed value (in which case the function f in (9.3) is identically equal to this value). In each of these cases it may be advantageous to use \mathbf{S} as a natural parametric domain for \mathbf{S}' .

“Surfaces on surfaces” can also be viewed as surfaces in a higher-dimensional space. This is because the set

$$\mathbf{S}'' := \{(\mathbf{s}, f(\mathbf{s})), \mathbf{s} \in \mathbf{S}\} \subset \mathbb{R}^3 \times \mathbb{R}^k, \quad (9.4)$$

can be thought of as a two-dimensional manifold imbedded in \mathbb{R}^{k+3} . In particular, if f is a scalar function, then \mathbf{S}'' is a two-dimensional surface in four-dimensional Euclidean space.

The first step in most surface reconstruction, modeling, and/or data-fitting problems, aimed at recovering an unknown function f from a set of data or at solving a partial differential equation, is to restrict the class of functions that are used. These restricted classes usually consist of “simple” functions that are suitable for numerical manipulation and at the same time can approximate well other (more general) functions. Thus, we typically work with spline-like or finite-element type functions, which have “piecewise character”, *i.e.*, they belong to a linear space with locally finite dimension. In the context of this chapter, f will be “composed” of functions $f_i : \mathbf{S}_i \rightarrow \mathbb{R}^k$ ($k = 1, 2, \dots$), where the \mathbf{S}_i ’s are in general non-planar surfaces tessellating the domain \mathbf{S} *i.e.*, $\mathbf{S} = \bigcup \mathbf{S}_i$.

At this point, the reader may wonder, and rightly so, whether considering general surfaces \mathbf{S}_i , instead of planar ones, is indeed a genuine and worthy generalization of the setting (9.1). This is a legitimate point since, strictly speaking, the general problem (9.2) can in fact be reduced to the simpler case of planar parametric domains. For example, if f is scalar ($k = 1$) and if we interpret $f(\mathbf{S})$ as the surface \mathbf{S}' in (9.3), then one can use

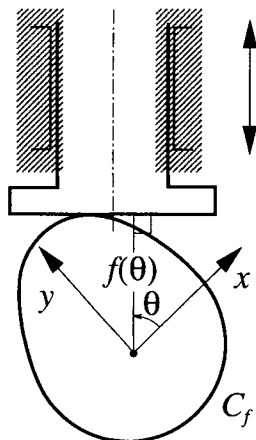


Figure 9.2. Cam mechanism with a flat face follower.

standard parametric polynomial spline techniques to represent/reconstruct f . Alternatively, it is possible to interpret f in the framework (9.4), in which case one could employ (4D) parametric polynomial splines to reconstruct \mathbf{S}'' and thereby also f , being the fourth component of \mathbf{S}'' (as was done *e.g.*, in [4]). Lastly, in many situations of practical interest, such as in the case of a sphere, the surface \mathbf{S} can be mapped onto a planar region, which again allows us to think of functions defined on \mathbf{S} as functions whose domain is planar. However, we shall see that in many instances it is beneficial to approach a given reconstruction problem in the mind-set of (9.2) and not (9.1). Before we address this point in more detail, it will be instructive to look at some concrete examples.

Example 1. Although in this chapter we are primarily interested in splines on surfaces, the following example, concerning the modeling of planar curves, will help motivate several important points. In particular, below we describe some interesting facts that arise in the context of designing cam profiles in mechanical engineering.

Cams are used in many kinds of mechanical devices. Their basic purpose is to convert rotary motion into linear motion that has a particular displacement as a function of time. Let us restrict ourselves here to cam mechanisms with the so-called flat face follower. The usual way to design a cam profile in this case is to construct the so-called support function f , which describes the displacement of the follower (*i.e.*, the vertical distance of the follower from the center of the cam) as a function of the rotation angle θ , see Figure 9.2. Thus, the support function is a scalar function whose domain \mathbf{S} is the unit circle, parametrized by the polar angle θ , that is $f : \mathbf{S} \rightarrow \mathbb{R}$, $\mathbf{S} := \{(\cos \theta, \sin \theta), \theta \in [0, 2\pi)\}$. The associated profile curve of the cam can be thought of as a parametric curve, *i.e.*, a vector function $\mathbf{C}_f : \mathbf{S} \rightarrow \mathbb{R}^2$, whose image is given by

$$\mathbf{C}_f = \{(x(\theta), y(\theta)), \theta \in [0, 2\pi)\},$$

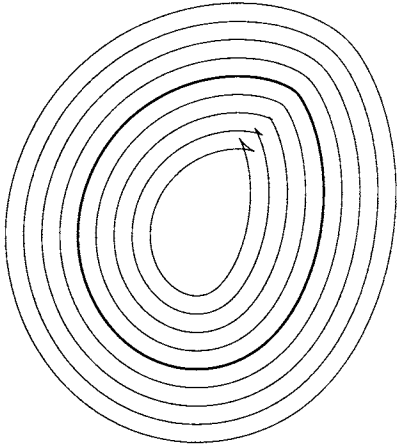


Figure 9.3. Offset curves to a cam profile obtained using trigonometric splines.

where the Cartesian coordinates satisfy [56]

$$x(\theta) = f(\theta) \cos(\theta) - f'(\theta) \sin(\theta), \quad y(\theta) = f(\theta) \sin(\theta) + f'(\theta) \cos(\theta).$$

The objective of cam design is to obtain appropriate support functions f that meet a prescribed set of requirements. Typically, f must take on specific values at given discrete angles and prescribed dwells (*i.e.*, intervals with constant displacement), and should result in a cam that is consistent with the desired behavior of vertical velocity, acceleration, and the so-called jerk.

In principle, to represent f , we could use a piecewise polynomial function that is periodic with period 2π , is sufficiently smooth, and has enough flexibility (*i.e.*, enough knots) to satisfy the various mentioned conditions. However, this would lead to a very complicated representation for the actual cam shape. This is apparent from the above formula for \mathbf{C}_f , which is a mixture of polynomial splines and trigonometric functions. In particular, these cam profiles typically do not possess piecewise rational parametrizations as are employed by CAD systems and accepted by CNC milling machines. Conversely, if one desires a (piecewise) polynomial or rational representation for \mathbf{C}_f , then the form of the displacement function f is generally far from simple, hence not appropriate for design and optimization purposes.

The mentioned shortcoming of polynomial splines has triggered a search for splines that are better suited to describe functions on the circle. Indeed, in [56] it has been shown that *trigonometric splines* (*i.e.*, functions consisting of pieces of trigonometric functions, see [69]) are an ideal tool for cam design. In fact, the formula for \mathbf{C}_f alone suggests that trigonometric functions play a natural role in representing functions on the circle.

In the context of cam design, trigonometric splines are an attractive alternative to polynomial splines. Namely, representing the support function f as a trigonometric spline leads to a piecewise rational profile curve \mathbf{C}_f . Moreover, it turns out that these curves have the remarkable property that their offsets are also piecewise rational [55]. This means that the corresponding cam profiles, as well as the cutter paths for CNC machining, both possess a standard NURBS representation.

The lesson to be learned from this example is that even though polynomial splines can be used to represent functions on the circle, it is more “natural” and advantageous to employ a function space that “matches” the circle, *i.e.*, one that is derived from trigonometric polynomials. ■

The next example shows that the type of space we use may not be just a question of convenience. In some cases it may also be a matter of necessity.

Example 2. Let us now consider the more difficult problem of constructing smooth functions on the sphere \mathbf{S} in \mathbb{R}^3 . The most obvious approach is to map the sphere onto the plane, using spherical coordinates, then employing classical bivariate tensor-product splines. To state the problem more precisely, suppose that \mathbf{S} is mapped onto the longitude-latitude rectangle

$$\Omega := \{(\theta, \phi) \mid 0 \leq \theta \leq 2\pi, -\pi/2 \leq \phi \leq \pi/2\}$$

and that $f : \Omega \rightarrow \mathbb{R}$ is a given function. We can view f as a function on \mathbf{S} , provided that it is periodic, that is

$$f(0, \phi) = f(2\pi, \phi), \quad -\pi/2 \leq \phi \leq \pi/2, \quad (9.5)$$

and constant at the poles *i.e.*,

$$f(\theta, -\pi/2) = f_S, \quad f(\theta, \pi/2) = f_N, \quad 0 \leq \theta \leq 2\pi, \quad (9.6)$$

where f_S and f_N are the values of f at the south and north poles, respectively.

To represent/approximate spherical functions, it is natural to consider the space of tensor-product polynomial splines, *i.e.*, the space of functions of the form

$$f(\theta, \phi) = \sum_{i=1}^M \sum_{j=1}^N c_{ij} \Theta_i(\theta) \Phi_j(\phi), \quad (9.7)$$

where Θ_i and Φ_j are univariate B-splines associated with a knot partition of the θ - and ϕ -axes, respectively. While functions of this form can be made arbitrarily smooth (on Ω) by choosing the degree of the spline sufficiently high, this will not automatically guarantee that f is smooth as a function on \mathbf{S} , even if it satisfies both (9.5) and (9.6). In fact, it was shown in [17,35] (see also [74]) that to obtain a smooth (C^1 or higher) function, the following set of equations must hold for the partial derivatives f_θ and f_ϕ :

$$f_\theta(0, \phi) = f_\theta(2\pi, \phi), \quad -\pi/2 \leq \phi \leq \pi/2,$$

and

$$f_\phi(\theta, -\pi/2) = A_S \cos \theta + B_S \sin \theta, \quad f_\phi(\theta, \pi/2) = A_N \cos \theta + B_N \sin \theta, \quad 0 \leq \theta \leq 2\pi,$$

where $A_S, B_S, A_N,$ and B_N are constants. These equations express the condition that f is C^1 -continuous along the prime meridian $\theta = 0$ and also at the poles, which is equivalent to continuity of the tangent plane of f on \mathbf{S} .

It can be seen that, unless the function f is flat at the poles, i.e., unless $A_S = B_S = A_N = B_N = 0$, no tensor-product spline of the form (9.7) can be smooth at the poles. This follows from the fact that the right-hand sides of the last two of the above equalities involve trigonometric polynomials. Thus, no matter how high the degree or fine the knot partition of the longitude-latitude axes, piecewise polynomials are inherently incompatible with the sphere, at least in the above-described setting.

Consequently, to obtain globally smooth spherical functions, we must abandon the idea of using the classical B-splines. Like in our previous example, it is a good idea to utilize trigonometric splines. More precisely, it was shown in [71] that by replacing the B-splines Θ_i in (9.7) with their trigonometric counterparts, it is indeed possible to construct smooth spherical functions of the form (9.7). ■

The above two examples have demonstrated that the straightforward approach to representing functions on \mathbf{S} is not necessarily optimal. Even if \mathbf{S} is simple enough that it can be mapped onto an interval or a planar region, it may still be useful to “custom-design” a spline space that is compatible with the given surface. As we have seen, in general this will lead to function spaces that are *non-polynomial* in nature.

A sceptical reader might still argue that there is always the possibility to use standard parametric surface patches, which are piecewise polynomial, to reconstruct f . For instance, in Example 2 we could use parametric surface patches to model the surface \mathbf{S}' in (9.3) and then obtain the corresponding scalar function as $f(\mathbf{s}) = (\mathbf{S}'(\mathbf{s}) - \mathbf{s}) \cdot \mathbf{n}_{\mathbf{s}}, \mathbf{s} \in \mathbf{S}$. While this is true in principle, there are several difficulties associated with this approach. First, using parametric surfaces requires working with vector-valued functions, as opposed to the simpler scalar functions. Second, constructing smooth composite parametric surfaces is a highly non-trivial task since the smoothness conditions between adjoining surface patches are more difficult to impose than in the scalar case. Third, the construction of the parametric surface must guarantee that \mathbf{S}' is star-like, i.e., such that every half-line emanating from the origin intersects \mathbf{S}' only once, for otherwise f would not be well defined. Lastly, once \mathbf{S}' has been constructed, to recover $f(\mathbf{s})$ for a given point \mathbf{s} on the sphere, it is necessary to compute the value $\mathbf{S}'(\mathbf{s})$. This is a difficult task since it requires solving a set of algebraic equations.

The basic question that arises in the context of (9.2) is how to construct spline spaces of functions on \mathbf{S} that resemble the usual spaces of piecewise polynomials in the plane (if this is indeed possible). Another important question is whether the well-established data fitting and modeling techniques for functions in the plane can be extended to the general setting of arbitrary surfaces. The hope is that many of the usual methods would carry over to this general situation without too much extra effort.

To address these questions, we will primarily focus our attention in Section 9.2 on the issue of constructing spline spaces on \mathbf{S} . Alternative approaches to this problem will be listed in Section 9.3.

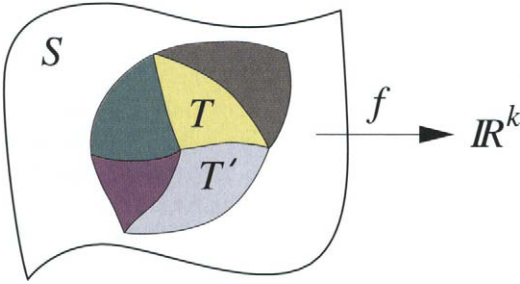


Figure 9.4. Triangulation of a surface S .

9.2. SCALAR SPLINES ON SMOOTH SURFACES

Having looked at examples of splines on specific manifolds, let us now consider the case of general surfaces. We will first define the context of our setting. In this section the surface S will be C^∞ -smooth in the usual sense of differential geometry. This is because our main interest will be in *smooth* splines on S , which would be an awkward requirement if S itself were not sufficiently smooth. Of course, for all practical purposes the infinite differentiability condition can be relaxed to match the order of smoothness of f . Also, we will restrict ourselves here to the case of scalar-valued functions f (i.e., $k = 1$ in (9.2)).

Splines are usually obtained by dividing up the domain of their definition, S in our case, into disjoint subsets. The most familiar means of partitioning a given domain is to triangulate it. If the domain is planar, the resulting spline space is the well-known space of piecewise polynomials on triangulations. While there exist a great variety of bivariate splines, corresponding to many types of grids and polygonal partitions, piecewise polynomials on planar triangulations are the most universal since all other types can be viewed and represented as splines on triangulations. Therefore, to address the problem on general surfaces S , we will also assume that the sought-for spline space will correspond to a triangulation of S .

As usual, a triangulation of S is a collection Δ of geodesic triangles on S , whose interiors are disjoint and such that the union of all triangles in Δ is S , i.e., $S = \bigcup_{T \in \Delta} T$. Here, a geodesic triangle T is a closed subset of S , homeomorphic to a planar triangle, whose boundary consists of three geodesic segments on S connecting a triple of points in T , called the vertices of T . These three geodesics are the edges of T . The vertices of T will be denoted by $V(T)$ and the edges by $E(T)$. The edges will be indexed by $V(T)$ such that $e_v \in E(T)$ will denote the edge of T opposite to $v \in V(T)$.

Perhaps the most popular way of representing splines on triangulations is by means of the Bernstein-Bézier formalism. Bernstein-Bézier techniques are extremely useful tools for constructing piecewise polynomial surfaces over triangulated planar domains. As is demonstrated in many chapters of the book (see Chapter 4), they play an important role in CAGD, data fitting, computer vision, and elsewhere (see [22,39]). To define analogs of Bernstein-Bézier techniques associated with S , we need to find finite-dimensional spaces of functions that play the role of ordinary polynomials in the plane. This in turn leads us to

the question of defining an appropriate counterpart of the space of linear functions, since then higher-degree “polynomials” will be obtained as products of the “linear” functions. To be able to express such generalized polynomials in a Bernstein-Bézier-like form, in every triangle $T \in \Delta$, we also need to find analogs of the well-known *barycentric coordinates*. Such generalized coordinates will be denoted as

$$b_{\mathbf{v}}^T : T \rightarrow \mathbb{R}, \quad \mathbf{v} \in V(T), T \in \Delta.$$

This reduces the problem of an appropriate definition of “piecewise polynomials” on Δ to finding a reasonable way to define the functions $b_{\mathbf{v}}^T$. To answer this query, let us list some of the main properties associated with the classical Bernstein-Bézier formalism:

- (1) Non-negativity and partition of unity of Bernstein polynomials associated with any given triangle;
- (2) The convex hull property;
- (3) Reproduction of algebraic polynomials;
- (4) Affine invariance *i.e.*, Bézier coefficients will not change after transforming the Euclidean plane by an affine transformation;
- (5) The possibility of obtaining arbitrarily smooth piecewise polynomials on any triangulation;
- (6) Smoothness between adjacent Bézier triangles can be expressed in terms of local Bézier coefficients, corresponding to these triangles.

While this may seem somewhat surprising, in the context of splines on general surfaces the most fundamental of the above items turn out to be properties (5) and (6). That is, spline spaces on general surfaces should be such that they can be used to build functions that are arbitrarily *globally* smooth (provided the degree is sufficiently high) and yet these spaces should be flexible enough so that it is possible to construct *local* methods of reconstruction. Another important property of the classical piecewise polynomials, that should be maintained in the general case, is that they are generated in a simple way from linear functions on each triangle. In our setting, the analog of the space of linear functions will be the three-dimensional space \mathcal{L}^T , which is the linear span of the barycentric coordinates, *i.e.*, $\mathcal{L}^T := \text{span}\{b_{\mathbf{v}}^T, \mathbf{v} \in V(T)\}$.

It is not difficult to see that the spaces \mathcal{L}^T cannot be quite arbitrary. In particular, the barycentric coordinates should interpolate at the vertices, and in fact

$$b_{\mathbf{v}}^T(\mathbf{v}) = 1 \quad \text{and} \quad b_{\mathbf{v}}^T(e_{\mathbf{v}}) = 0, \quad \mathbf{v} \in V(T). \tag{9.8}$$

This implies that \mathcal{L}^T restricted to any of the three edges of T is two dimensional, rather than three dimensional. In addition, a requirement for global continuity (C^0) of a spline function in the “linear” spline space

$$S_1^{-1}(\Delta) := \{f : \mathbf{S} \rightarrow \mathbb{R}, f|_T \in \mathcal{L}^T, T \in \Delta\},$$

is that the spaces $\mathcal{L}^T|_e$ and $\mathcal{L}^{T'}|_e$, corresponding to any pair of neighboring triangles T and T' , sharing a common edge $e = T \cap T'$, should be identical, *i.e.*,

$$\mathcal{L}^T|_e = \mathcal{L}^{T'}|_e. \quad (9.9)$$

Otherwise it would be impossible to join two neighboring triangular patches in this (or any induced higher-degree) spline space continuously, let alone smoothly.

Assuming that all spaces $\mathcal{L}^T, T \in \Delta$, satisfy (9.8) and (9.9), we can define the space of continuous splines of degree n on \mathbf{S} as

$$S_n^0(\Delta) := \{f \in C^0(\mathbf{S}), f|_T \in \mathcal{P}_n^T, T \in \Delta\},$$

where \mathcal{P}_n^T is the space of “polynomials” of degree n on $T \in \Delta$, defined as

$$\mathcal{P}_n^T := \left\{ f = \sum_{|i|=n} c_i^T B_i^{n,T}, c_i^T \in \mathbb{R} \right\},$$

and where $B_i^{n,T}$ are the “Bernstein polynomials” or *B-polynomials*

$$B_i^{n,T}(\mathbf{s}) := \frac{n!}{\prod_{\mathbf{v} \in V(T)} (i_{\mathbf{v}}!)} \prod_{\mathbf{v} \in V(T)} (b_{\mathbf{v}}^T(\mathbf{s}))^{i_{\mathbf{v}}}, \quad \mathbf{s} \in T,$$

for every multi-index $i = (i_{\mathbf{v}})_{\mathbf{v} \in V(T)}$ with $|i| := \sum_{\mathbf{v} \in V(T)} i_{\mathbf{v}} = n$. Note that the B-polynomials reduce to the ordinary (algebraic) polynomials if $b_{\mathbf{v}}^T$ are the usual barycentric coordinates associated with a planar triangle T . Also note that

$$\sum_{|i|=n} B_i^{n,T} = \left(\sum_{\mathbf{v} \in V(T)} b_{\mathbf{v}}^T \right)^n,$$

and that the B-polynomials are linearly independent (hence form a basis for \mathcal{P}_n^T), as a consequence of (9.8).

Do conditions (9.8) and (9.9) guarantee that the spaces $S_n^0(\Delta), n > 1$, will automatically contain nontrivial smooth functions, C^1 say? It should not come as a surprise that the answer is negative. This can be seen in the planar case $\mathbf{S} = \mathbb{R}^2$ if we choose a set of non-standard functions $b_{\mathbf{v}}^T$. Intuitively, the reason for this is that conditions (9.8) and (9.9) do not enforce compatibility of derivatives of these functions across the edges of the triangulation.

Example 3. In Example 2, the sphere \mathbf{S} was parametrized using the spherical coordinates. An alternative is to parametrize \mathbf{S} by the standard octahedron with vertices $\mathbf{v}_1 = (0, 0, 1), \mathbf{v}_2 = (1, 0, 0), \mathbf{v}_3 = (0, 1, 0), \mathbf{v}_4 = (-1, 0, 0), \mathbf{v}_5 = (0, -1, 0), \mathbf{v}_6 = (0, 0, -1)$ (or by any polyhedron inscribed in \mathbf{S}). The vertices of the octahedron give rise to a triangulation Δ of the sphere, consisting of eight spherical triangles. Consider two adjacent triangles in Δ , say T and T' , determined by their vertices $V(T) = \{\mathbf{v}_1, \mathbf{v}_3, \mathbf{v}_2\}$ and $V(T') = \{\mathbf{v}_1, \mathbf{v}_3, \mathbf{v}_4\}$, respectively. Let us now define the functions $b_{\mathbf{v}}^T, b_{\mathbf{v}}^{T'}$ for the two triangles as the usual barycentric coordinates associated with the *planar* triangles with vertices $V(T)$ and $V(T')$. One can check that this leads to

$$b_{\mathbf{v}_1}^T(\mathbf{s}) = \frac{z}{x+y+z}, \quad b_{\mathbf{v}_3}^T(\mathbf{s}) = \frac{y}{x+y+z}, \quad b_{\mathbf{v}_2}^T(\mathbf{s}) = \frac{x}{x+y+z},$$

where $\mathbf{s} = (x, y, z) \in T$ and

$$b_{\mathbf{v}_1}^{T'}(\mathbf{s}) = \frac{z}{-x + y + z}, \quad b_{\mathbf{v}_3}^{T'}(\mathbf{s}) = \frac{y}{-x + y + z}, \quad b_{\mathbf{v}_4}^{T'}(\mathbf{s}) = \frac{-x}{-x + y + z},$$

where $\mathbf{s} = (x, y, z) \in T'$. The reader is invited to verify that (9.8) and (9.9) are satisfied in this case. However, given a function $f \in \mathcal{P}_2^T$, it may be impossible to find an $f' \in \mathcal{P}_2^{T'}$ such that the two functions join smoothly (C^1) across the common edge $\mathbf{v}_1\mathbf{v}_3$ (for example, take $f = (b_{\mathbf{v}_1}^T)^2$). ■

The following proposition gives a necessary condition for a smooth join between neighboring triangles [54].

Proposition 1. Let $T, T' \in \Delta$ be two adjacent triangles on \mathbf{S} such that $T \cup T'$ is homeomorphic to a disk in \mathbb{R}^2 . Let $b_{\mathbf{v}}^T \in C^\infty(T)$, $\mathbf{v} \in V(T)$, and $b_{\mathbf{v}}^{T'} \in C^\infty(T')$, $\mathbf{v} \in V(T')$. Suppose that for every n and every $f \in \mathcal{P}_n^T$ there exists an $f' \in \mathcal{P}_n^{T'}$ such that f and f' join with C^{n-1} continuity along the common edge $T \cap T'$. Then each $b_{\mathbf{v}}^T$, $\mathbf{v} \in T$, can be extended as a C^∞ function on $T \cup T'$ which, when restricted to T' , belongs to $\mathcal{L}^{T'}$. Equivalently, there exists a three-dimensional space \mathcal{L} of C^∞ functions on $T \cup T'$ such that $\mathcal{L}|_T = \mathcal{L}^T$ and $\mathcal{L}|_{T'} = \mathcal{L}^{T'}$.

A consequence of this result is that the choice of the functions $b_{\mathbf{v}}^T$ is greatly restricted. In particular, the proposition says that all barycentric coordinates $b_{\mathbf{v}}^T$ corresponding to any given triangle T , can be smoothly extended across edges of T to neighboring triangles and hence all spaces \mathcal{L}^T must *locally* belong to a *single* three-dimensional space of “linear functions” \mathcal{L} . The following example shows that such a space \mathcal{L} might not contain *globally* continuous functions, *i.e.*, for some surfaces one may be able to define \mathcal{L} only locally.

Example 4. Let \mathbf{S} be the circular cylinder parametrized by

$$(\sin \phi, \cos \phi, z) \in \mathbb{R}^3, \quad \phi \in \mathbb{R}, z \in \mathbb{R},$$

and let

$$\mathcal{L} := \text{span}\{1, \phi, z\}.$$

Hence, the point on \mathbf{S} whose parameters are (ϕ, z) is assigned the value $a + b\phi + cz$, where a, b, c are real coefficients. Note that \mathcal{L} is not defined globally since there is no globally continuous function in it. However, on every “strip”

$$\{(\phi, z), \phi \in (\alpha, \beta), z \in \mathbb{R}\},$$

where $\beta - \alpha \leq 2\pi$, the space \mathcal{L} is well defined and generated by C^∞ functions.

If Δ is a geodesic triangulation of \mathbf{S} consisting of “small” triangles (*i.e.*, triangles for which the values of ϕ are in an interval of length not exceeding π), then it is not difficult to see that for every such triangle T , $\dim(\mathcal{L}^T|_e) = 2, e \in E(T)$, where $\mathcal{L}^T := \mathcal{L}|_T$. This follows from the observation that every $f \in \mathcal{L}$ vanishes along geodesics. Namely, if $f(\phi, z) = a + b\phi + cz$, where $b \neq 0$, then $f(\phi, z) = 0$ along the curve

$$\{(\sin((a + cz)/b), \cos((a + cz)/b), z), z \in \mathbb{R}\},$$

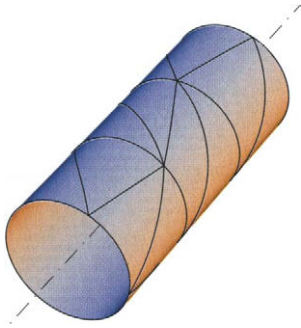


Figure 9.5. Triangulation of a cylinder.

which is a helix, hence a geodesic. Otherwise, if $b = 0$, z is a constant, which also corresponds to a geodesic on \mathbf{S} .

The above discussion shows that for every sufficiently small geodesic triangle $T \in \Delta$, one can define *cylindrical barycentric coordinates*. It turns out that many properties of these coordinates are similar to the properties of their planar counterparts. In particular, the cylindrical barycentric coordinates make it possible to define a spline space, which can be used to construct smooth splines on \mathbf{S} . Figure 9.5 shows an example of a cylindrical triangulation Δ and Figure 9.6 shows a C^1 smooth spline in the space $S_5^0(\Delta)$, corresponding to this triangulation. ■

Example 4 raises the question whether a space \mathcal{L} , or a collection of spaces \mathcal{L}^T satisfying the conditions implied by Proposition 1, always exists on any \mathbf{S} and, if so, how can one find such spaces. A simple approach to constructing \mathcal{L} is to take advantage of our knowledge of \mathbf{S} , *i.e.*, the implicit assumption here that we can evaluate \mathbf{S} at any point. In particular, we can define \mathcal{L} as the span of the three functions $x(\mathbf{s}), y(\mathbf{s}), z(\mathbf{s})$, the Cartesian coordinates of \mathbf{s} *i.e.*, $\mathbf{s} = (x(\mathbf{s}), y(\mathbf{s}), z(\mathbf{s}))$, $\mathbf{s} \in \mathbf{S}$. This works well in the important special case of a sphere in \mathbb{R}^3 .

Example 5. The problem of constructing spherical analogs of Bézier triangles has an interesting history. It has received ample attention in the spline literature for the obvious reason that splines on a sphere have many potential applications in geosciences, including meteorology, geophysics, and geodesy. Researchers had been searching for many years for appropriate spherical analogs of Bézier methods, but were hampered by the difficulty of defining *spherical barycentric coordinates*. For example, several candidates for such coordinates have been introduced in [13,14,41], but they lacked many key properties of the planar coordinates. As it turned out, the mentioned attempts were destined to be unsuccessful for the simple reason that they all insisted on the partition of unity property, which is well known for the classical barycentric coordinates. Namely, it was shown in [14] that spherical barycentric coordinates that sum to one and satisfy a list of other reasonable assumptions, do not exist. This negative result provided an explanation why the various earlier generalizations were unsuccessful in building smooth splines on

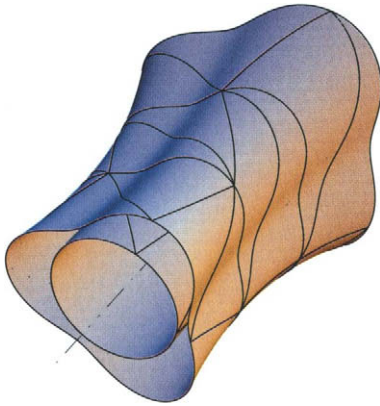


Figure 9.6. A C^1 smooth quintic spline on the cylinder.

spherical triangulations.

A breakthrough in this development came when the authors of [1] realized, by studying identities of spherical trigonometry, that there is in fact a natural way of defining barycentric coordinates for spherical triangles. Let T be a spherical triangle with vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbf{S}$. Thus the edges of T are the three shortest geodesics connecting the vertices. One can define the spherical barycentric coordinates of a point $\mathbf{s} \in T$ as

$$b_{\mathbf{v}_i}^T(\mathbf{s}) := \frac{\sin \delta_i}{\sin(\delta_i + \gamma_i)}, \quad i = 1, 2, 3,$$

where δ_i is the geodesic distance (measured along the great circle passing through \mathbf{s} and \mathbf{v}_i) of \mathbf{u}_i and \mathbf{s} , and γ_i is the geodesic distance of \mathbf{s} and \mathbf{v}_i (see Figure 9.7).

Despite the inevitable fact that the above coordinates do not add up to one, it has been shown in [1] that they resemble the standard barycentric coordinates in many respects. First, the barycentric coordinates are infinitely smooth functions on T and they satisfy (9.8). The space \mathcal{L}^T , the span of the three coordinates, reduces to dimension two along the edges of T and in fact along every great circle intersecting T . More precisely, the restriction of \mathcal{L}^T to any such great circle can be shown to be the linear span of $\{\sin \alpha, \cos \alpha\}$, where α is the arclength distance measured along this circle. Another important consequence of the above definition is that the spaces \mathcal{L}^T and $\mathcal{L}^{T'}$, associated with neighboring triangles T and T' , satisfy (9.9). Furthermore, the properties of \mathcal{L}^T are consistent with those specified in Proposition 1. In particular, this space can be extended to a three-dimensional space \mathcal{L} of infinitely differentiable functions over all of \mathbf{S} . Conversely, for any triangle T the space \mathcal{L}^T is just the restriction of \mathcal{L} to T .

The space \mathcal{L} has many interesting properties that indicate that spherical barycentric coordinates, as defined here, are unique. More precisely, \mathcal{L} is the only three-dimensional space of functions on the sphere \mathbf{S} that is rotationally invariant and such that its dimension is reduced along great circles. Moreover, \mathcal{L} can be easily described using the idea suggested earlier. Namely, \mathcal{L} is precisely the span of the Cartesian coordinates $x(\mathbf{s}), y(\mathbf{s}), z(\mathbf{s})$, viewed as functions on \mathbf{S} . This is equivalent to saying that \mathcal{L} is the space of *spherical harmonics*

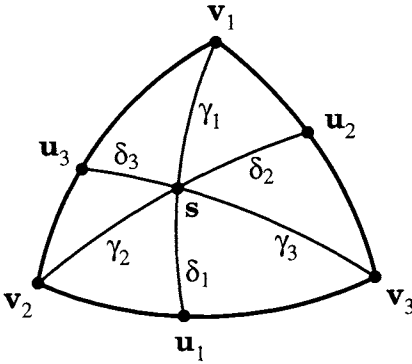


Figure 9.7. Definition of spherical barycentric coordinates.

of degree one [49].

The intimate connection between spherical harmonics and spherical barycentric coordinates is not unexpected. In retrospect, it seems quite obvious that these functions should have been considered early on as the most natural candidates for “linear functions” on the sphere. As a matter of fact, such functions, along with the corresponding barycentric coordinates defined above, had been considered some 150 years before the paper [1] appeared. The authors of that paper found out only later that their idea of defining barycentric coordinates was not new after all, since the same definition had already been given by Möbius [47].

Spherical barycentric coordinates give rise to Bernstein-Bézier-type methods, with immediate applications to a variety of problems on the sphere. Indeed, because of the close analogy with standard Bernstein-Bézier techniques, virtually all of the classical methods for piecewise polynomials on planar triangulations can be carried over to the spherical setting, and indeed to any setting where barycentric coordinates are available. A detailed treatment of some of these methods has been given in [2,3]. The spherical Bernstein-Bézier methods are also of interest in the design of surfaces, especially star-like surfaces, even though some of the geometric properties of planar Bézier methods are missing on the sphere, such as the convex hull property.

Typical scattered data interpolation/approximation methods on the sphere start with a triangulation of the sphere, for example the so-called Delaunay triangulation. We refer the reader to [69], for a survey on triangulations, and also to [11,39], for a discussion of triangulation methods on general surfaces. Methods for triangulating scattered data points on the sphere are discussed in [41,58,66]. Figures 9.9 and 9.10 show wire plots of smooth C^1 quadratic and cubic spherical splines, respectively, corresponding to (a part of) the triangulation in Figure 9.8. Details about various data-fitting methods that lead to such spherical splines are discussed in [3]. ■

Besides the sphere, the suggestion to use Cartesian coordinates to define the space \mathcal{L} makes sense, as long as the triangulation Δ of \mathbf{S} consists of triangles whose edges reduce

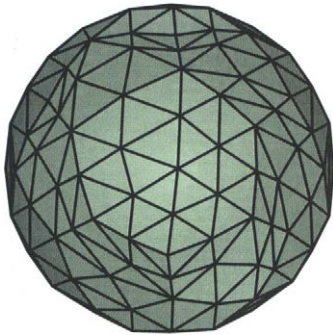


Figure 9.8. A triangulation of the sphere.

the dimension of \mathcal{L} . This is equivalent to the condition that the edges of Δ are planar and in the same plane with the origin $(0, 0, 0)$. This is acceptable for certain surfaces (for example, for star-like surfaces [3]), but is more restrictive in cases in which the assumption of coplanarity would result in severely distorted triangles (relative to the geodesic ones).

To cope with this difficulty, a possibility is to make a coordinate transformation to minimize the distortion of the triangles, *e.g.*, one could shift the origin of the Cartesian system. More generally, one could in principle take “any” three-dimensional space \mathcal{H} of smooth functions in \mathbb{R}^3 and define \mathcal{L} as the restriction of \mathcal{H} to \mathbf{S} . In the spherical case, this would lead to $\mathcal{L} = \mathcal{H}|_{\mathbf{S}}$, where $\mathcal{H} := \text{span}\{x, y, z\}$. In the planar case $\mathbf{S} = \mathbb{R}^2$, we can think of \mathcal{L} as $\mathcal{H}|_{\mathbf{S}}$, where $\mathcal{H} := \text{span}\{1, x, y\}$. In this way one can also interpret the construction of \mathcal{L} for the cylinder. In particular, we can take for any fixed $\alpha \in \mathbb{R}$,

$$\mathcal{H} := \text{span} \left\{ \arctan \left(\frac{x \cos \alpha - y \sin \alpha}{x \sin \alpha + y \cos \alpha} \right), z, 1 \right\},$$

in which case \mathcal{L} is the restriction of \mathcal{H} to the strip

$$\{(\phi, z), \alpha - \pi/2 < \phi < \alpha + \pi/2, z \in \mathbb{R}\}.$$

On the other hand, it is not clear how to choose \mathcal{H} for a general surface \mathbf{S} so as to obtain “least distorted” triangulations.

Above, we have seen two examples of non-planar surfaces for which it is possible to construct meaningful analogs of spline spaces. The characteristic property shared by both types of splines, as well as by the classical bivariate splines on planar triangulations, is that they can be generated by a three-dimensional space \mathcal{L} of “linear functions” on \mathbf{S} . This space is such that

- Functions in \mathcal{L} vanish along geodesics on \mathbf{S} *i.e.*, for every nontrivial $f \in \mathcal{L}$, the set $\mathbf{C} := \{\mathbf{s} \in \mathbf{S}, f(\mathbf{s}) = 0\}$ is a geodesic. Conversely, for every geodesic \mathbf{C} on \mathbf{S} , there exists a nonzero function $f \in \mathcal{L}$ vanishing on \mathbf{C} .

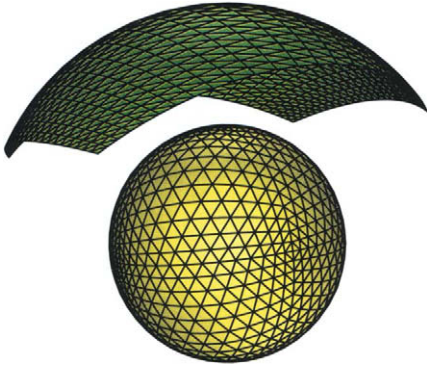


Figure 9.9. A C^1 smooth quadratic spherical spline.

- \mathcal{L} is invariant under isometric transformations of \mathbf{S} . Thus, if $I : \mathbf{S} \rightarrow \mathbf{S}$ is an isometry (e.g., a rotation, if \mathbf{S} is the sphere), then $f \circ I \in \mathcal{L}$ for all $f \in \mathcal{L}$.

It is clear that these two properties imply that \mathcal{L} is quite special and that for a given surface \mathbf{S} one cannot expect to have more than one such space. In fact, in the three mentioned cases, \mathcal{L} is uniquely determined by the above two properties. It is an intriguing open question whether there always exists a space satisfying at least the first property. This in turn is closely related to the central issue of this section of whether one can find analogs of spline spaces on general surfaces. Using the well-known Beltrami's Theorem about the existence of local geodesic mappings [18], it can be shown that \mathcal{L} can be found for surfaces of constant (Gaussian) curvature [54]. However, it is not known if one can go beyond such surfaces.

It should be said that, for all practical purposes, we need not require that \mathcal{L} strictly satisfy the mentioned conditions. The lack of a theoretical proof of the existence of \mathcal{L} should not prevent us from being able to establish useful spline spaces on \mathbf{S} . For example, for a given fixed geodesic triangulation Δ , one could use a space \mathcal{L} which has a reduced dimension along all of its edges, but which does not necessarily have the property that *all* functions in \mathcal{L} vanish along geodesics. Such space \mathcal{L} will still allow the construction of barycentric coordinates for all triangles of Δ , hence also the construction of a spline space corresponding to Δ .

Another possibility is to relax the assumption that the triangles in Δ are strictly geodesic. In fact, parametric surfaces \mathbf{S} composed of triangular Bézier patches are already equipped with a natural triangulation Δ in which every triangle corresponds to a Bézier patch. Such a triangulation is in general *not* geodesic. In this situation there is a particularly simple way to choose the barycentric functions $b_{\mathbf{v}}^T$. Recall that Example 3

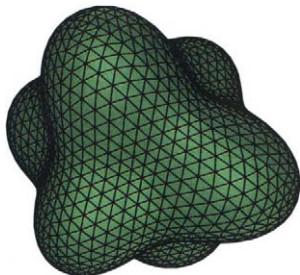


Figure 9.10. A C^1 smooth cubic spherical spline.

gives reasons why it is in general not a good idea to use the standard barycentric coordinates corresponding to the triangular facets of the piecewise linear interpolant to the vertices of Δ . This is because this will in general not permit us to design smooth splines over \mathbf{S} . However, it turns out that this approach *will* work if \mathbf{S} is a composite surface consisting of triangular polynomial patches. Namely, the fact that neighboring triangular patches are joined smoothly guarantees that the ordinary *planar* barycentric coordinates corresponding to these triangles are compatible, in the sense of Proposition 1. Thus in any given triangle these coordinates can be extended to the neighboring triangles as smooth functions (where the degree of smoothness will depend on the smoothness of \mathbf{S}).

We have seen that the success or failure of constructing spline spaces on general surfaces hinges upon the existence of barycentric coordinates. It is also clear from our examples, that even for simple surfaces such constructions may be far from trivial. Still, the discussed framework of splines on surfaces offers many benefits, compared to other existing reconstruction methods. The main argument supporting this claim is that the Bernstein-Bézier formalism for splines on surfaces is the same for all surfaces. That is, it is essentially irrelevant whether we work on the sphere, in the plane, or on any other surface. As a consequence, we can use the same algorithms for splines on all surfaces, as long as we use a correct procedure to compute the barycentric coordinates (which do depend on the type of the surface).

Example 6. To illustrate the above point, suppose that $T, T' \in \Delta$ are two adjacent triangles on \mathbf{S} , with vertices $V(T) = \{\mathbf{v}_1, \mathbf{v}_3, \mathbf{v}_2\}$ and $V(T') = \{\mathbf{v}_1, \mathbf{v}_3, \mathbf{v}_4\}$. Let f and f' be two “polynomials” of degree n of the form

$$f = \sum_{|i|=n} c_i^T B_i^{n,T}, \quad f' = \sum_{|i|=n} c_i^{T'} B_i^{n,T'},$$

where $c_i^T, c_i^{T'} \in \mathbb{R}$ are given Bézier coefficients. Then f and f' join continuously along

the edge $\mathbf{v}_1\mathbf{v}_3$ if

$$c_{i_1, i_3, 0}^{T'} = c_{i_1, i_3, 0}^T, \quad i_1 + i_3 = n,$$

and they join with C^1 continuity if and only if

$$c_{i_1, i_3, 1}^{T'} = c_{i_1+1, i_3, 0}^T b_{\mathbf{v}_1}^T(\mathbf{v}_4) + c_{i_1, i_3+1, 0}^T b_{\mathbf{v}_3}^T(\mathbf{v}_4) + c_{i_1, i_3, 1}^T b_{\mathbf{v}_2}^T(\mathbf{v}_4), \quad i_1 + i_3 = n - 1,$$

where we used the abbreviations $i_j := i_{\mathbf{v}_j}$, $j = 1, 2, 3$. Moreover, the values $b_{\mathbf{v}_j}^T(\mathbf{v}_4)$ are obtained by first extending $b_{\mathbf{v}_j}^T$ as smooth functions to $T \cup T'$, and then evaluating them at \mathbf{v}_4 . This extension is possible as long as the barycentric coordinates are compatible in the sense of Proposition 1. ■

The above conditions for a smooth join between two generalized Bézier triangles can be immediately seen to be formally *identical* to the corresponding conditions in the planar setting. This explains why we have not addressed in this section the actual reconstruction problem (e.g., interpolation or approximation) using splines on surfaces. The reason is that the same methods known in the plane can be transformed almost automatically to any surface. Various extensions of such planar methods to the sphere are discussed in [3] and it is indeed clear from that paper that, except for a few details, the planar methods carry over to the spherical setting, and indeed to the setting of any smooth surface \mathbf{S} .

9.3. ALTERNATIVE METHODS FOR FUNCTIONS ON SURFACES

In this chapter our focus was mainly on discussing splines on surfaces, *i.e.*, analogs of piecewise polynomials on planar triangulations. However, this should not leave the reader with the impression that there are not other methods that have been used successfully in data-fitting and reconstruction problems on surfaces. Therefore, in this section we will give a brief description of other available methods and provide references for further study. This is also a good place to mention other surveys that have been written on the topic of splines on surfaces, such as Chapter 9.7 in [39] and [8].

9.3.1. Discrete surfaces

Having discussed the idealized problem in which \mathbf{S} was a smooth surface, it should be mentioned that in some applications surfaces come as discrete collections of points. For example, \mathbf{S} and f could be sampled at a set of discrete points $\mathbf{s}_i \in \mathbb{R}^3$, where the \mathbf{s}_i could represent the triple (latitude, longitude, altitude), determining a physical location on the surface of the earth and f_i could be the air pressure at \mathbf{s}_i . Before we can find an approximation to f using the setting described in the previous section, we first need a preprocessing step for reconstructing the surface \mathbf{S} . We refer the reader to Chapter 26 in this book and also to [7, 11, 39, 46, 75], for an overview of and references on several methods for surface reconstruction from scattered data points in \mathbb{R}^3 .

9.3.2. Radial basis functions

An increasingly popular class of methods for interpolation and approximation of scattered data are those based on *radial basis functions* (RBF), *i.e.*, functions that are radially symmetric. The reason why such methods are often preferred to other techniques is that these methods are *meshless*. This means that to interpolate a set of functional data associated

with scattered points on a surface it is not necessary to create a triangulation, or any other type of mesh that connects the given data sites, prior to the actual reconstruction phase.

Perhaps the first types of radial basis functions used in the context of interpolation of functions on surfaces were the *Hardy multiquadrics*, introduced for application in geophysics in [36–38] (see also [8,21,26,62]). The advantage of using radial basis functions over other methods in the setting of functions on surfaces is that a given function is approximated and/or interpolated by a linear combination of translates of a *single* RBF, which gives a very simple representation of the resulting reconstruction.

Many papers have been written on the use of RBFs in the spherical setting, see [23,24,62,72], and also the survey [25]. Recently, RBFs on the sphere have been employed not only in the context of interpolation but also the context of solving differential equations on the sphere [48]. There are also several papers on Lagrange and Hermite interpolation on general Riemannian manifolds [19,20,50], which are of interest for functions on surfaces of arbitrary topology.

Methods based on RBFs generally suffer from poor conditioning of the linear systems that arise in interpolating large sets of data. However, lately there has been considerable progress on compactly supported RBFs (see [25]), which have the potential to reduce this shortcoming of RBFs.

9.3.3. Variational methods

A well-established approach to constructing bivariate functions interpolating scattered data in the plane is to use the so-called variational methods, which are based on minimizing an “energy functional” subject to given interpolation conditions. Strictly speaking, variational methods belong to the previous subsection since the kernels associated with such extremal problems are also radially symmetric. The methods have been frequently used in geosciences, see [29–31,33,76–78]. Another popular method, often applied in meteorology, is the classical *spectral method*. This method uses spherical harmonics of high degrees to approximate functions defined on the sphere [44].

9.3.4. Distance-weighting methods

This class of methods, sometimes also called Shepard-type methods, uses a weighted combination of function values that are being interpolated on the surface S , to reconstruct an unknown function from discrete data. The weights are chosen so as to decrease the influence of a particular data point on the surface as we move away from this point. In this way points that are far from a given region on the surface will have little or no effect on the reconstructed function in that region. For details on how to choose the weight functions, we refer the reader to [11,12] and Chapter 9.7 of [39].

9.3.5. Transfinite methods

Many techniques for defining functions on triangulated (or otherwise partitioned) surfaces are based on the idea of transfinite interpolation. This method starts with the construction of a network of smooth functions defined on the edges of the partition and then uses transfinite triangular interpolants to interpolate the network of curves by a smooth function on the domain surface S . The papers [11,10,40,41,58,66,67] describe a construction

of such transfinite triangular interpolants for the sphere, whereas [61] can be applied to arbitrary surfaces that are at least C^2 differentiable.

9.3.6. Implicit methods

In [5–7], *implicit* Bernstein-Bézier patches (also called A-patches) are used to simultaneously reconstruct the surface \mathbf{S} (given as a set of unorganized points) and the unknown function f , whose values at these points are given. An advantage of this method is that it does not require any assumption on the convexity and/or differentiability of \mathbf{S} and that it can handle oriented manifolds of unrestricted topological types. On the other hand, the drawback of using implicit surface patches is that they require essentially trivariate Bernstein-Bézier techniques, which are harder to deal with than bivariate ones.

9.3.7. Other types of splines

Besides splines on triangulations of a surface \mathbf{S} , various other types of splines can be constructed. For example, the papers [52,60] give a construction of *spherical simplex splines*. A recent introduction on simplex splines is [53]. The paper [42] uses spherical splines on triangulations to define hybrid cubic Bézier patches, which are analogs of the classical rational Bézier patches. The author is not aware of similar methods for other types of surfaces (except planar).

9.3.8. Multiresolution methods

This overview would be incomplete if we did not mention the possibility of using *subdivision* or *wavelet*-type methods in connection with the reconstruction of functions on surfaces. Subdivision methods are a powerful means of constructing surfaces and hence one would expect that they might also lend themselves to problems in the area of “surfaces on surfaces”. There is no question that wavelet and subdivision techniques are becoming increasingly popular in many application areas, ranging from signal and image processing to CAGD and computer animation. That said, there do not seem to exist many methods based on subdivision and/or wavelets designed specifically to deal with the problem of functions on surfaces. Notable exceptions are the various existing constructions of wavelets on the sphere (see [16,32,33,43,51,68]) and the paper [73], where wavelets are constructed on general surfaces.

9.3.9. Visualization of surfaces on surfaces

Although visualization is not addressed in this chapter, it is important to stress that a good visualization of the reconstructed/modeled surfaces and functions is essential in the context of functions on surfaces since, as we have pointed out earlier, these surfaces can be viewed as being imbedded in a higher-dimensional space. Therefore, an appropriate technique to display the results of the reconstruction can significantly enhance our understanding of the behavior of the reconstructed functions and surfaces. While visualizing a surface in the four-dimensional space is very difficult, there are ways to display them by a judicious use of interactive color computer graphics. The topic of visualization of such surfaces is discussed at length in many publications, including [4,9,28,57,59,63–65].

REFERENCES

1. P. Alfeld, M. Neamtu, and L.L. Schumaker. Bernstein-Bézier polynomials on spheres, and sphere-like surfaces. *Computer Aided Geometric Design*, 13:333-349, 1996.
2. P. Alfeld, M. Neamtu, and L.L. Schumaker. Dimension and local bases of homogeneous spline spaces. *SIAM J. Math. Anal.*, 27:1482-1501, 1996.
3. P. Alfeld, M. Neamtu, and L.L. Schumaker. Fitting scattered data on sphere-like surfaces using spherical splines. *J. Comp. Appl. Math.*, 73:5-43, 1996.
4. C. Bajaj and G. Xu. Modeling and visualization of scattered function data on curved surfaces. In J. Chen, N. Thalmann, Z. Tang, and D. Thalmann, editors, *Fundamentals of Computer Graphics*, pages 19-29. World Scientific Publishing Co., 1994.
5. C. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. *Computer Graphics*, Annual Conference Series, *Proceedings of SIGGRAPH 95*, 109-118, ACM SIGGRAPH, 1995.
6. C. Bajaj, F. Bernardini, and G. Xu. Adaptive reconstruction of surfaces and scalar fields from dense scattered trivariate data. Computer Science Technical Report, TR 95-028, Purdue University, 1995.
7. C. Bajaj, F. Bernardini, and G. Xu. Reconstruction of surfaces and surfaces-on-surfaces from unorganized weighted points. *Algorithmica*, 19:243-261, 1997.
8. R.E. Barnhill and T.A. Foley. Methods for constructing surfaces on surfaces. In H. Hagen and D. Roller, editors, *Geometric Modeling, Methods and Applications*, pages 1-15. Springer, Berlin, 1991.
9. R.E. Barnhill, G.T. Makatura, and S.E. Stead. A new look at higher dimensional surfaces through computer graphics. In G.E. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 123-130. SIAM, Philadelphia, 1987.
10. R.E. Barnhill, K. Opitz, and H. Pottmann. Fat surfaces: a trivariate approach to triangle-based interpolation on surfaces. *Computer Aided Geometric Design*, 9:365-378, 1992.
11. R.E. Barnhill and H.S. Ou. Surfaces defined on surfaces. *Computer Aided Geometric Design*, 7:323-336, 1990.
12. R.E. Barnhill, B.R. Piper, and K.L. Rescorla. Interpolation to arbitrary data on a surface. In G.E. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 281-290. SIAM, Philadelphia, 1987.
13. J.R. Baumgardner and P. Frederickson. Icosahedral discretization of the two-sphere. *SIAM J. Numer. Anal.*, 22:1107-1115, 1985.
14. J.L. Brown and A.J. Worsey. Problems with defining barycentric coordinates for the sphere. *Mathematical Modelling and Numerical Analysis*, 26:37-49, 1992.
15. G.L. Browning, J.J. Hack, and P.N. Swarztrauber. A comparison of three numerical methods for solving differential equations on the sphere. *Mon. Weather Rev.*, 117:1058-1075, 1989.
16. S. Dahlke, W. Dahmen, I. Weinreich, and E. Schmitt. Multiresolution analysis and wavelets on S^2 and S^3 . *Numer. Funct. Anal. Optimiz.*, 16:19-41, 1995.
17. P. Dierckx. Algorithms for smoothing data on the sphere with tensor product splines. *Computing*, 32:319-342, 1984.
18. M.P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, New

- Jersey, 1976.
19. N. Dyn, F.J. Narcowich, and J.D. Ward. A framework for interpolation and approximation on Riemannian manifolds. In *Approximation Theory and Optimization*, (Cambridge, 1996), pages 133–144. Cambridge Univ. Press, Cambridge, 1997.
 20. N. Dyn, F.J. Narcowich, and J.D. Ward. Variational principles and Sobolev-type estimates for generalized interpolation on a Riemannian manifold. *Constr. Approx.*, 15:175–208, 1999.
 21. M. Eck. MQ-curves are curves in tension. In *Mathematical Methods in Computer Aided Geometric Design*, II (Biri, 1991), pages 217–228, Academic Press, Boston, MA, 1992.
 22. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*, 4th Edition. Academic Press, Boston, 1996.
 23. G.E. Fasshauer. Adaptive least squares fitting with radial basis functions on the sphere. In M. Dæhlen, T. Lyche, L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, pages 141–150. Vanderbilt University Press, Nashville, 1995.
 24. G.E. Fasshauer. Hermite interpolation with radial basis functions on spheres. *Adv. Comput. Math.*, 10:81–96, 1999.
 25. G.E. Fasshauer and L.L. Schumaker. Scattered data fitting on the sphere. In M. Dæhlen, T. Lyche, L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces II*, pages 117–166. Vanderbilt Univ. Press, Nashville, 1998.
 26. T.A. Foley. Interpolation to scattered data on a spherical domain. In J.C. Mason and M.G. Cox, editors, *Algorithms for Approximation II*, pages 303–310. Chapman & Hall, London, 1990.
 27. T.A. Foley, D.A. Lane, G.M. Nielson, R. Franke, and H. Hagen. Interpolation of scattered data on closed surfaces. *Computer Aided Geometric Design*, 7:303–312, 1989.
 28. T.A. Foley, D.A. Lane, G.M. Nielson, and R. Ramaraj. Visualizing functions over a sphere. *IEEE Comp. Graphics & Appl.*, 10:32–40, 1990.
 29. W. Freeden. On spherical spline interpolation and approximation. *Math. Meth. Appl. Sci.*, 3:551–575, 1981.
 30. W. Freeden. Spherical spline interpolation-basic theory and computational aspects. *J. Comp. Appl. Math.*, 11:367–375, 1985.
 31. W. Freeden, T. Gervens, and M. Schreiner. *Constructive Approximation on the Sphere With Applications to Geomathematics*. Oxford University Press, Oxford, 1998.
 32. W. Freeden and M. Schreiner. Orthogonal and nonorthogonal multiresolution analysis, scale discrete and exact fully discrete wavelet transform on the sphere. *Constr. Approx.*, 14:493–515, 1998.
 33. W. Freeden, M. Schreiner, and R. Franke. A survey on spherical spline approximation. *Surveys Math. Indust.*, 7:29–85, 1997.
 34. F.X. Giraldo. Lagrange-Galerkin methods on spherical geodesic grids: the shallow water equations. *J. Comput. Phys.*, 160:336–368, 2000.
 35. R.H.J. Gmelig Meyling and P. Pfluger. B-spline approximation of a closed surface. *IMA J. Numer. Anal.*, 7:73–96, 1987.
 36. R.L. Hardy. Multiquadric equations of topography and other irregular surfaces. *J. Geophysical Res.*, 76:1905–1915, 1971.

37. R.L. Hardy and W.M. Göpfert. Least squares prediction of gravity anomalies, geoidal undulations, and deflection of the vertical with multiquadric harmonic functions. *Geophys. Res. Letters*, 2:423–426, 1981.
38. R.L. Hardy and S.A. Nelson. A multiquadric-biharmonic representation and approximation of disturbing potential. *Geophys. Res. Letters*, 13:18–21, 1986.
39. J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Design*, AK Peters, Boston, 1993.
40. C.L. Lawson. Subroutines for C^1 surface interpolation to data defined over the surface of a sphere. *Rpt. 487*, JPL, Cal. Tech, 1982.
41. C.L. Lawson. C^1 surface interpolation for scattered data on a sphere. *Rocky Mountain J. Math.*, 14:177–202, 1984.
42. X. Liu and L.L. Schumaker. Hybrid cubic Bézier patches on spheres and sphere-like surfaces. *J. Comput. Appl. Math.*, 73:157–172, 1996.
43. T. Lyche and L.L. Schumaker. A multiresolution tensor spline method for fitting functions on the sphere. *SIAM J. Sci. Comp.*, 22:724–747, 2000.
44. B. Machenhauer. The spectral method. In *Numerical Methods Used in Atmospheric Models*, vol. II of GARP Pub. Ser. No. 17. JOC, World Meteorological Organization, Geneva, Switzerland, 121-275, 1979.
45. U. Mayer. A numerical scheme for moving boundary problems that are gradient flows for the area functional. *European J. Appl. Math.*, 11:61–80, 2000.
46. R. Mencl and H. Müller. Interpolation and approximation of surfaces from three-dimensional scattered data points. In *Scientific Visualization*, Dagstuhl'97, IEEE Publishers, 2000.
47. A.F. Möbius. Ueber eine neue Behandlungsweise der analytischen Sphärik. In *Abhandlungen bei Begründung der Königl. Sächs. Gesellschaft der Wissenschaften*, Jablonowski Gesellschaft, Leipzig, 45–86, 1846; Reappeared in: A.F. Möbius, *Gesammelte Werke*, F. Klein, editor, Vol. 2, Leipzig, 1886, 1–54.
48. T. Morton and M. Neamtu. Error bounds for solving pseudodifferential equations on spheres by collocation with zonal kernels. To appear in *J. Approx. Theory*.
49. C. Müller. *Spherical Harmonics*, Lecture Notes in Mathematics, vol. 17, Springer-Verlag, Berlin-Heidelberg, 1966.
50. F.J. Narcowich. Generalized Hermite interpolation and positive definite kernels on a Riemannian manifold. *J. Math. Anal. Appl.*, 190:165–193, 1995.
51. F.J. Narcowich and J.D. Ward. Nonstationary wavelets on the m -sphere for scattered data. *Appl. Comput. Harmon. Anal.*, 3:324–336, 1996.
52. M. Neamtu. Homogeneous simplex splines. *J. Comp. Appl. Math.*, 73:173–189, 1996.
53. M. Neamtu. What is the natural generalization of univariate splines to higher dimensions?. In T. Lyche and L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces: Oslo 2000*. Vanderbilt University Press, Nashville, TN, pp. 355–392, 2001.
54. M. Neamtu. Barycentric coordinates defined on surfaces. In preparation.
55. M. Neamtu, H. Pottmann, and L.L. Schumaker. Dual focal splines and rational curves with rational offsets. *Math. Eng. Ind.*, 7:139–154, 1998.
56. M. Neamtu, H. Pottmann, and L.L. Schumaker. Designing NURBS cam profiles using trigonometric splines. *J. Mech. Des.*, 120:175–180, 1998.

57. G.M. Nielson. Modeling and visualizing volumetric and surface-on-surface data. In H. Hagen, H. Mueller, and G. Nielson, editors, *Focus on Scientific Visualization*, Springer-Verlag, Berlin, 219–274, 1992.
58. G.M. Nielson and R. Ramaraj. Interpolation over a sphere based upon a minimum norm network. *Computer Aided Geometric Design*, 4:41–58, 1987.
59. K. Opitz and H. Pottmann. Computing shortest paths on polyhedra: applications in geometric modeling and scientific visualization. *Int. J. Comp. Geom. Applic.*, 4:165–178, 1994.
60. R.N. Pfeifle and H.-P. Seidel. Spherical triangular B-splines with application to data fitting. *Proc. Eurographics*, 89–96, 1995.
61. H. Pottmann. Interpolation on surfaces using minimum norm networks. *Computer Aided Geometric Design*, 9:51–67, 1992.
62. H. Pottmann and M. Eck. Modified multiquadric methods for scattered data interpolation over a sphere. *Computer Aided Geometric Design*, 7:313–321, 1990.
63. H. Pottmann, H. Hagen, and A. Divivier. Visualizing functions on a surface. *Journal of Visualization and Computer Animation*, 2:52–58, 1991.
64. H. Pottmann and K. Opitz. Curvature analysis and visualization for functions defined on Euclidean spaces or surfaces. *Computer Aided Geometric Design*, 11:655–674, 1994.
65. R. Ramaraj. Interpolation and display of scattered data over a sphere. MS thesis, Arizona State University, 1986.
66. R.J. Renka. Interpolation of data on the surface of a sphere. *ACM Trans. Math. Software*, 10:417–436, 1984.
67. R.J. Renka. Algorithm 623: Interpolation on the surface of a sphere. *ACM Trans. Math. Software*, 10:437–439, 1984.
68. P. Schröder and W. Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. *Proceedings of Siggraph*, 1995.
69. L.L. Schumaker. *Spline Functions: Basic Theory*. Interscience, New York, 1981. Reprinted by Krieger, Malabar, Florida, 1993.
70. L.L. Schumaker. Triangulation methods in CAGD. *IEEE Computer Graphics and Applications*, 13:47–52, 1993.
71. L.L. Schumaker and C.R. Traas. Fitting scattered data on spherelike surfaces using tensor products of trigonometric and polynomial splines. *Numer. Math.*, 60:133–144, 1991.
72. S.L. Svensson. Finite elements on the sphere. *J. Approx. Theory*, 40:246–260, 1984.
73. W. Sweldens. The lifting scheme: a construction of second generation wavelets. *SIAM J. Math. Anal.*, 29:511–546, 1998.
74. C.R. Traas. Smooth approximation of data on the sphere with splines. *Computing*, 38:177–184, 1987.
75. R. Veltkamp. *Closed object boundaries from scattered points*, Lecture Notes in Computer Science, 885, Springer-Verlag, Berlin, 1994.
76. G. Wahba. Spline interpolation and smoothing on the sphere. *SIAM J. Sci. Statist. Comput.*, 2:5–16, 1981. Errata: G. Wahba, Spline interpolation and smoothing on the sphere. *SIAM J. Sci. Statist. Comput.*, 3:385–386, 1982.
77. G. Wahba. Vector splines on the sphere, with application to the estimation of vorticity

- and divergence from discrete, noisy data. In W. Schempp and K. Zeller, editors, *Multivariate Approximation Theory II*, Birkhäuser (Basel), 407–429, 1982.
78. G. Wahba. Surface fitting with scattered noisy data on Euclidean d -space and on the sphere. *Rocky Mountain J. Math*, 14:281–299, 1984.
79. D.L. Williamson and G.L. Browning. Comparison of grids and difference approximations for numerical weather prediction over a sphere. *J. Appl. Meteor.*, 12:264–274, 1973.

Chapter 10

Box Splines

Hartmut Prautzsch and Wolfgang Boehm

This chapter provides a brief introduction to box and half-box splines with particular focus on triangular splines and surface design. A particular example of box splines are the B-splines with equidistant knots. In general, box splines consist of regularly arranged polynomial pieces and they have a useful geometric interpretation. Namely they can be viewed as density functions of the shadows of higher dimensional boxes and half-boxes. Of particular interest for Geometric Design are box spline surfaces that consist of triangular polynomial pieces. These box spline surfaces have planar domains, but it is quite simple to construct arbitrary two-dimensional surfaces, i.e., manifolds, with these box splines.

10.1. BOX SPLINES

A very comprehensive treatment of box splines and their general theory is given in the book by de Boor, Höllig and Riemenschneider [10] who also give valuable information on many references.

10.1.1. Inductive definition

An s -variate **box spline** $B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k)$ is determined by some k directions \mathbf{v}_i in \mathbf{R}^s . For simplicity, we will assume that $k \geq s$ and that $\mathbf{v}_1, \dots, \mathbf{v}_s$ are linearly independent. Under these assumptions the box splines $B_\kappa(\mathbf{x}) := B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_\kappa)$, $\kappa = s + 1, \dots, k$, are defined by successive convolutions [6,21,22,35],

$$B_s(\mathbf{x}) := \begin{cases} 1/\det[\mathbf{v}_1 \dots \mathbf{v}_s] & \text{if } \mathbf{x} \in [\mathbf{v}_1 \dots \mathbf{v}_s][0, 1)^s \\ 0 & \text{else} \end{cases}$$
$$B_\kappa(\mathbf{x}) := \int_0^1 B_{\kappa-1}(\mathbf{x} - t\mathbf{v}_\kappa) dt, \quad \kappa > s .$$

This is illustrated in Figure 10.1 for $s = 2$ and $[\mathbf{v}_1 \dots \mathbf{v}_4] = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$. Figure 10.2 shows translates of some box splines that are tensor product box splines.

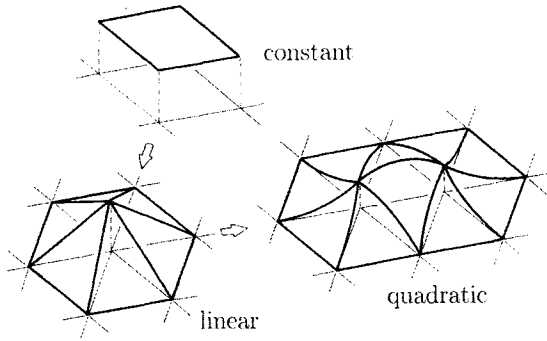


Figure 10.1. Bivariate box splines over the triangular grid.

These box splines are normalized such that

$$\int_{\mathbf{R}^s} B_k(\mathbf{x}) d\mathbf{x} = 1 ,$$

which can easily be verified for $k = s$ and further by induction over k . Namely

$$\int_{\mathbf{R}^s} \int_0^1 B_{k-1}(\mathbf{x} - t\mathbf{v}_k) dt d\mathbf{x} = \int_0^1 \int_{\mathbf{R}^s} B_{k-1}(\mathbf{x} - t\mathbf{v}_k) d\mathbf{x} dt = \int_0^1 dt = 1 .$$

10.1.2. Geometric definition

A box spline $B_k(\mathbf{x}) = B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k)$ can also be constructed geometrically as illustrated in Figure 10.3 for $k = 4$ and $s = 2$.

Let π be the orthogonal projection

$$\pi : [t_1 \dots t_k]^t \mapsto [t_1 \dots t_s]^t ,$$

and let

$$\beta_k = [\mathbf{u}_1 \dots \mathbf{u}_k][0, 1]^k$$

be a parallelepiped such that $\mathbf{v}_i = \pi \mathbf{u}_i$.

Then, $B_k(\mathbf{x})$ represents the density of the “shadow” of β_k , i.e.,

$$B_k(\mathbf{x}) = \frac{1}{\text{vol}_k \beta_k} \text{vol}_{k-s} \beta_k(\mathbf{x}) , \tag{10.1}$$

where

$$\beta_k(\mathbf{x}) = \pi^{-1} \mathbf{x} \cap \beta_k .$$

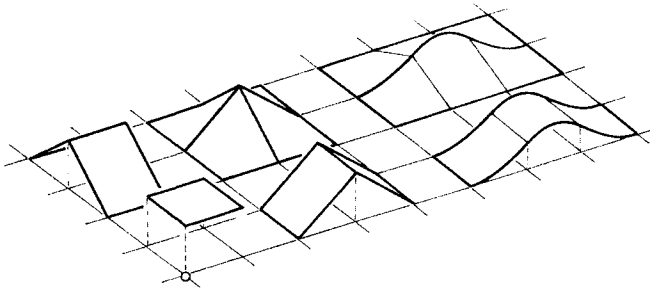


Figure 10.2. Translates of tensor product box splines.

For $k = 3$ and $s = 2$, the corresponding geometric construction is illustrated in Figure 10.4. It is due to [6], while the idea of polyhedral shadows can be traced back to [36] and [37].

We prove this characterization of box splines by induction: For $k = s$, Equation (10.1) is obvious, and, for greater values of k , we observe that

$$\beta_k(\mathbf{x}) = \bigcup_{s \in [0,1]} (\beta_{k-1}(\mathbf{x}) + s\mathbf{u}_k) .$$

Hence, if h measures the distance between β_{k-1} and $\mathbf{u}_k + \beta_k$ along the k th unit vector in \mathbf{R}^k , then it follows that

$$\text{vol}_{k-s}\beta_k(\mathbf{x}) = \int_0^1 h \text{vol}_{k-s-1}(\beta_{k-1}(\mathbf{x} - s\mathbf{v}_k)) ds ,$$

which corresponds, up to a constant factor, to the inductive definition of box splines. Consequently, $\text{vol}_{k-s}\beta_k(\mathbf{x})$ is a multiple of the box spline $B_k(\mathbf{x})$, and, since

$$\int_{\mathbf{R}^s} \text{vol}_{k-s}\beta_k(\mathbf{x}) d\mathbf{x} = \text{vol}_k\beta_k \quad \text{and} \quad \int_{\mathbf{R}^s} B_k(\mathbf{x}) d\mathbf{x} = 1 ,$$

Equation (10.1) follows.

The parallelepiped β_k is an affine image of the unit cube, which, sometimes, is also called a **box**. The projection π can be composed with this affine map. In this way, any box spline can also be constructed as the shadow of a box under a certain affine map. Hence, the name box spline, which is first used in [9].

10.1.3. Further definitions of Box splines

From the geometric definition it follows that a box spline is characterizable as the solution of the functional equation

$$\int_{\mathbf{R}^s} B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k) f(\mathbf{x}) d\mathbf{x} = \int_{[0,1]^k} f([\mathbf{v}_1 \dots \mathbf{v}_k]\mathbf{t}) dt$$

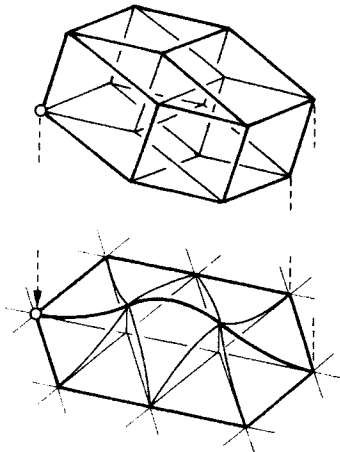


Figure 10.3. Box splines are “shadows” of boxes.

for all continuous test functions $f(\mathbf{x})$.

Another possibility to define a box spline is the following recursion

$$B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k) = \frac{1}{k-s} \sum_{r=1}^k (\alpha_r B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_r^* \dots \mathbf{v}_k) + (1-\alpha_r) B(\mathbf{x}-\mathbf{v}_r|\mathbf{v}_1 \dots \mathbf{v}_r^* \dots \mathbf{v}_k)) ,$$

where $\mathbf{x} = \sum_{r=1}^k \alpha_r \mathbf{v}_r$. This recursion is due to de Boor and Höllig [8]. A geometric proof can be found in [4] and numerical and algorithmic aspects are discussed in [2,13,7,25].

10.1.4. Basic properties of Box splines

From the geometric construction of box splines it follows that $B(\mathbf{x}) := B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k)$

- does not depend on the **ordering** of the directions \mathbf{v}_i ,
- is **positive** over the convex set $[\mathbf{v}_1 \dots \mathbf{v}_k][0, 1)^k$,
- has the **support** $\text{supp}B(\mathbf{x}) = [\mathbf{v}_1 \dots \mathbf{v}_k][0, 1)^k$,
- is **symmetric** with respect to the center of its support.

Further, let $B(\mathbf{x})$ be the shadow of a box β as in (10.1). The $(s-1)$ -dimensional faces of β projected into \mathbf{R}^s form a tessellation of the support. It is illustrated in Figure 10.5 for

$$[\mathbf{v}_1 \dots \mathbf{v}_k] = \begin{bmatrix} 1 & 1 & 1 & 0 \\ -1 & 0 & 1 & 1 \end{bmatrix} \quad \text{and} \quad [\mathbf{v}_1 \dots \mathbf{v}_k] = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} .$$

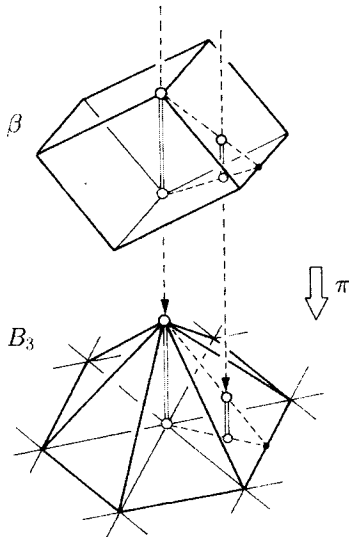


Figure 10.4. The geometric construction of a piecewise linear box spline over a triangular grid.

- The box spline $B(\mathbf{x})$ is polynomial of degree $\leq k - s$ over each tile of this partition.

For a proof, we observe that the extreme points of the convex sets $\pi^{-1}\mathbf{x} \cap \beta$ lie in s -dimensional faces of β . Hence, an extreme point is of the form $[\mathbf{x}^t \mathbf{e}^t]^t$, where $\mathbf{e} \in \mathbf{R}^{k-s}$ depends linearly on \mathbf{x} over the projection of the corresponding s -dimensional face. The volume of $\pi^{-1}\mathbf{x} \cap \beta$ can be expressed as a linear combination of determinants of $k - s \times k - s$ matrices whose columns represent differences of extreme points \mathbf{e} . Hence, the volume is a polynomial of degree $\leq k - s$ in \mathbf{x} over each tile of the tessellation above.

10.1.5. Derivatives

From the inductive or geometric definition it follows that the restricted box spline $B(y) := B(\mathbf{x} + y\mathbf{v}_r)$ is piecewise constant in y if $\mathbf{v}_r \notin \text{span}\{\mathbf{v}_1, \dots, \mathbf{v}_r^*, \dots, \mathbf{v}_k\}$. If $\mathbf{v}_r \in \text{span}\{\mathbf{v}_1, \dots, \mathbf{v}_r^*, \dots, \mathbf{v}_k\}$, then $B(y)$ is continuous since it can be obtained by a convolution from $B^*(y) = B(\mathbf{x} + y\mathbf{v}_r | \mathbf{v}_1 \dots \mathbf{v}_r^* \dots \mathbf{v}_k)$,

$$\begin{aligned}
 B(y) &= \int_0^1 B^*(y-t)dt = \int_{y-1}^y B^*(t)dt \\
 &= \int_{-\infty}^y B^*(t) - B^*(t-1)dt .
 \end{aligned}
 \tag{10.2}$$

Further, the **directional derivative** with respect to \mathbf{v}_r is given by

$$D_{\mathbf{v}_r} B(\mathbf{x}) = B'(y)|_{y=0} = B^*(\mathbf{x}) - B^*(\mathbf{x} - \mathbf{v}_r) .
 \tag{10.3}$$

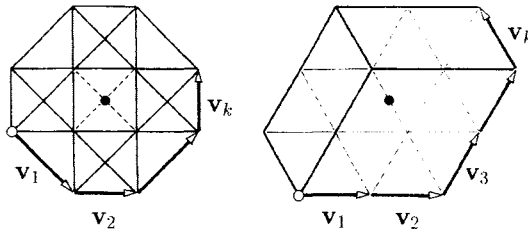


Figure 10.5. The supports of a quadratic and a cubic box spline over the criss cross and the regular triangular grid.

If $\mathbf{v}_1, \dots, \mathbf{v}_r^*, \dots, \mathbf{v}_k$ span \mathbf{R}^s for $r = 1, \dots, s$, then $B(\mathbf{x})$ is continuous and its directional derivatives can be written as linear combinations of translates of the box splines $B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_r^* \dots \mathbf{v}_k)$. Applying this argument repeatedly we see that

- $B(\mathbf{x})$ is r times continuously differentiable if all subsets of $\{\mathbf{v}_1 \dots \mathbf{v}_k\}$ obtained by deleting $r + 1$ vectors \mathbf{v}_i span \mathbf{R}^s .

Remark: Another and inductive proof of the polynomial properties of $B(\mathbf{x})$ is based on (10.3). Namely, if $B^*(\mathbf{x})$ is a polynomial of degree $\leq k - s - 1$ over each tile of the partition above, then $B^*(\mathbf{x} - \mathbf{v}_r)$ has the same property. Consequently, $B(\mathbf{x})$ is piecewise polynomial of degree $k - s$ in each direction \mathbf{v}_r over the partition above and hence in \mathbf{x} .

Remark: The piecewise quadratic C^1 box spline whose support is shown on the left side of Figure 10.5 is called a **Zwart-Powell element**. Because of its symmetry, it is even quadratic and not piecewise quadratic over the inner square.

10.2. BOX SPLINE SURFACES

10.2.1. Translates of Box splines

In the sequel we assume that the directions $\mathbf{v}_1, \dots, \mathbf{v}_k$ are in \mathbf{Z}^s and, as before, that $\mathbf{v}_1, \dots, \mathbf{v}_s$ span \mathbf{R}^s .

Obviously, the translates $B(\mathbf{x} - \mathbf{j}|\mathbf{v}_1 \dots \mathbf{v}_s)$, $\mathbf{j} \in [\mathbf{v}_1 \dots \mathbf{v}_s]\mathbf{Z}^s$, of the piecewise constant box spline sum to

$$\gamma := 1/|\det[\mathbf{v}_1 \dots \mathbf{v}_s]| . \tag{10.4}$$

Since \mathbf{Z}^s can be decomposed into, say m , sets $\mathbf{i} + [\mathbf{v}_1 \dots \mathbf{v}_s]\mathbf{Z}^s$, see Figure 10.6, it follows that

$$\sum_{\mathbf{i} \in \mathbf{Z}^s} B(\mathbf{x} - \mathbf{i}|\mathbf{v}_1 \dots \mathbf{v}_s) = m\gamma .$$

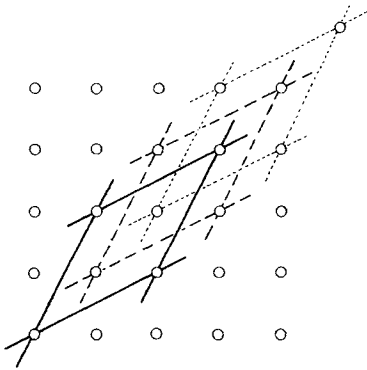


Figure 10.6. Decomposition of \mathbf{Z}^s into translates of coarser grids.

Further, since $\int_0^1 m\gamma dt = m\gamma$, we get by $k - s$ and again s successive convolutions

$$\begin{aligned} m\gamma &= \sum_{\mathbf{i} \in \mathbf{Z}^s} B(\mathbf{x} - \mathbf{i} | \mathbf{v}_1 \dots \mathbf{v}_k) \\ &= \sum_{\mathbf{i} \in \mathbf{Z}^s} B(\mathbf{x} - \mathbf{i} | \mathbf{e}_1 \dots \mathbf{e}_s \mathbf{v}_1 \dots \mathbf{v}_k) , \end{aligned}$$

where \mathbf{e}_i is the i th unit vector. Since the last sum is identically one due to (10.4), where $\mathbf{v}_1, \dots, \mathbf{v}_k$ are replaced by $\mathbf{e}_1 \dots \mathbf{e}_s$, we have shown that the (integer) shifts of any box spline $B(\mathbf{x}) := B(\mathbf{x} | \mathbf{v}_1 \dots \mathbf{v}_k)$ form a **partition of unity**.

Consequently, any **box spline surface**

$$\mathbf{s}(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbf{Z}^s} \mathbf{c}_i B(\mathbf{x} - \mathbf{i})$$

is an affine combination of its **control points** \mathbf{c}_i and this surface representation is **affinely invariant** meaning that under any affine map the control point images control the surface image.

Since the box splines are non-negative, $\mathbf{s}(\mathbf{x})$ is even a convex combination of its control points and lies in their **convex hull**.

Further, we see that $B(\mathbf{x} - \mathbf{i} | \mathbf{v}_1 \dots \mathbf{v}_k), \mathbf{i} \in \mathbf{Z}^s$, is linearly dependent if $|\det[\mathbf{v}_1 \dots \mathbf{v}_s]| \neq 1$. Since the ordering of the \mathbf{v}_i does not matter, this sequence is linearly dependent [9] also if there is any independent subsequence $\mathbf{v}_{i_1}, \dots, \mathbf{v}_{i_s}$ with $|\det[\mathbf{v}_{i_1} \dots \mathbf{v}_{i_s}]| \neq 1$. The converse is also true, see [16,19,23,24]. Together we have the following.

*$B(\mathbf{x} - \mathbf{i} | \mathbf{v}_1 \dots \mathbf{v}_k), \mathbf{i} \in \mathbf{Z}^s$, is linearly independent over each open subset of \mathbf{R}^s if and only if $[\mathbf{v}_1 \dots \mathbf{v}_k]$ is **unimodular**,*

which means that the determinant of any regular submatrix $[\mathbf{v}_{i_1} \dots \mathbf{v}_{i_s}]$ is 1 or -1 .

10.2.2. Derivatives and polynomial properties

If the directions $\mathbf{v}_1, \dots, \mathbf{v}_{k-1}$ span \mathbf{R}^s , then we can compute the **directional derivative** $D_{\mathbf{v}_k} \mathbf{s}$ of \mathbf{s} with respect to \mathbf{v}_k . Using the derivative formula (10.3) in 10.1.5 we obtain

$$D_{\mathbf{v}_k} \mathbf{s}(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbf{Z}^s} \nabla_{\mathbf{v}_k} \mathbf{c}_i B(\mathbf{x} - \mathbf{i} | \mathbf{v}_1 \dots \mathbf{v}_{k-1}) , \tag{10.5}$$

where $\nabla_{\mathbf{v}} \mathbf{c}_i := \mathbf{c}_i - \mathbf{c}_{i-\mathbf{v}}$, see [9]. Further, if for all $j = 1, \dots, k$ the $k - 1$ directions $\mathbf{v}_1, \dots, \mathbf{v}_j^* \dots \mathbf{v}_k$ span \mathbf{R}^s , then $B(\mathbf{x})$ is continuous as shown in 10.1.5 and the span of its shifts contains all linear polynomials as we show in the sequel. In particular, if

$$\mathbf{m}_i := \mathbf{i} + \frac{1}{2}(\mathbf{v}_1 + \dots + \mathbf{v}_k)$$

denotes the **center** of $\text{supp}B(\mathbf{x} - \mathbf{i})$, then

$$\sum_{\mathbf{i} \in \mathbf{Z}^s} \mathbf{m}_i B(\mathbf{x} - \mathbf{i}) = \mathbf{x} . \tag{10.6}$$

Namely, because of symmetry, this equation holds for $\mathbf{x} = \mathbf{o}$, and for all $j = 1, \dots, s$ we have [30]

$$D_{\mathbf{v}_j} \sum_{\mathbf{i}} \mathbf{m}_i B(\mathbf{x} - \mathbf{i}) = \mathbf{v}_j .$$

Since the box spline representation is affinely invariant, we obtain for any linear polynomial $l(\mathbf{x})$

$$l(\mathbf{x}) = \sum l(\mathbf{m}_i) B(\mathbf{x} - \mathbf{i}) .$$

This property is referred to as the **linear precision** of the box spline representation.

More generally, let the directions $\mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbf{Z}^s$ span \mathbf{R}^s , and assume that the associated box spline $B(\mathbf{x}) := B(\mathbf{x} | \mathbf{v}_1 \dots \mathbf{v}_k)$ is r times continuously differentiable. Then it follows, for example by induction over k that the map $c(\mathbf{x}) \mapsto \sum c(\mathbf{i}) B(\mathbf{x} - \mathbf{i})$ is a regular linear map on the space of all polynomials of degree $\leq r + 1$. See [9] for further results.

10.2.3. Convexity

Let $\mathbf{e}_1, \dots, \mathbf{e}_s$ denote the unit directions and let $\mathbf{e} = \mathbf{e}_1 + \dots + \mathbf{e}_s$. Further, let

$$\mathbf{s}(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbf{Z}^s} \mathbf{c}_i B(\mathbf{x} - \mathbf{i} | \mathbf{e}_1 \dots \mathbf{e}_1 \dots \mathbf{e}_s \dots \mathbf{e}_s \mathbf{e} \dots \mathbf{e})$$

be a box spline surface with these directions. The piecewise linear box spline surface

$$\mathbf{c}(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbf{Z}^s} \mathbf{c}_i B(\mathbf{x} - \mathbf{i} | \mathbf{e}_1 \dots \mathbf{e}_s \mathbf{e})$$

is said to be the control net of the surface $\mathbf{s}(\mathbf{x})$.

If the control net $\mathbf{c}(\mathbf{x})$ is a scalar valued and convex, then the surface $\mathbf{s}(\mathbf{x})$ is also a convex function, see [20,31]. Furthermore, any control net of $\mathbf{s}(\mathbf{x})$ obtained under subdivision as described next is convex [31].

10.2.4. Subdivision

Any box $\beta = [\mathbf{u}_1 \dots \mathbf{u}_k][0, 1)^k$ in \mathbf{R}^k can be partitioned into 2^k translates of the scaled box $\hat{\beta} = \beta/2$ spanned by the half directions $\hat{\mathbf{u}}_i = \mathbf{u}_i/2$, see Figure 10.7. Based on this observation Prautzsch [28] concluded in 1993 that the non-normalized “shadow” $M_\beta(\mathbf{x}) = \text{vol}_{k-s}(\pi^{-1}\mathbf{x} \cap \beta)$ of β under the projection $\pi : [t_1 \dots t_k]^k \mapsto [t_1 \dots t_s]^s$ can be written as a linear combination of translates of the scaled box spline $M_{\hat{\beta}}(\mathbf{x}) = 2^{s-k}M_\beta(2\mathbf{x})$. Consequently, if the projections $\mathbf{v}_i = \pi\mathbf{u}_i$ lie in \mathbf{Z}^s , then any box spline surface

$$\mathbf{s}(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbf{Z}^s} \mathbf{c}_i^1 B(\mathbf{x} - \mathbf{i})$$

with $B(\mathbf{x}) = B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k)$ has also a “finer” representation

$$\mathbf{s}(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbf{Z}^s} \mathbf{c}_i^2 B(2\mathbf{x} - \mathbf{i}) .$$

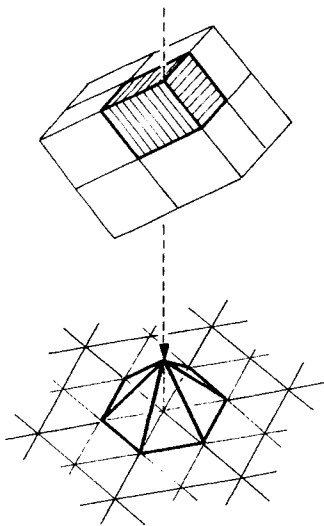


Figure 10.7. Subdividing a box spline by subdividing its box.

The new control points \mathbf{c}_i^2 can be computed iteratively from the initial control points

\mathbf{c}_i^1 by the recursion

$$\mathbf{d}^0(\mathbf{i}) := \begin{cases} \mathbf{o} & \text{if } \mathbf{i}/2 \notin \mathbb{Z}^s \\ \mathbf{c}_{\mathbf{i}/2}^1 & \text{if } \mathbf{i}/2 \in \mathbb{Z}^s \end{cases},$$

$$\mathbf{d}^r(\mathbf{i}) := (\mathbf{d}^{r-1}(\mathbf{i}) + \mathbf{d}^{r-1}(\mathbf{i} - \mathbf{v}_r))/2, \quad r = 1 \dots k,$$

$$\mathbf{c}_i^2 := 2^s \mathbf{d}^k(\mathbf{i}).$$

For a proof we follow the ideas in [28] and subdivide each box

$$\beta_{r-1} = [\hat{\mathbf{u}}_1 \dots \hat{\mathbf{u}}_{r-1} \mathbf{u}_r \dots \mathbf{u}_k][0, 1)^k$$

into β_r and $\hat{\mathbf{u}}_r + \beta_r$. The associated shadows satisfy

$$M_{\beta_{r-1}}(\mathbf{x}) = M_{\beta_r}(\mathbf{x}) + M_{\beta_r}(\mathbf{x} - \hat{\mathbf{v}}_r), \tag{10.7}$$

where $\hat{\mathbf{v}}_r = \mathbf{v}_r/2$. Dividing this equation by $\text{vol } \beta_{r-1} = 2 \text{vol } \beta_r$, gives

$$B_{r-1}(\mathbf{x}) = (B_r(\mathbf{x}) + B_r(\mathbf{x} - \hat{\mathbf{v}}_r))/2, \tag{10.8}$$

where $B_r(\mathbf{x}) := B(\mathbf{x}|\hat{\mathbf{v}}_1 \dots \hat{\mathbf{v}}_r, \mathbf{v}_{r+1} \dots \mathbf{v}_k)$.

Using this identity repeatedly and the relation $B(\mathbf{x}|\hat{\mathbf{v}}_1 \dots \hat{\mathbf{v}}_k) = 2^s B(2\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k)$ gives

$$\begin{aligned} \mathbf{s}(\mathbf{x}) &= \sum_{\mathbf{i} \in \mathbb{Z}^s} \mathbf{c}_i^1 B_0(\mathbf{x} - \mathbf{i}) \\ &= \sum_{\mathbf{i} \in \mathbb{Z}^s} \mathbf{d}^r(\mathbf{i}) B_r(\mathbf{x} - \mathbf{i}/2), \quad r = 0, 1 \dots k, \\ &= \sum_{\mathbf{i} \in \mathbb{Z}^s} \mathbf{c}_i^2 B_0(2\mathbf{x} - \mathbf{i}), \end{aligned}$$

which concludes the proof.

For quadratic univariate box splines with equidistant knots this algorithm bears the name of Chaikin [11] although it had been discussed already by de Rham [34]. Lane and Riesenfeld [26] generalized Chaikin's algorithm to univariate box splines of arbitrary degree with equidistant knots and in [3], which prepublishes results of [28] a mask is introduced to describe the subdivision algorithm above for three direction box splines. Later Loop [27] used Boehm's mask to build a subdivision algorithm for arbitrary triangular nets.

Remark: If $[\mathbf{v}_1 \dots \mathbf{v}_s]\mathbb{Z}^s = \mathbb{Z}^s$, then $2^s \mathbf{d}_i^s = \mathbf{c}_{\lfloor \mathbf{i}/2 \rfloor}^1$ and every point \mathbf{c}_i^2 is a convex combination of the initial points \mathbf{c}_i^1 . Moreover, the \mathbf{c}_i^2 lie in the convex hull of the \mathbf{c}_i^1 also if $[\mathbf{v}_1 \dots \mathbf{v}_s]\mathbb{Z}^k = \mathbb{Z}^s$.

10.2.5. General subdivision

It is straightforward to generalize the subdivision algorithm to obtain for any $m \in \mathbb{N}$ the finer representation

$$\mathbf{s}(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbb{Z}^s} \mathbf{c}_i^m B(m\mathbf{x} - \mathbf{i}),$$

where the “points” $\mathbf{c}_i^m = m^s \mathbf{d}^k(\mathbf{i})$ can be computed successively by

$$\mathbf{d}^0(\mathbf{i}) := \begin{cases} \mathbf{o} & \text{if } \mathbf{i}/m \notin \mathbf{Z}^s \\ \mathbf{c}_{\mathbf{i}/m}^1 & \text{if } \mathbf{i}/m \in \mathbf{Z}^s \end{cases},$$

$$\mathbf{d}^r(\mathbf{i}) := \frac{1}{m} \sum_{l=0}^{m-1} \mathbf{d}^{r-1}(\mathbf{i} - l\mathbf{v}_r), \quad r = 1, \dots, k.$$

In this and in an even more general form this algorithm can be found in [12] and [17,18], where it is derived algebraically.

The subdivision algorithm can be used a second time to compute control points of $\mathbf{s}(\mathbf{x})$ over any finer grid $\mathbf{Z}^s/(mn)$. Since the partition of a box β into translates of the scaled box $\beta/(mn)$ is unique, we get as a result the same points \mathbf{c}_i^{mn} that can be computed directly from the initial control points \mathbf{c}_i^1 by one application of the subdivision algorithm.

Similarly we see that the points \mathbf{c}_i^m do not depend on the ordering of the directions $\mathbf{v}_1, \dots, \mathbf{v}_k$, i.e., the ordering of the averaging steps.

Remark: The geometric derivation of the subdivision algorithm above shows that any new control point \mathbf{c}_i^m only depends on the control points \mathbf{c}_i^1 , where $\text{supp}B(m\mathbf{x} - \mathbf{j}) \subset \text{supp}B(\mathbf{x} - \mathbf{i})$. Their number is bounded by some h not depending on m and \mathbf{j} . Hence, we have

$$\|\mathbf{c}_i^m\| \leq h \sup_{i \in \mathbf{Z}^s} \|\mathbf{c}_i^1\|. \tag{10.9}$$

10.2.6. Convergence under subdivision

Repeated subdivision by the algorithm 10.2.5 gives the control points of the finer representation

$$\mathbf{s}(\mathbf{x}) = \sum_{i \in \mathbf{Z}^s} \mathbf{c}_i^m B(m\mathbf{x} - \mathbf{i}), \quad \text{where } B(\mathbf{x}) = B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k),$$

of any surface $\mathbf{s}(\mathbf{x})$ over all scaled grids \mathbf{Z}^s/m , $m \in \mathbf{N}$. A major value of this procedure is that under some reasonable conditions on $V = [\mathbf{v}_1 \dots \mathbf{v}_k]$ the points \mathbf{c}_i^m converge towards $\mathbf{s}(\mathbf{x})$, see [26,28,29,15,14].

Let h be as in (10.9) and $M = \sup\{\|\nabla_{\mathbf{v}_r} \mathbf{c}_i^1\|\}$, where $\mathbf{i} \in \mathbf{Z}^s$ and $\mathbf{v}_1, \dots, \mathbf{v}_r^*, \dots, \mathbf{v}_k$ span \mathbf{R}^s . Then the following holds.

$$\text{If } [\mathbf{v}_1 \dots \mathbf{v}_k] \mathbf{Z}^k = \mathbf{Z}^s, \text{ then } \|\mathbf{c}_i^m - \mathbf{s}(\mathbf{x})\| \leq hM/m \text{ for all } \mathbf{i},$$

where $B(m\mathbf{x} - \mathbf{i}) > 0$.

In this generality this result is due to de Boor et al. [10] who also show quadratic convergence under the conditions $[\mathbf{v}_1 \dots \mathbf{v}_i^* \dots \mathbf{v}_k] \mathbf{Z}^{k-1} = \mathbf{Z}^s$ and $B(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k)$ is differentiable.

On the other hand, if $[\mathbf{v}_1 \dots \mathbf{v}_k] \mathbf{Z}^k \neq \mathbf{Z}^s$, then convergence cannot be proved as we show for $s(\mathbf{x}) = B(\mathbf{x})$.

Namely, if there is some grid point $\mathbf{i} \in \mathbf{Z}^s$, which does not lie in $[\mathbf{v}_1 \dots \mathbf{v}_k] \mathbf{Z}^k$, then all grid points $\mathbf{j} \in J := \mathbf{i} + [\mathbf{v}_1 \dots \mathbf{v}_s] \mathbf{Z}^s$ do also not lie in $[\mathbf{v}_1 \dots \mathbf{v}_k] \mathbf{Z}^k$. Since $\sum_{j \in J} B(\mathbf{x} - \mathbf{j}) > 0$, see (10.4) in 10.2.1, there is for every $\mathbf{x} \in \mathbf{R}^s$ some $\mathbf{j} \in J$ such that $B(\mathbf{x} - \mathbf{j}) > 0$.

Subdividing the single box spline $B(\mathbf{x})$ by the algorithm in 10.2.5 gives control points c_i^m such that

$$B(\mathbf{x}) = \sum_{i \in \mathbb{Z}^s} c_i^m B(m\mathbf{x} - \mathbf{i}) .$$

The geometric derivation of the algorithm in 10.2.4 shows that $c_i^m \neq 0$ if and only if $\mathbf{i} \in [\mathbf{v}_1 \dots \mathbf{v}_k] \mathbb{Z}_m^k$.

Thus if $[\mathbf{v}_1 \dots \mathbf{v}_k] \mathbb{Z}^k \neq \mathbb{Z}^s$ or $J \neq \emptyset$, there is for every $\mathbf{x} \in \mathbb{R}^s$ some zero control point $c_j^m = 0$ with $B(m\mathbf{x} - \mathbf{j}) > 0$. For all \mathbf{x} , where $B(\mathbf{x}) > 0$, these control points do not converge to $B(\mathbf{x})$ as m tends to infinity.

10.2.7. Bézier representation

Often it is useful or necessary to have the Bézier representation of a box spline surface. In this section we describe how to compute the Bézier points of a box spline surface over a regular triangular grid, see also [35,1,28,5].

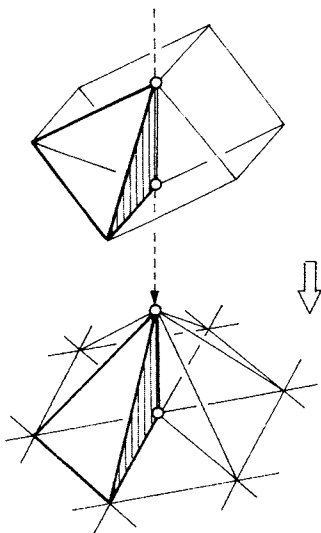


Figure 10.8. Bernstein polynomials are shadows of simplices.

Let $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ be the unit directions $[1 \ 0]^t, [0 \ 1]^t, -[1 \ 1]^t$ and let $\mathbf{v}_1, \dots, \mathbf{v}_k \in \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$. Further, let $\mathbf{b}_n(\mathbf{i}), \mathbf{i} \in \mathbb{Z}^2$, be the Bézier points of the box spline surface

$$s_n(\mathbf{x}) = \sum_{i \in \mathbb{Z}^2} c_i B(\mathbf{x} - \mathbf{i} | \mathbf{e}_1 \mathbf{e}_2 \mathbf{v}_1 \dots \mathbf{v}_n)$$

such that $\mathbf{s}_n(\mathbf{x})$ restricted to some grid triangle with vertices $\mathbf{a}, \mathbf{a} + \mathbf{e}_1, \mathbf{a} - \mathbf{e}_\mu$ and $\mu = 2$ or 3 has the Bézier representation

$$\mathbf{s}_n(\mathbf{x}) = \sum \mathbf{b}_n(n\mathbf{a} + i\mathbf{e}_1 + j\mathbf{e}_\mu) \frac{n!}{i!j!k!} u^i v^j w^k ,$$

where $i + j + k = n$ and $\mathbf{x} = u\mathbf{a} + v(\mathbf{a} + \mathbf{e}_1) + w(\mathbf{a} - \mathbf{e}_\mu)$.

Recall that integration means summation of the Bézier points. Consequently, since

$$\mathbf{s}_n(\mathbf{x}) = \int_0^1 \mathbf{s}_{n-1}(\mathbf{x} - t\mathbf{v}_n) dt ,$$

we can compute the Bézier points $\mathbf{b}_n(\mathbf{i})$ recursively from the copies

$$\mathbf{a}_n(n\mathbf{j} + \mathbf{v}_n - r\mathbf{e}_\mu - s\mathbf{e}_\nu) := \mathbf{b}_{n-1}((n-1)\mathbf{j} - r\mathbf{e}_\mu - s\mathbf{e}_\nu) ,$$

where $\mu, \nu \in 1, 2, 3$ and $\mathbf{e}_\mu + \mathbf{e}_\nu = -\mathbf{v}_n$ and $r, s = 0, \dots, n-1$ and $\mathbf{j} \in \mathbb{Z}^2$, by the summation

$$\mathbf{b}_n(\mathbf{i}) := \frac{1}{n} \sum_{\nu=0}^{n-1} \mathbf{a}_n(\mathbf{i} - \nu\mathbf{v}_n)$$

as is illustrated schematically in Figure 10.9 for $n = 3$.

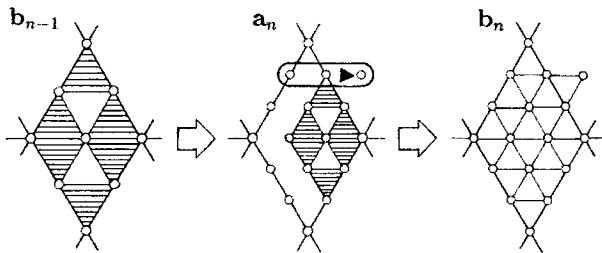


Figure 10.9. Computing Bézier points recursively.

If $\mathbf{v}_1 = \mathbf{e}_3$, then $\mathbf{b}_1(\mathbf{i}) = \mathbf{c}_i$, which terminates the recursion.

Note that the Bézier net \mathbf{b}_n can be computed faster if the summation is replaced by

$$\mathbf{b}_n(\mathbf{i}) := \mathbf{b}_n(\mathbf{i} - \mathbf{v}_n) + [\mathbf{a}_n(\mathbf{i}) - \mathbf{a}_n(\mathbf{i} - n\mathbf{v}_n)]/n .$$

The summation step can also be described by a convolution of \mathbf{a}_n with the mask

$$\sigma_n(\mathbf{i}) = \begin{cases} 1/n & \text{for } \mathbf{i} = \mathbf{o}, -\mathbf{v}_n, \dots, -(n-1)\mathbf{v}_n \\ 0 & \text{otherwise} \end{cases}$$

defined on \mathbb{Z}^2 . This means

$$\mathbf{b}_n = \mathbf{a}_n * \sigma_n ,$$

where

$$(\mathbf{a}_n * \sigma_n)(\mathbf{i}) = \sum_{\mathbf{j} \in \mathbb{Z}^2} \mathbf{a}_n(\mathbf{j}) \sigma_n(\mathbf{j} - \mathbf{i}) .$$

10.2.8. Bézier representation of symmetric Box splines

Of particular interest are the symmetric box splines, where $\mathbf{e}_1 \mathbf{e}_2 \mathbf{v}_1 \dots \mathbf{v}_n = \mathbf{e}_1 \mathbf{e}_2 \mathbf{e}_3 \dots \mathbf{e}_1 \mathbf{e}_2 \mathbf{e}_3$. For these box splines there is a symmetric version of the algorithm above. Combining three successive integrations leads to the recursion

$$\mathbf{b}_n = C_n \mathbf{b}_{n-3} * \alpha_n ,$$

where the mask α_n represents a degree elevated Bézier net of the symmetric piecewise linear box spline,

$$\alpha(\mathbf{i}) = B(\mathbf{i}/n | \mathbf{e}_1 \mathbf{e}_2 \mathbf{e}_3) ,$$

and where $C_n \mathbf{b}_{n-3}$ is a copy with additionally zeroes of the Bézier net \mathbf{b}_{n-3} as illustrated in Figure 10.10 for $n = 5$.

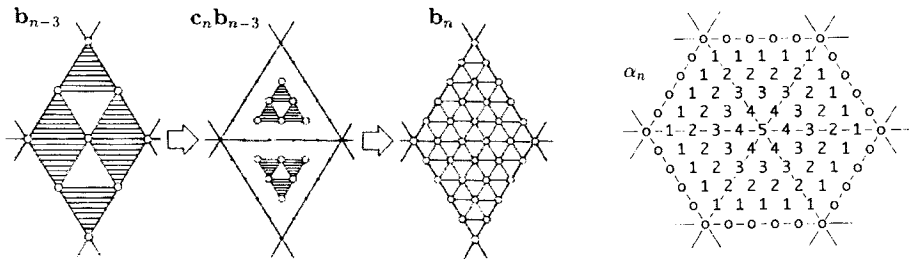


Figure 10.10. Symmetric recursion for the Bézier points.

Formally, C_n is defined by

$$C_n \mathbf{b}_{n-3}(n\mathbf{i} - (r+1)\mathbf{e}_1 - (s+1)\mathbf{e}_\mu) = \begin{cases} \mathbf{0} & \text{for } r \text{ or } s = -1 \\ \mathbf{b}_{n-3}((n-3)\mathbf{i} + r\mathbf{e}_1 - s\mathbf{e}_\mu) & \text{for } r, s = 0, \dots, n-3 \end{cases}$$

for $\mathbf{i} \in \mathbb{Z}^2$.

The computation of the Bézier points can be organized in different ways. In the sequel we present the geometrically nice one due to [1].

Let β be the Bézier net of the box spline $B(\mathbf{x}|\mathbf{e}_1\mathbf{e}_2\mathbf{v}_1 \dots \mathbf{v}_n)$ and let $\gamma = C_n\beta$. Then

$$\begin{aligned} \mathbf{b}_n(\mathbf{j}) &= \sum_{\mathbf{i} \in \mathbb{Z}^2} \mathbf{c}_i \beta(\mathbf{k} - \mathbf{i}) * \alpha(\mathbf{k}) \\ &= \sum_{\mathbf{k}} \sum_{\mathbf{i}} \mathbf{c}_i \beta(\mathbf{k} - \mathbf{i}) \cdot \alpha(\mathbf{k}) \\ &= \sum_{\mathbf{l}} \sum_{\mathbf{i}} \mathbf{c}_i \alpha(\mathbf{l} - \mathbf{i}) \cdot \beta(\mathbf{j} - \mathbf{l}) \\ &= \sum_{\mathbf{i}} \mathbf{c}_i \alpha(\mathbf{l} - \mathbf{i}) * \beta(\mathbf{l}) \end{aligned}$$

The net $\sum_{\mathbf{i}} \mathbf{c}_i \alpha(\mathbf{l} - \mathbf{i})$ is the Bézier net \mathbf{b}_1 after raising the degree from 1 to n . It is a refinement of the control net. Together with the mask $\beta(\mathbf{l})$ it is shown in Figure 10.11 for $n = 4$.

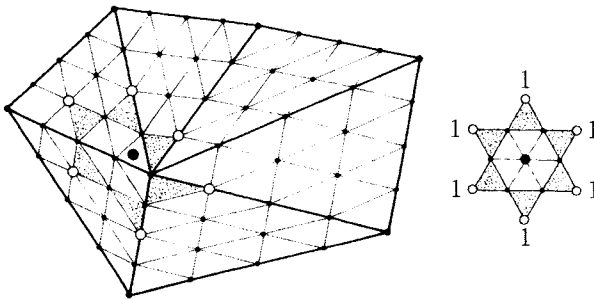


Figure 10.11. Computing the Bézier points of a quartic box spline surface.

10.2.9. Generalized Box spline surfaces

A bivariate box spline surface has a planar domain. However, with the symmetric box splines

$$B_k(\mathbf{x}) = B(\mathbf{x}|\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 \cdot^k \cdot \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3)$$

it is possible to build smooth arbitrarily free-form surfaces with non-planar domains. The support of the box splines $B_k(\mathbf{x})$ consists of k rings of triangles as shown in Figure 10.12 for $k = 2, 3$. This implies that any triangular patch of a box spline surface

$$\mathbf{s}(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbb{Z}^2} \mathbf{c}_i B_k(\mathbf{x})$$

is controlled by k rings of control points as illustrated schematically in Figure 10.13 for $k = 2, 3$.

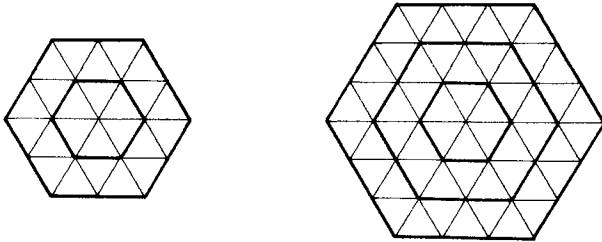


Figure 10.12. Support of the box splines B_2 (left) and B_3 (right).

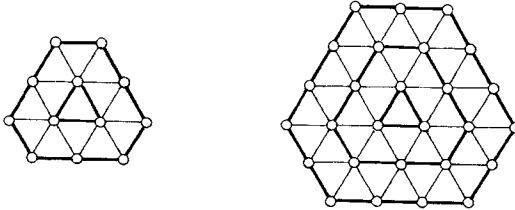


Figure 10.13. Net primitives of order 2 (left) and 3 (right).

We will call the net controlling one triangular patch a **minimal net** or a **B-primitive** of order k .

A **generalized box spline surface** of degree $3k - 2$ or order k is given by an arbitrary control net and consists of all triangular patches controlled by a B-primitive of order k that is part of the entire control net.

An interior vertex of the triangular control net is called **regular** if it has valence 6 and otherwise it is called **irregular**. If the irregular vertices of a triangular net are surrounded by sufficiently many regular vertices, then the generalized box spline surface has an m -sided hole for any irregular vertex with valence m as illustrated schematically in Figure 10.14 for $k = 2$ and $m = 4$ with a generalized box spline surface consisting of 2 rings of triangular patches around a four sided hole. Such a hole can be filled smoothly by $k - 1$ rings of triangles.

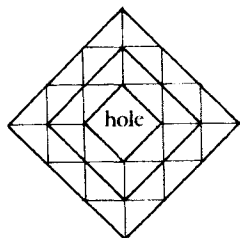


Figure 10.14. Schematic view of a hole of a generalized box spline surface of order 2.

The larger k , the larger the holes are schematically. A remedy for this is to let $k-2$ rings of control points around an irregular control point coalesce as illustrated schematically in Figure 10.15 for $k = 3$ and $m = 4$. When looking for the net primitives in the overall control net, we interpret these multiple control points in different ways as part of a regular triangular net that is collapsed into this one point so as to obtain as many net primitives as possible. Thus, the schematic size of the holes depends only on m and is the same for all k .

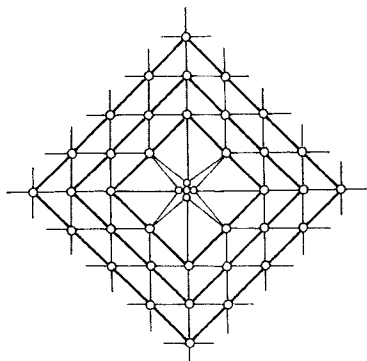


Figure 10.15. Control net of a generalized box spline surface of order 3 with multiple irregular vertex.

These holes can be filled with m triangular patches of degree $2k$ with G^k -joints, see [38,32,33]. Figure 10.16 shows an example of a generalized box spline surface of order $k = 2$. The control net is seen left, the generalized box spline surface in the middle and on the right the holes are filled with octic polynomials so as to obtain an overall curvature

continuous surface.



Figure 10.16. A generalized box spline surface of order 2 (middle) with control net (left) and G^2 -fillings (right). (courtesy of Georg Umlauf)

10.3. HALF-BOX SPLINES

10.3.1. Inductive definition

Half-box splines are defined over the triangular grid spanned by $\mathbf{e}_1 = [1\ 0]^t$, $\mathbf{e}_2 = [0\ 1]^t$ and $\mathbf{e}_3 = -[1\ 1]^t$. We give the inductive definition due to Sabin [35] and its geometric interpretation due to [28].

Splitting the unit square along its diagonal in direction \mathbf{e} gives the two triangles

$$\Delta := \{\mathbf{x} | 0 \leq x \leq y < 1\} \quad \text{and} \quad \nabla := \{\mathbf{x} | 0 \leq y < x < 1\} .$$

They support the piecewise constant half-box splines

$$H_{\Delta}(\mathbf{x}) := \begin{cases} 1 & \text{if } \mathbf{x} \in \Delta \\ 0 & \text{else} \end{cases}$$

and

$$H_{\nabla}(\mathbf{x}) := \begin{cases} 1 & \text{if } \mathbf{x} \in \nabla \\ 0 & \text{else} \end{cases} .$$

As with box spline we obtain half-box splines of higher order by successive convolutions,

$$H_{\Delta}(\mathbf{x} | \mathbf{v}_1 \dots \mathbf{v}_k) := \int_0^1 H_{\Delta}(\mathbf{x} - t\mathbf{v}_k | \mathbf{v}_1 \dots \mathbf{v}_{k-1}) dt$$

and

$$H_{\nabla}(\mathbf{x} | \mathbf{v}_1 \dots \mathbf{v}_k) := \int_0^1 H_{\nabla}(\mathbf{x} - t\mathbf{v}_k | \mathbf{v}_1 \dots \mathbf{v}_{k-1}) dt ,$$

where $k \geq 1$. Hence, we assume that the directions are the unit directions,

$$\mathbf{v}_1, \dots, \mathbf{v}_k \in \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}\} .$$

Note that the definitions of the two piecewise constant half-box splines $H_\Delta(\mathbf{x})$ and $H_\nabla(\mathbf{x})$ are not completely symmetric. Consequently, any two half-box splines $H_\Delta(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k)$ and $H_\nabla(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k)$ sum to the box spline $B(\mathbf{x}|\mathbf{e}_1\mathbf{e}_2\mathbf{v}_1 \dots \mathbf{v}_k)$ even for $k = 0$. The two piecewise cubic C^1 -half-box splines $H_\Delta(\mathbf{x}|\mathbf{e}_1\mathbf{e}_2\mathbf{e})$ and $H_\nabla(\mathbf{x}|\mathbf{e}_1\mathbf{e}_2\mathbf{e})$ are shown in Figure 10.17.

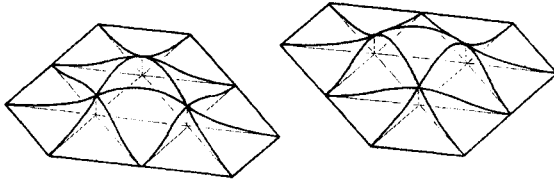


Figure 10.17. The two piecewise cubic C^1 -half-box splines.

10.3.2. Basic properties

As for box splines, one can derive the following properties of half-box splines. Because of symmetry reasons, it suffices to list these properties for $H(\mathbf{x}) := H_\Delta(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k)$.

A half-box spline is **normalized** such that

$$\int_{\mathbf{R}^2} H(\mathbf{x})d\mathbf{x} = 1/2 .$$

Any k independent directions $\mathbf{u}_1, \dots, \mathbf{u}_k \in \mathbf{R}^k$ define a half-box $\vartheta := \{\sum \mathbf{u}_i\alpha_i \mid 0 \leq \alpha_1 \leq \alpha_2 \text{ and } \alpha_2 \dots \alpha_k \in [0, 1]\}$. The **density of the shadow** of this half-box represents a half-box spline. If π denotes the projection from \mathbf{R}^k onto \mathbf{R}^2 mapping $\mathbf{u}_1, \dots, \mathbf{u}_k$ onto $\mathbf{v}_1, \dots, \mathbf{v}_k$, then

$$H(\mathbf{x}|\mathbf{v}_1 \dots \mathbf{v}_k) = \frac{1}{2 \text{vol}_k \vartheta} \text{vol}_{k-2}(\pi^{-1}\mathbf{x} \cap \vartheta)$$

as illustrated in Figure 10.18.

From this geometric construction, it follows that $H(\mathbf{x})$

- does not depend on the **ordering** of $\mathbf{v}_3, \dots, \mathbf{v}_k$,
- is **positive** over the convex set $\Delta + [\mathbf{v}_3 \dots \mathbf{v}_k][0, 1]^{k-2}$,
- has the **support** $\text{closure}(\Delta) + [\mathbf{v}_3 \dots \mathbf{v}_k][0, 1]^{k-2}$.

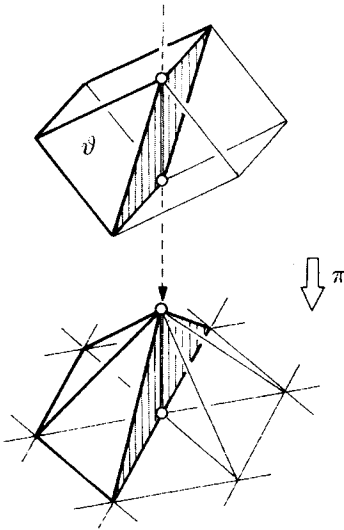


Figure 10.18. Geometric construction of a half-box spline.

10.3.3. Derivatives and polynomial structure

The following further properties of half-box splines follow most easily from their inductive definition. The half-box spline $H(\mathbf{x})$

- has the **directional derivative**

$$D_{\mathbf{v}_r} H(\mathbf{x}) = H(\mathbf{x} | \mathbf{v}_3 \dots \mathbf{v}_r^* \dots \mathbf{v}_k) - H(\mathbf{x} - \mathbf{v}_r | \mathbf{v}_3 \dots \mathbf{v}_r^* \dots \mathbf{v}_k) \tag{10.10}$$

with respect to \mathbf{v}_r , $r \geq 3$,

- is r times **continuously differentiable** if all subsets of $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ obtained by deleting $r + 1$ vectors \mathbf{v}_i span \mathbf{R}^2 ,
- is **polynomial of total degree $\leq k - 2$** over all triangles $\mathbf{i} + \Delta$ and $\mathbf{i} + \nabla$, $\mathbf{i} \in \mathbf{Z}^2$.

For example, the half-box splines $H_{\Delta}(\mathbf{x} | \mathbf{e}_1 \dots \mathbf{e}_k)$ are $2k - 1$ times continuously differentiable and are of polynomial degree $\leq 3k$.

10.4. HALF-BOX SPLINE SURFACES

10.4.1. Translates of Half-Box splines

Any pair of half-box splines

$$H_{\Delta}(\mathbf{x}) := H(\mathbf{x}|\mathbf{e}_3\mathbf{v}_1 \dots \mathbf{v}_k) ,$$

$$H_{\nabla}(\mathbf{x}) := H(\mathbf{x}|\mathbf{e}_3\mathbf{v}_1 \dots \mathbf{v}_k)$$

sums to the box spline $B(\mathbf{x}|\mathbf{e}_1\mathbf{e}_3\mathbf{v}_1 \dots \mathbf{v}_k)$. Hence, the translates $H_{\Delta}(\mathbf{x} - \mathbf{i})$ and $H_{\nabla}(\mathbf{x} - \mathbf{i})$, $\mathbf{i} \in \mathbb{Z}^2$, form a **partition of unity** .

Consequently, any **half-box spline surface**

$$s(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbb{Z}^2} (c_{\mathbf{i}}^{\Delta} H_{\Delta}(\mathbf{x} - \mathbf{i}) + c_{\mathbf{i}}^{\nabla} H_{\nabla}(\mathbf{x} - \mathbf{i}))$$

is an affine combination of its **control points** $c_{\mathbf{i}}^{\Delta}$ and $c_{\mathbf{i}}^{\nabla}$ and this representation is **affinely invariant** meaning that under any affine map the images of the control points control the image of $s(\mathbf{x})$.

Since the half-box splines are non-negative, $s(\mathbf{x})$ is even a convex combination of its control points and lies in their **convex hull**.

If we connect control points $c_{\mathbf{i}}^{\Delta}$ and $c_{\mathbf{j}}^{\nabla}$ whose associated triangles $\mathbf{i} + \Delta$ and $\mathbf{j} + \nabla$ have a common edge, then we obtain a hexagonal net, the control net of s . An example is illustrated in Figure 10.19.

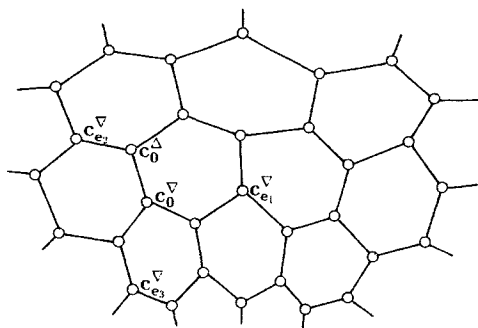


Figure 10.19. A hexagonal control net.

10.4.2. Derivatives and polynomial properties

The **directional derivative** of s with respect to \mathbf{v}_r can be computed by the derivative formula (10.10) and it is

$$D_{\mathbf{v}_r} s(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbb{Z}^s} (\nabla_{\mathbf{v}_r} \mathbf{c}_i^\Delta H_\Delta(\mathbf{x} | \mathbf{v}_1 \dots \mathbf{v}_r^* \dots \mathbf{v}_k) + \nabla_{\mathbf{v}_r} \mathbf{c}_i^\nabla H_\nabla(\mathbf{x} | \mathbf{v}_1 \dots \mathbf{v}_r^* \dots \mathbf{v}_k)) ,$$

where $\nabla_{\mathbf{v}} \mathbf{c}_i = \mathbf{c}_i - \mathbf{c}_{i-\mathbf{v}}$.

If $H_\Delta(\mathbf{x})$ is continuous or equivalently, if there are two independent directions among $\mathbf{v}_1, \dots, \mathbf{v}_k$, then all directional derivatives of the sum of all shifts, $\sum H_\Delta(\mathbf{x} - \mathbf{i})$, are zero. Therefore this sum is a constant function. Because of symmetry reasons and since the shifts of both half-box splines H_Δ and H_∇ form a partition of unity, we get

$$\sum_{\mathbf{i} \in \mathbb{Z}^2} H_\Delta(\mathbf{x} - \mathbf{i}) = \sum_{\mathbf{i} \in \mathbb{Z}^2} H_\nabla(\mathbf{x} - \mathbf{i}) = 1/2 . \tag{10.11}$$

In particular, this implies that the shifts of H_Δ and H_∇ are **linearly dependent** .

Further, if the box spline $B(\mathbf{x}) = B(\mathbf{x} | \mathbf{e}_1 \mathbf{e}_2 \mathbf{v}_1 \dots \mathbf{v}_k) = H_\Delta(\mathbf{x}) + H_\nabla(\mathbf{x})$ is continuous, we recall from (10.6) in 10.2.2 that

$$\sum_{\mathbf{i} \in \mathbb{Z}^2} \mathbf{m}_i (H_\Delta(\mathbf{x} - \mathbf{i}) + H_\nabla(\mathbf{x} - \mathbf{i})) = \mathbf{x} ,$$

where \mathbf{m}_i is the center of $\text{supp} B(\mathbf{x} - \mathbf{i})$. If H_Δ is continuous, we can use (10.11) and get for any $\mathbf{v} \in \mathbb{R}^2$

$$\sum_{\mathbf{i} \in \mathbb{Z}^2} ((\mathbf{m}_i + \mathbf{v}) H_\Delta(\mathbf{x} - \mathbf{i}) + (\mathbf{m}_i - \mathbf{v}) H_\nabla(\mathbf{x} - \mathbf{i})) = \mathbf{x} .$$

For example, if $\mathbf{v} = (\mathbf{e}_1 - \mathbf{e}_2)/6$, then the points $\mathbf{m}_i^\Delta := \mathbf{m}_i + \mathbf{v}$ and $\mathbf{m}_i^\nabla := \mathbf{m}_i - \mathbf{v}$ form a regular hexagonal grid as illustrated in Figure 10.20.

Since the half-box spline representation is affinely invariant, we obtain for any linear polynomial $l(\mathbf{x})$

$$l(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbb{Z}^2} (l(\mathbf{m}_i^\Delta) H_\Delta(\mathbf{x} - \mathbf{i}) + l(\mathbf{m}_i^\nabla) H_\nabla(\mathbf{x} - \mathbf{i})) .$$

This property is referred to as the **linear precision** of the half-box spline representation.

10.4.3. Subdivision

Any half-box spline surface

$$s(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbb{Z}^2} (\mathbf{c}_i^\Delta H_\Delta(\mathbf{x} - \mathbf{i}) + \mathbf{c}_i^\nabla H_\nabla(\mathbf{x} - \mathbf{i}))$$

has also a “finer” representation

$$s(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbb{Z}^2} (\mathbf{d}_i^\Delta H_\Delta(m\mathbf{x} - \mathbf{i}) + \mathbf{d}_i^\nabla H_\nabla(m\mathbf{x} - \mathbf{i}))$$

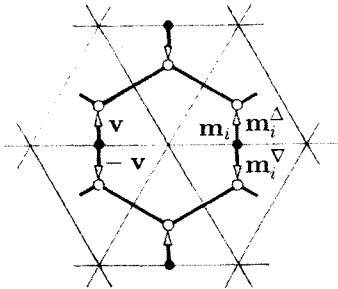


Figure 10.20. The hexagonal grid of the “centers” m_i^Δ and m_i^∇ .

for any $m \in \mathbf{N}$.

As for box splines one can derive a subdivision algorithm for the computation of the new control points $d_r^\square(\mathbf{i}) = d^r(\mathbf{i})$ from the control points c_i^\diamond , where \square and $\diamond \in \{\Delta, \nabla\}$. This algorithm, which is due to [28], is given by the recursion

$$d_r^\square(\mathbf{i}) := \frac{1}{m} \sum_{l=0}^{m-1} d_{r-1}^\square(\mathbf{i} - l\mathbf{v}_r), \quad r = 3, \dots, k.$$

$$d_2^\square(\mathbf{i}) := c_j^\diamond$$

for $\mathbf{i}, \mathbf{j} \in \mathbf{Z}^2$ and $\text{supp}H^r(m\mathbf{x} - \mathbf{i}) \subset \text{supp}H^r(\mathbf{x} - \mathbf{j})$.

One can also combine the recursion steps. For example, for $m = 2$ and for the symmetric cubic half-box splines, where $\mathbf{v}_3 \dots \mathbf{v}_k = \mathbf{e}_1 \mathbf{e}_2 \mathbf{e}_3$, the $d_5^\square(\mathbf{i})$ can be computed from the c_i^\diamond by the masks shown in Figure 10.21 and from the $d_2^\square(\mathbf{i})$ by the mask shown in Figure 10.22.

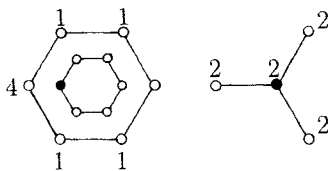


Figure 10.21. The masks to subdivide cubic C^1 -half-box spline surfaces.

As with box splines, the control points $d^r(\mathbf{i})$ converge to the half-box spline surface s with order $O(1/m^2)$.

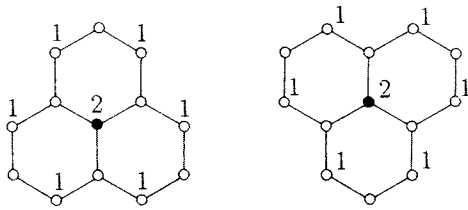


Figure 10.22. The mask for three averaging steps.

10.4.4. Bézier representation

One can compute the Bézier points of a half-box spline surface by the same recursion as for box spline surfaces [35,1,28]. The only difference is that the recursion terminates differently. For example if, e.g., $\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3 = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$, then

$$\mathbf{b}_3 = \alpha_3 * C_3\mathbf{b}_0 ,$$

where

$$C_3\mathbf{b}_0(\mathbf{j}) = \begin{cases} \mathbf{c}_i^\Delta & \text{if } \mathbf{j} = 3\mathbf{i} + \mathbf{e}_1 - \mathbf{e}_3 \\ \mathbf{c}_i^\nabla & \text{if } \mathbf{j} = 3\mathbf{i} + \mathbf{e}_2 - \mathbf{e}_3 \\ \mathbf{0} & \text{otherwise} \end{cases} .$$

This means that the Bézier points $\mathbf{b}_3(\mathbf{i})$ of a piecewise cubic C^1 -half-box spline can be computed from the hexagonal control net \mathbf{c} with the masks shown in Figure 10.23.

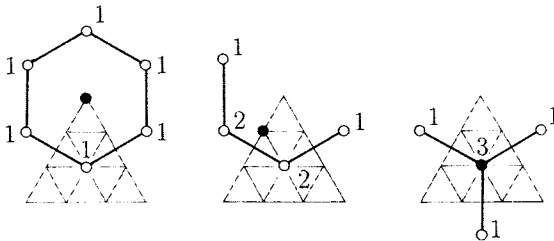


Figure 10.23. The masks for the Bézier points of the cubic half-box splines.

10.4.5. Generalized Half-Box spline surfaces

As with box splines it is possible to build arbitrary free form surfaces with the half-box splines

$$H_\Delta^k(\mathbf{x}) := H_\Delta(\mathbf{x}|\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 \cdot^k \cdot \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3) \quad \text{and} \quad H_\nabla^k(\mathbf{x}) := H_\nabla(\mathbf{x}|\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3 \cdot^k \cdot \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3) ,$$

see [38,33]. The support of a half-box spline $H_{\Delta}^k(\mathbf{x})$ consists of k rings of triangles around a triangle as shown in Figure 10.24 for $k = 1, 2$. This implies that any triangular patch of a half-box spline surface

$$s(\mathbf{x}) = \sum_{\mathbf{i} \in \mathbb{Z}^2} (\mathbf{c}_i^{\Delta} H_{\Delta}^k(\mathbf{x}) + \mathbf{c}_i^{\nabla} H_{\nabla}^k(\mathbf{x}))$$

is controlled by k rings of control points around a single control point as illustrated schematically in Figure 10.25 for $k = 1, 2$.

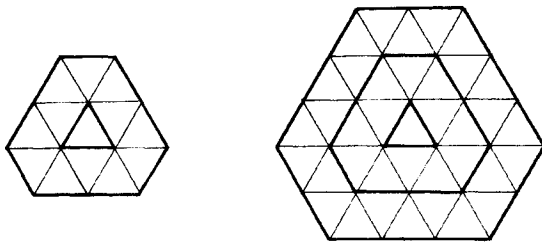


Figure 10.24. Support of the half-box splines H_{Δ}^2 (left) and H_{Δ}^3 (right).

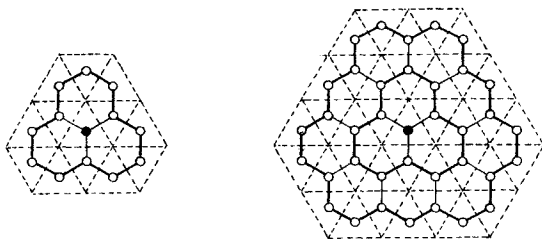


Figure 10.25. H-primitives of order 2 (left) and 3 (right).

We will call the net controlling one triangular patch a **minimal net** or a **H-primitive** of order k . Note that the H-primitives of order k are dual to the B-primitives of order $k+1$. This means that the triangles and interior vertices of the dual B-primitive correspond to the vertices and hexagonal meshes of the H-primitive.

A **generalized half-box spline surface** of order k is given by an arbitrary control net with isolated non-hexagonal meshes and consists of all triangular patches controlled by a H-primitive of order k that is part of the entire control net.

For any generalized half-box spline surface s_H of order k there is a generalized box spline surface s_B of order $k + 1$ whose control net is dual to the control net of the generalized half-box spline surface. Hence, s_H and s_B have the same number of triangular patches and these have the same topological adjacencies.

In particular, this is true for nets with multiple control points. Consequently, if $k - 1$ rings of control points around an m -sided mesh coalesce, then the corresponding half-box spline surface has a hole whose boundary is given by m triangular patches.

Figure 10.26 shows a generalized half-box spline surface (middle) of order $k = 1$ with its control net (left) and with a G^1 -filling (right).

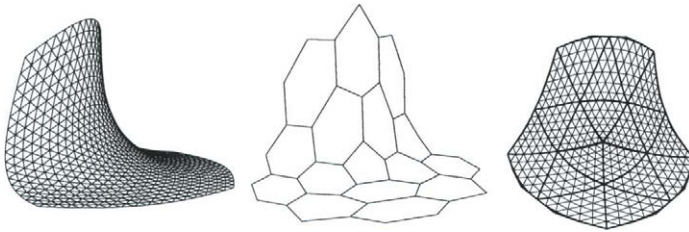


Figure 10.26. A generalized half box spline surface with smooth filling (left), control net (middle) and Bézier net (right). (courtesy of Markus Florenz and Georg Umlauf)

REFERENCES

1. W. Boehm. Generating the Bézier points of triangular splines. In R.E. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 77–91. Elsevier Science Publishers B.V. (North-Holland), 1983.
2. W. Boehm. The de Boor algorithm for triangular splines. In R.E. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, pages 109–120. Elsevier Science Publishers B.V. (North-Holland), 1983.
3. W. Boehm. Subdividing multivariate splines. *Computer-Aided Design*, 15:345–352, 1983.
4. W. Boehm. Calculating with box splines. *Computer Aided Geometric Design*, 1:149–162, 1984.
5. W. Boehm, H. Prautzsch, and P. Arner. On triangular splines. *Constr. Approx.*, 3:157–167, 1987.
6. C. de Boor and R. DeVore. Approximation by smooth multivariate splines. *Trans. Amer. Math. Soc.*, 276:775–788, 1983.
7. C. de Boor. On the evaluation of box splines. *Numer. Algorithms*, 5:5–23, 1993.

8. C. de Boor and K. Höllig. Recurrence relations for multivariate B-splines. *Proc. Amer. Math. Soc.*, 85:397–400, 1982.
9. C. de Boor and K. Höllig. B-splines from parallelepipeds. *J. Analyse Math.*, 42:99–115, 1982.
10. C. de Boor, K. Höllig, and S. Riemenschneider. *Box Splines*, Springer-Verlag, 1993.
11. G.M. Chaikin. An algorithm for high speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.
12. E. Cohen, T. Lyche, and R. Riesenfeld. Discrete Box splines and refinement algorithms. *Computer Aided Geometric Design*, 1:131–148, 1984.
13. M. Dæhlen. On the evaluation of Box-splines. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 167–179. Academic Press, N.Y, 1989.
14. W. Dahmen. Subdivision algorithms converge quadratically. *J. Comput. Appl. Math.*, 16:145–158, 1986.
15. W. Dahmen, N. Dyn, and D. Levin. On the convergence rates of subdivision algorithms for box spline surfaces. *Constr. Approx.*, 1:305–322, 1985.
16. W. Dahmen and C.A. Micchelli. Translates of multivariate splines. *Linear Algebra Appl.*, 52:217–234, 1983.
17. W. Dahmen and C.A. Micchelli. Subdivision algorithms for the generation of box-spline surfaces. *Computer Aided Geometric Design*, 1:115–129, 1984.
18. W. Dahmen and C.A. Micchelli. Line average algorithm: a method for the computer generation of smooth surfaces. *Computer Aided Geometric Design*, 2:77–85, 1985.
19. W. Dahmen and C.A. Micchelli. On the local linear independence of translates of a box spline. *Studia Math.*, 82:243–262, 1985.
20. W. Dahmen and C.A. Micchelli. Convexity of multivariate Bernstein polynomials and box spline surfaces. *Studia Math.*, 23:265–287, 1988.
21. P.O. Frederickson. *Triangular spline interpolation*. Rpt.6 70, Lakehead Univ, 1970.
22. P.O. Frederickson. *Generalized triangular splines*. Rpt.7 71, Lakehead Univ, 1971.
23. Rong-qing Jia. Linear independence of translates of a box spline. *J. Approx. Theory*, 40:158–160, 1984.
24. Rong-qing Jia. Local linear independence of the translates of a box spline. *Constr. Approx.*, 1:175–182, 1985.
25. L. Kobbelt. Stable evaluation of box splines. *Numerical Algorithms*, 14(4):377–382, 1997.
26. J.M. Lane and R.F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Trans. Pattern Anal. Mach. Intellig.*, 2:35–45, 1980.
27. C.T. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.
28. H. Prautzsch. Unterteilungsalgorithmen für multivariate Splines – ein geometrischer Zugang. Dissertation, Univ. Braunschweig, 1984.
29. H. Prautzsch. Generalized subdivision and convergence. *Computer Aided Geometric Design*, 2:69–75, 1985.
30. H. Prautzsch. The location of the control points in the case of box splines. *IMA J. Numer. Anal.*, 6:43–49, 1986.

31. H. Prautzsch. On convex Bézier triangles. *Mathematical Modelling and Numerical Analysis*, 26:23–36, 1992.
32. H. Prautzsch. Freeform splines. *Computer Aided Geometric Design*, 14:201–206, 1997.
33. H. Prautzsch and G. Umlauf. Triangular G^2 splines, In Laurent et.al., editors, *Curves and Surface Design*, pages 335–342. Vanderbilt University Press, 1999.
34. G. de Rham. Un peu de mathématique à propos d'une courbe plane. *Elem. Math.*, 2:73–76,89–97, 1947.
35. M.A. Sabin. *The Use of Piecewise Forms for the Numerical Representation of Shape*. Dissertation, MTA Budapest, 1977.
36. I.J. Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions, Part A: On the problem of smoothing of graduation, a first class of analytic approximation. *Quart. Appl Math.*, 4:45–99, 1946.
37. A. Sommerfeld. Eine besondere anschauliche Ableitung des Gaussischen Fehlergesetzes. In Geburtstage, Verlag von J.A. Barth, *Festschrift Ludwig Boltzmann gewidmet zum 60*, pages 848–859. Leipzig, 1904.
38. G. Umlauf. *Glatte Freiformflächen und Optimierte Unterteilungsflächen*. Dissertation, Univ. Karlsruhe, 1999.

Chapter 11

Finite Element Approximation with Splines

Klaus Höllig

We describe in this chapter finite element methods with uniform B-splines. These techniques do not require any grid generation and yield highly accurate approximations with relatively few parameters. They are particularly well suited in combination with spline-based geometric modeling systems.

11.1. INTRODUCTION

The finite element method has become the most widely accepted general purpose technique for a broad range of applications in continuum mechanics, fluid dynamics, field theory and other areas in engineering and mathematical physics (cf., e.g., [33,31,13]). While the label "finite element method" was first used by Clough [14], the key ideas date back much further. Ritz [39] described how to solve variational problems with finite dimensional approximations, an approach already employed by Rayleigh [38]. Following an observation of Bubnow [11], Galerkin [23] approximated differential equations for boundary value problems directly, without resorting to the variational formulation. Courant [16] was the first to use piecewise linear hat-functions, the standard "finite element" in his discussion of the St. Venant torsion problem. The systematic use of variational approximations in engineering applications began much later with the work of Turner, Clough, Martin and Topp [49], and Argyris [2,3]. While these are perhaps the most well-known foundations of finite element analysis, there are many other contributions, and we refer to [35] for a survey of the extensive literature.

The basic principle of finite element methods can be illustrated for the Poisson equation on a bounded domain with homogeneous Dirichlet boundary conditions,

$$-\Delta u = f \text{ in } D \subset \mathbb{R}^m, \quad u = 0 \text{ on } \partial D. \quad (11.1)$$

A weak solution of this model problem can be characterized as the minimum of the

functional

$$\frac{1}{2} \int_D \text{grad } v \text{ grad } v - \int_D f v, \quad v \in V, \quad (11.2)$$

or, equivalently, by the equations

$$\int_D \text{grad } u \text{ grad } v = \int_D f v, \quad \forall v \in V, \quad (11.3)$$

where $V = H_0^1(D)$ is the Sobolev space of functions which vanish on the boundary and have square integrable first derivatives. Both characterizations suggest the following natural discretization. We replace $H_0^1(D)$ by finite dimensional spaces V_h (finite element subspaces), which contain close approximations to u as their dimension increases. This leads to special cases of the methods of Ritz and Galerkin, respectively. Obviously, both methods are very general and apply, with appropriate modifications, to virtually any elliptic boundary value problem.

Since the early beginnings in the sixties, numerous variants of the finite element method have been developed and become well established [12,50]. Most techniques use a mesh of the simulation domain (partition into tetrahedra, hexahedra, etc.) to construct the approximating subspaces. For complicated domains, the generation of such meshes requires often the major portion of the computing time and implementing good algorithms is a challenging task [29,36]. Moreover, on unstructured meshes, higher order approximations lead to huge systems. Therefore, considerable efforts have been made to develop meshless methods, to a large extent building upon Babuška's classical ideas [4,5] (cf., e.g., the surveys [6,7]). In particular for complicated domains, such techniques can be more efficient than standard approximations.

We describe in this chapter meshless methods, which use weighted finite element subspaces. For example, a solution of (11.1) is approximated by

$$u \approx w p, \quad p \in S,$$

where w is a fixed positive function, which vanishes on ∂D , and S is a suitable linear space. Such weighted approximations were already suggested by Kantorovich [32]. They have become particularly successful in connection with Rvachev's R -functions method (cf. the survey [40] for an overview), which automatizes the construction of weight functions. The use of B-splines as basis for S suggests itself and was considered, e.g., in [46,41]. With appropriate modifications [26], the resulting finite element subspaces possess all the standard approximation properties. Numerical solutions can be computed very efficiently, in particular with the aid of software for manipulating B-splines [9] and tools from geometric modeling [17,18,30].

We use the following notational conventions. Dependencies on parameters are not always indicated, if they are clear from the context. For example, $b_k = b_{k,h}^n$ denotes the tensor product B-spline defined in Subsection 11.2.1. Moreover, we write $f \preceq g$, if $f \leq c g$ with a constant c , which does not depend on the grid width h , indices, or arguments of functions. The symbols \succeq and \asymp are defined analogously. Finally, $\| \cdot \|$ denotes the 2-norm for vectors and matrices and

$$\|u\|_{\ell,D} := \left(\sum_{\alpha_1 + \dots + \alpha_m \leq \ell} \int_D |\partial^\alpha u(x)|^2 dx \right)^{1/2}, \quad \partial^\alpha := \left(\frac{\partial}{\partial x_1} \right)^{\alpha_1} \cdots \left(\frac{\partial}{\partial x_m} \right)^{\alpha_m}$$

the norm on the Sobolev spaces $H^\ell(D)$ [1]. We assume throughout that the boundary ∂D is piecewise smooth and has the cone property, i.e., there are no cusp-like singularities.

11.2. SPLINES ON UNIFORM GRIDS

Uniform splines, discussed in this section, are spanned by scaled translates of a single B-spline (cf. Chapter 6 by C. de Boor). Algorithms are particularly efficient and elegant, and there are a number of beautiful theoretical results originating from the classical work of Schoenberg [43] (cf. also [10] and Chapter 10 by H. Prautzsch and W. Böhm).

Unlike for univariate splines, general knot sequences do not permit local refinement in several variables. Hence, from a practical point of view, there is not a very significant advantage over uniform knots. Adaptive refinement is possible in both cases with the aid of hierarchical bases and wavelet constructions.

11.2.1. Uniform B-splines

Uniform B-splines are special cases of the general B-splines $B_{k,n,t}$, described in Chapter 6 by C. de Boor, and correspond to the uniform knot sequence $t = \dots, -h, 0, h, \dots$. They are scaled translates of the standard cardinal B-spline, which can be constructed with a particularly simple averaging process.

Definition 11.2.1 *The standard uniform B-spline of order $n > 1$ is defined by the recursion*

$$b^n(x) := \int_{x-1}^x b^{n-1},$$

starting with the characteristic function b^1 of the unit interval $[0, 1]$.

With the first averaging step, we obtain the piecewise linear hat-function b^2 , which equals 1 at $x = 1$ and vanishes outside $(0, 2)$. One more average yields the piecewise quadratic B-spline b^3 with support $(0, 3)$. In general, b^n is a positive piecewise polynomial of degree $< n$, which is $(n - 2)$ -times continuously differentiable at the integers and vanishes outside $(0, n)$.

Forming products of uniform B-splines, we obtain multivariate B-splines on tensor product grids.

Definition 11.2.2 *A normalized uniform m -variate B-spline of order n , grid width h , and shift $k = (k_1, \dots, k_m) \in \mathbb{Z}^m$ is defined by*

$$b_{k,h}^n(x) := h^{-m/2} \prod_{\nu=1}^m b^n(x_\nu/h - k_\nu), \quad x = (x_1, \dots, x_m) \in \mathbb{R}^m.$$

As is illustrated in Figure 11.1, the B-spline b_k is a scaled and translated normalized univariate B-spline in each coordinate direction. In particular, b_k has support

$$kh + (nQ_*)h, \quad Q_* := (0, 1)^m,$$

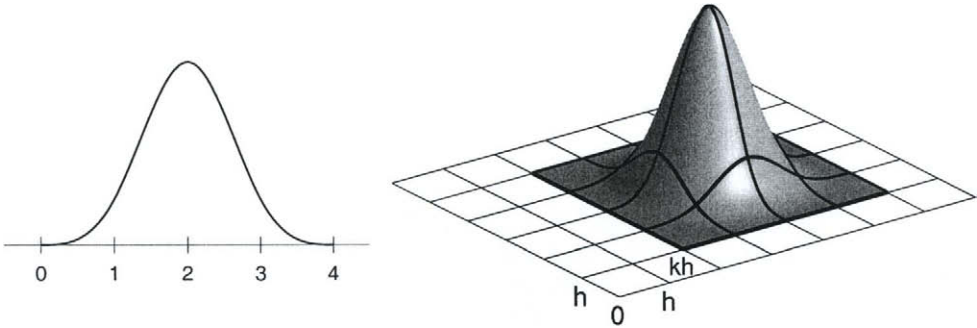


Figure 11.1. The cubic B-spline b^4 and a corresponding bivariate B-spline $b^4_{(2,1),h}$.

and is a polynomial of coordinate degree $< n$ on each grid cell $Q = lh + Q_*h$. The normalizing factor $h^{-m/2}$ ensures that the B-splines are uniformly bounded with respect to the L_2 -norm, i.e.,

$$\int_{\mathbb{R}^m} |b_k|^2 \asymp 1.$$

More generally, $\|b_k\|_\ell \asymp h^{-\ell}$ for $\ell < n$. This normalization is particularly convenient for finite element applications, so that we prefer it to the standard choice of simple scaling in this context.

11.2.2. Splines on bounded domains

Splines are linear combinations of B-splines. The following more precise definition will be used for constructing finite element subspaces in the next section.

Definition 11.2.3 *The splines $S(D)$ consist of the linear combinations*

$$\sum_{k \in K} c_k b_k(x),$$

where the sum is taken over all relevant shifts k , i.e., all k , for which b_k has some support in D .

Definition 11.2.3 is illustrated in Figure 11.2 for quadratic splines ($n = 3$), where we have marked the lower left corners kh , $k \in K$, of the supports of the relevant B-splines. Apparently, depending on the shape of the domain D , the set of relevant shifts k may be complicated to describe explicitly. Therefore, in the implementation of algorithms, we will always allocate storage for an m -dimensional array of coefficients and simply mark the positions corresponding to the relevant shifts $k \in K$. This is much more efficient than storing precise lists.

It is important to note that the B-spline basis for $S(D)$ is not uniformly stable with respect to the grid width h . This is due to the outer B-splines

$$b_j, \quad j \in J$$

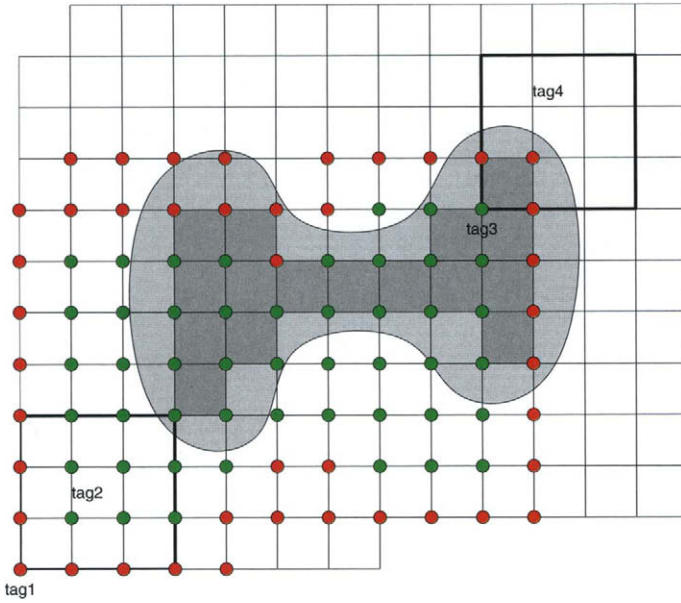


Figure 11.2. Relevant B-splines b_k , spanning $S(D)$, and classification into inner B-splines b_i and outer B-splines b_j .

(marked with red dots in Figure 11.2), for which no grid cell in their support is completely contained in D . The distinction between outer and inner B-splines

$$b_i, \quad i \in K \setminus J$$

(marked with green dots in the Figure), which have at least one grid cell in D , will be essential for stabilizing the basis (cf. Subsection 11.3.2).

Splines have the local approximation power of polynomials, as stated in the following basic error estimate [44].

Theorem 11.2.1 *For any $u \in H^k(D)$ there exists an approximation $u_h \in S(D)$ with*

$$\|u - u_h\|_{\ell, D} \leq h^{k-\ell} \|u\|_{k, D}$$

for $\ell < k \leq n$.

The approximation u_h can be constructed by various methods, for example with standard quasi-interpolation techniques.

11.2.3. Hierarchical bases

To resolve small geometric details or rapid changes in solutions, local grid refinement is essential. There are numerous techniques, in particular in connection with wavelets

(cf. Chapter 14 by L. Kobbelt). As an example, we describe an elementary strategy for hierarchical subdivision of B-splines. This technique is very easy to implement and well suited for spline-based finite element approximations.

The hierarchical B-spline basis corresponds to a nested sequence of domains D_ν , which specifies where the grid should be refined. The following natural selection of the basis functions, based on the standard subdivision schemes for splines [8,15], was proposed in [34].

Definition 11.2.4 *The hierarchical spline space $S(\mathbb{D})$, corresponding to the domains*

$$\mathbb{D}: D = D_0 \supset D_1 \supset D_2 \supset \cdots \supset D_\ell = \emptyset,$$

is spanned by the B-splines

$$b_{k,h_\nu}, \quad k \in K_\nu, h_\nu := 2^{-\nu}h, \quad 0 \leq \nu < \ell$$

where K_ν denotes the shifts k , for which $D \cap \text{supp } b_{k,h_\nu}$ is a nonempty subset of D_ν , but is not contained in $D_{\nu+1}$.

This definition can be interpreted in the following way. We select a subset of the relevant B-splines for D with grid width h and replace it by B-splines of grid width $h/2$ via subdivision. From the B-splines on the finer grid, again, a subset is selected and refined. This process is repeated according to the sequence of domains D_ν .

It is easily checked that the B-splines, spanning $S(\mathbb{D})$, are linearly independent. If

$$\sum_{\nu < \ell} \sum_{k \in K_\nu} c_{k,\nu} b_{k,h_\nu}(x) = 0 \quad \forall x \in D,$$

we can show inductively, for $\nu = 0, 1, \dots$, that the coefficients $c_{k,\nu}$ are zero. First, we restrict x to the open set $D_{0,1} := D_0 \setminus \overline{D_1}$. By Definition 11.2.4, all B-splines with grid width h_ν , $\nu > 0$, vanish on this set. On the other hand, for each B-spline b_{k,h_0} , $k \in K_0$, there exists a point $x \in D_{0,1}$, where this B-spline is nonzero. Hence, by the local linear independence of the B-splines corresponding to one grid, all coefficients $c_{k,0}$ must be zero. Now, we repeat the argument, restricting x successively to the sets $D_{1,2}, D_{2,3}, \dots$

The hierarchical spline space $S(\mathbb{D})$ contains for each domain D_ν all B-splines b_{k,h_ν} , for which the portion of their support in D is completely contained in D_ν . Either such a B-spline belongs to the hierarchical basis, or, if $D \cap \text{supp } b_{k,h_\nu} \subset D_{\nu+1}$, it can be represented as a linear combination of B-splines with smaller grid width via subdivision. Hence, as is to be expected, the local approximation power of $S(\mathbb{D})$ corresponds to the level of refinement.

Figure 11.3 illustrates Definition 11.2.4 for bilinear splines with 3 grids. The subdomains $D_2 \subset D_1 \subset D_0 = D$ are displayed with different gray levels, and the lower left corners of the supports

$$kh_\nu + (0, 2)^2 h_\nu$$

of the B-splines in the basis are marked according to their level. For example, a blue/green dot corresponds to two B-splines with grid-widths h_0 and $h_1 = h_0/2$, respectively.

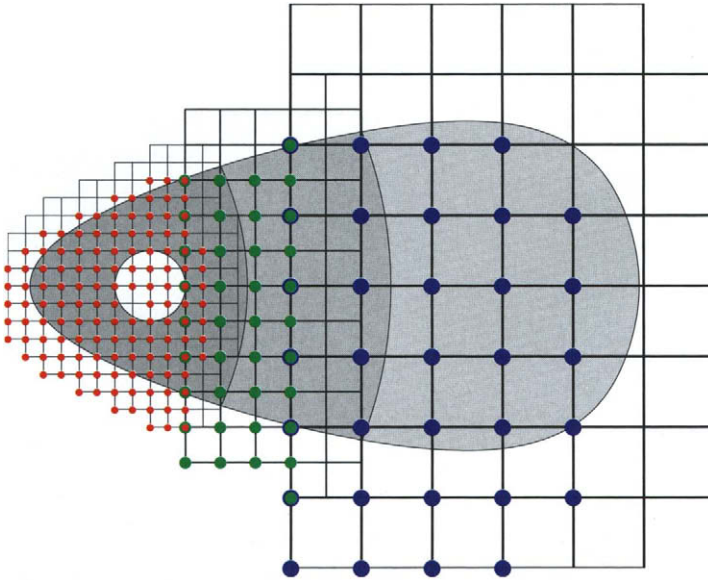


Figure 11.3. Bilinear hierarchical B-spline basis.

11.3. FINITE ELEMENT BASES

In this section, we describe several types of weighted finite element subspaces, as proposed, e.g., in [40,46,26]. The essential difference, compared to standard mesh-based subspaces, is the treatment of boundary conditions. These are incorporated via weight functions, so that no mesh generation is required. As a result, we obtain finite elements on uniform grids, which share all the computational advantages of B-spline representations.

11.3.1. Mesh-based elements

Most of the standard finite element bases are constructed using a mesh of the domain. Figure 11.4 gives examples of triangular meshes in two and three dimensions, constructed with the ART-algorithm of A. Fuchs [20,21]. This method uses a physically motivated optimization strategy to achieve an almost regular mesh structure, i.e., almost all interior vertices have the same number of neighbors.

Generating finite element meshes is often the bottleneck in finite element simulations. Especially for three-dimensional problems, the construction of meshes with good geometric properties is a highly nontrivial task. Among the many algorithms, which have been developed, one can distinguish between three major approaches: advancing front techniques, domain decomposition and Delaunay triangulations. All methods have their pros and cons and, which one to choose, depends very much on the particular application.

The construction of finite element bases with respect to a given mesh is straightforward,

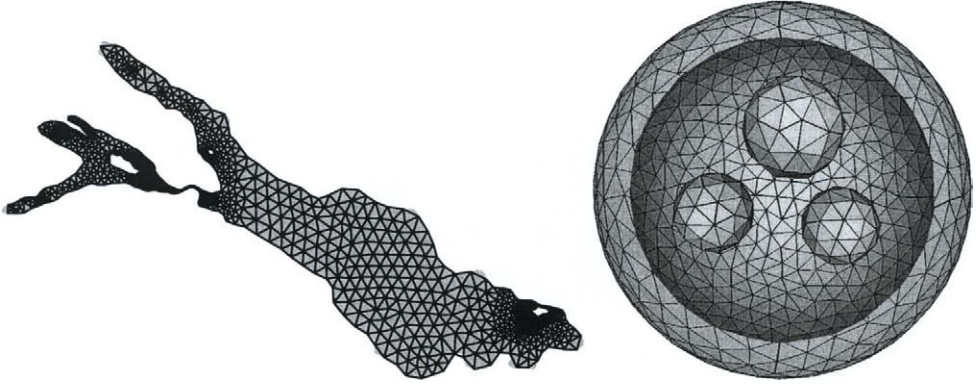


Figure 11.4. Triangular and tetrahedral finite element meshes.

at least if no continuous derivatives are required. Figure 11.5 lists some commonly used elements for triangular meshes. They are described in terms of the parameters which define the finite element approximation on a single triangle. For example, the standard linear element (degree 1) is determined by the 3 values at the vertices (dimension 3), indicated by dots. The corresponding piecewise linear approximations are continuous, hence belong to H^1 . Higher order elements can be defined by specifying more interpolation points, which are arranged in a regular triangular array. The quadratic case is shown in the Figure. To construct elements which join continuously differentiable across edges is more difficult. A classical example of a H^2 -element is Argyris' triangle. It has degree 5 (dimension 21) and is determined by specifying values, first, and second derivatives at the corners (marked with dots, small, and large circles) and normal derivatives at the mid-points of the edges (marked with orthogonal bars). An alternative is the Clough-Tocher macro-element, which achieves H^2 -smoothness with degree 3 by splitting the triangle at the centroid.

Figure 11.5 shows only a very small selection of the available possibilities. We refer to [12,50] for other examples and for quadrilateral and trivariate elements. No attempt is made here to analyze mesh-based elements in more detail. The brief description was included merely for comparison purposes. As is apparent from the bivariate examples given in the Figure, higher order or smooth elements require many parameters. In contrast, elements constructed with B-splines, as discussed in the next subsection, yield smooth highly accurate approximations with relatively few parameters. Moreover, no mesh generation is necessary, so that computing times are significantly reduced.

11.3.2. WEB-basis

At first sight, the use of B-splines as finite element basis functions seems not feasible, because the uniform grid does not conform to the boundary. However, this difficulty is easily overcome. To satisfy essential homogeneous boundary conditions, we multiply with

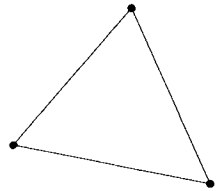
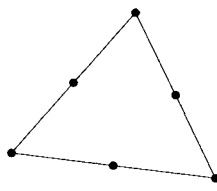
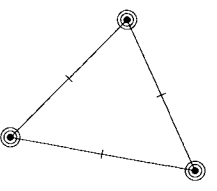
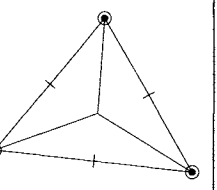
			
<p>linear element</p> <p>deg = 1, $\in H^1$, dim = 3.</p>	<p>quadratic element</p> <p>deg = 2, $\in H^1$, dim = 6.</p>	<p>Argyris</p> <p>deg = 5, $\in H^2$, dim = 21.</p>	<p>Clough-Tocher</p> <p>deg = 3, $\in H^2$, dim = 12.</p>

Figure 11.5. Some bivariate triangular finite elements.

a positive weight function w , which vanishes on ∂D , i.e., we approximate with the space

$$S_w := wS = \text{span}_{k \in K} w b_k,$$

spanned by weighted relevant B-splines. For example, for a smooth domain, we may use a weight function, which is equivalent to the distance to the boundary, i.e.

$$w(x) \asymp \text{dist}(x, \partial D). \tag{11.4}$$

There are a number of other possibilities, in particular for domains, which can be described in terms of simple primitives (lines, planes, circles, cylinders, etc.). A systematic approach will be discussed in the next subsection.

As we already remarked in Subsection 11.2.2, the B-spline basis is not uniformly stable when restricted to D . This may lead, e.g., to excessively large condition numbers of finite element systems. A well conditioned basis can be constructed with the aid of the classification of the relevant B-splines b_k for the domain D into inner and outer B-splines (cf. Figure 11.2). To stabilize the basis, the outer B-splines b_j , for which, by definition, no mesh cell of their support is contained in D , are attached to the inner B-splines b_i . This has to be done in such a way, that polynomial precision, which guarantees full approximation order, is preserved. Hence, let us consider the B-spline representation

$$p(x) = \sum_{k \in K} q(k) b_k(x), \quad x \in D \tag{11.5}$$

of a polynomial p of order n on D . As is well-known, q is a polynomial of the same order as p (cf., e.g., [10] for a proof in a more general context). Therefore, we can compute any coefficient $q(j)$, $j \in J$, from n^m coefficients $q(i)$, corresponding to an array of shifts

$$I(j) := \ell + \{1, \dots, n\}^m \subset I, \quad (\ell = \ell(j))$$

closest to j . To compute $q(j)$, we interpolate the values $q(i)$ at the shifts in $I(j)$ and evaluate the interpolant at j . Hence, if

$$e_{i,j} := \prod_{\mu=1}^m \prod_{\substack{\alpha=1 \\ \alpha \neq i_\mu - \ell_\mu}}^n \frac{j_\mu - \ell_\mu - \alpha}{i_\mu - \ell_\mu - \alpha}$$

denotes the value at j of the Lagrange polynomial associated with $i \in I(j)$, we have

$$q(j) = \sum_{i \in I(j)} e_{i,j} q(i).$$

Inserting the expression for $q(j)$ into (11.5) and interchanging sums gives

$$p(x) = \sum_{i \in I} q(i) \left[b_i(x) + \sum_{j \in J(i)} e_{i,j} b_j(x) \right], \quad x \in D,$$

where $J(i) \subset J$ is the set of all shifts j , for which $i \in I(j)$. This identity suggests to define the term in square brackets as the proper extension of the inner B-spline b_i . Of course, we have to multiply by the weight function w and choose an appropriate normalization. These considerations led to the following definition [26] (cf. also <http://www.web-spline.de>).

Definition 11.3.1 *The weighted extended B-splines (web-splines)*

$$B_i := \frac{w}{w(x_i)} \left[b_i + \sum_{j \in J(i)} e_{i,j} b_j \right], \quad i \in I,$$

with x_i the center of an interior grid cell $Q_i \subset D$ in the support of b_i form a basis for the web-space S_{we} .

Not all finite element simulations require subspaces that conform to boundary conditions. If no weight function is used, we denote the span of the extended B-splines by S_e . Formally, this corresponds to the case $w \equiv 1$ in Definition 11.3.1.

Figure 11.6 illustrates the construction of the web-basis for quadratic splines ($n = 3$). On the left-hand side, the values of the coefficients $e_{i,j}$ with $i \in I(j)$ are given for an outer B-spline b_j . They are associated with the lower left corners of the supports of the inner B-splines b_i . The zeros in the first and second column indicate that this outer B-spline is actually involved only in 3 web-splines B_i . This is because $j_1 = i_1$ for the right column of the shifts $i \in I(j)$, causing the Lagrange polynomials associated with the other shifts in $I(j)$ to vanish at j . On the right-hand side, the support of a web-spline B_i is shown together with the coefficients $e_{i,j}$ of the adjoined outer B-splines b_j , $j \in J(i)$. The point x_i is marked by a cross.

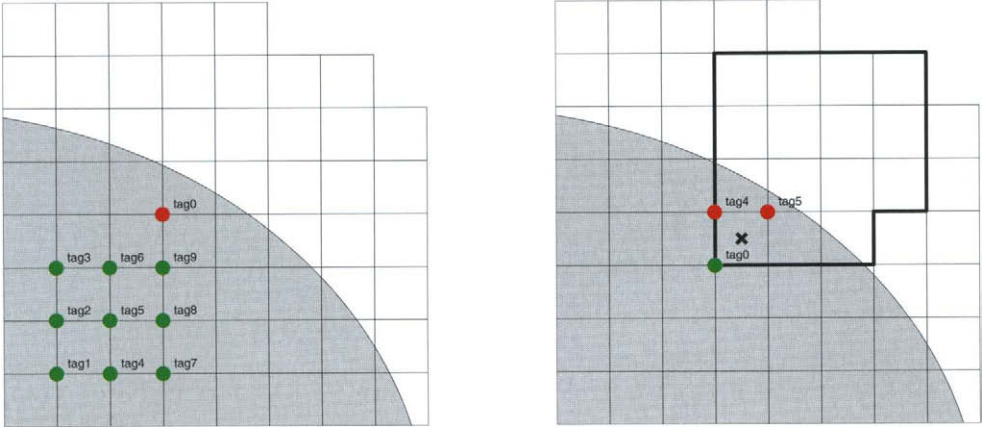


Figure 11.6. Coefficients $e_{i,j}$, $i \in I(j)$ (left) and support of a web-spline B_i with coefficients $e_{i,j}$, $j \in J(i)$ (right).

11.3.3. R-functions

Many simple domains admit ad-hoc definitions of weight functions w . For example, for an annulus with radii 1 and 2, we may set

$$w = f_0 f_1, \quad f_0(x) = x_1^2 + x_2^2 - 4, \quad f_1(x) = 1 - x_1^2 - x_2^2.$$

However, multiplying functions of implicit equations for boundary curves is not always possible. If we define

$$w(x) = f_1(x)x_1x_2$$

for the domain on the right of Figure 11.7, all functions in the finite element space S_w vanish on the coordinate axes. This imposes an undesirable constraint on the approximations. Fortunately, there exists a systematic approach, based on Rvachev’s concept of R -functions (cf., e.g., [40,45]). We describe below a special case of this fairly general theory, which is adequate for illustration purposes.

Definition 11.3.2 *A function $r : \mathbb{R}^\ell \rightarrow \mathbb{R}$ is an R -function, if its sign depends only on the sign of its arguments.*

R -functions are closely related to Boolean functions. In fact, for any Boolean set operation \circ there exist associated R -functions r_\circ , which define the corresponding operation on weight functions. In other words, if w_ν is a weight function for a set D_ν , then

$$(w_1 \circ_R w_2)(x) := r_\circ(w_1(x), w_2(x))$$

is a weight function for $D_1 \circ D_2$. A popular choice of associated R -functions is shown in the following table.

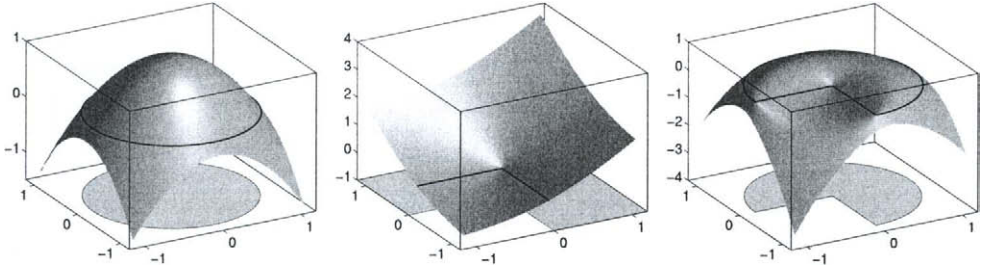


Figure 11.7. Domain description with R -functions.

set operation	corresponding R -function
$D_1 \cap D_2$	$r_{\cap}(x) = x_1 + x_2 - \sqrt{x_1^2 + x_2^2}$
$D_1 \cup D_2$	$r_{\cup}(x) = x_1 + x_2 + \sqrt{x_1^2 + x_2^2}$
D^c	$r_c(x) = -x$

Returning to the discussion of the domain

$$D = D_1 \cap (D_2 \cup D_3), \quad D_{\nu} := \{x \in \mathbb{R}^2 : f_{\nu}(x) > 0\}$$

with $f_2(x) = x_1$ and $f_3(x) = x_2$, we can define

$$w(x) = (f_1 \cap_R (f_2 \cup_R f_3))(x) = r_{\cap}(f_1(x), r_{\cup}(f_2(x), f_3(x))),$$

as is illustrated in Figure 11.7. The function f_1 is depicted on the left, $r_{\cup}(f_2(x), f_3(x))$ in the middle, and finally w on the right of the Figure.

The explicit form of the weight function can be complicated. However, for computations this is irrelevant. Evaluation and differentiation is performed using the algorithmic definition with the aid of automatic differentiation [37,24]. The procedure is similar to algorithms in constructive solid geometry.

For general domains, described, e.g., by NURBS-representations [30,19], numerical techniques have to be used. A straightforward construction of a weight function is illustrated in Figure 11.8. The distance function is used near the boundary, where it is free of singularities, and blended smoothly with a plateau inside the domain. More precisely, we define

$$w(x) = 1 - (\max(\delta - \text{dist}(x, \partial D), 0)/\delta)^{\gamma}, \quad (11.6)$$

where δ controls the size of the plateau and γ the smoothness. The plateau facilitates the use of precomputed values when assembling finite element matrices. However, it should not be chosen too large in order to keep the derivatives of the weight function small.

As is apparent from the above examples, there is a great deal of flexibility in the construction of weight functions. In particular, spline approximations and numerical techniques can be effectively combined with the R -functions method.

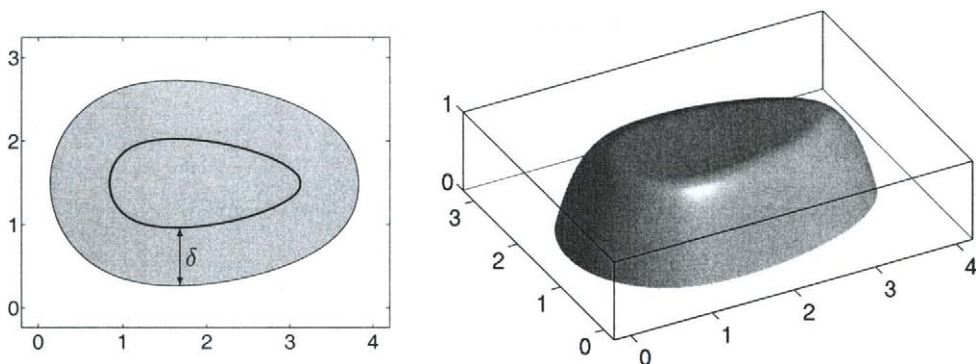


Figure 11.8. Smoothed distance function: $w(x) \asymp \text{dist}(x, \partial D)$.

11.3.4. Stability

With appropriate normalization, the norm of a finite element approximation is equivalent to the corresponding norm of the coefficients of the basis functions. This important stability property is also valid for the web-spaces S_{we} under mild assumptions on the weight function. For example, we may require that w is smooth and

$$\|\text{grad } w(y)\| \preceq w(x)/\text{dist}(x, \partial D), \quad \|x - y\| \preceq \text{dist}(x, \partial D). \tag{11.7}$$

Essentially, this condition implies that the norm of the gradient vanishes with one order less at the boundary than the weight function. It is valid for powers of the weight function (11.6) and also for products of such weight functions, constructed from distance functions to different boundary segments. Of course, (11.7) also holds for the trivial weight function $w \equiv 1$ formally corresponding to the spaces S_e .

Stability can be proven with the aid of dual functions λ_ℓ , which are biorthogonal to the B-splines b_i and uniformly bounded,

$$\int_{\mathbb{R}^m} \lambda_\ell b_i = \delta_{\ell,i}, \quad \|\lambda_\ell\|_0 \preceq 1. \tag{11.8}$$

There is a great deal of flexibility in the construction of such dual functions. In particular, we may require that the support of λ_ℓ is contained in the subcell

$$Q'_\ell := x_\ell + [-h/4, h/4]^m$$

of the grid cell $Q_\ell \subset \text{supp } b_\ell \cap D$.

We claim that (11.8) remains valid if we replace b_i by the web-splines B_i and λ_ℓ by the weighted dual functions

$$\Lambda_\ell(x) := \frac{w(x_\ell)}{w(x)} \lambda_\ell.$$

The biorthogonality follows easily since none of the outer B-splines b_j has support on any of the grid cells $Q_\ell \subset D$. By Definition 11.3.1, this implies

$$B_i(x) = \frac{w(x)}{w(x_i)} b_i(x), \quad x \in Q_\ell,$$

and therefore

$$\int_{\mathbb{R}^m} \Lambda_\ell B_i = \frac{w(x_\ell)}{w(x_i)} \int_{Q'_\ell} \lambda_\ell b_i.$$

For the uniform boundedness we observe that

$$\frac{w(x_\ell)}{w(x)} = 1 + \frac{w(x_\ell) - w(x)}{w(x)} \preceq 1 + h/\text{dist}(x, \partial D) \preceq 1, \quad x \in Q'_\ell,$$

so that $\|\Lambda_\ell\|_0 \preceq \|\lambda_\ell\|_0$. The first inequality follows from the assumption (11.7), noting that for any point y on the line segment between x_ℓ and x

$$\|x_\ell - x\| \preceq h \preceq \text{dist}(x, \partial D)$$

since x lies in a subcell of the grid cell $Q_\ell \subset D$.

Similarly, we can show that $h^\ell \|B_i\|_\ell \preceq 1$ for $\ell = 0, 1$. In view of the local support of the web-splines and the dual functions, this implies the following stability result.

Theorem 11.3.1 *For any linear combination $u_h = \sum_{i \in I} u_i B_i$ of web-splines with coefficient vector U ,*

$$h \|u_h\|_1 \preceq \|U\| \asymp \|u_h\|_0.$$

We emphasize that this Theorem is in general false for the spaces S and S_w , spanned by B-splines and weighted B-splines, respectively. For these spaces, the instability of the basis may lead to excessively large condition numbers of Galerkin systems and very slow convergence rates for iterative methods.

11.4. APPROXIMATION OF BOUNDARY VALUE PROBLEMS

Using weighted spline spaces, elliptic boundary value problems can be approximated in a familiar fashion [12]. We illustrate this by considering several typical model problems. In particular, the treatment of different types of boundary conditions is discussed since this affects the construction of the finite element subspaces. Finally, we state a basic error estimate and sketch the implementation of spline-based finite element techniques.

11.4.1. Essential boundary conditions

Essential boundary conditions have to be incorporated in the finite element subspaces since they are not automatically satisfied by solutions of appropriate variational problems. As a model problem, we consider the Poisson equation with homogeneous Dirichlet boundary conditions (11.1), mentioned in the introduction. We can construct approximations

$$u_h := \sum_i u_i B_i,$$

choosing either one of the spaces $S_w(D)$, $S_{we}(D)$, or the weighted hierarchical spline space $S_w(\mathbb{D})$ as finite element subspace V_h . Both the minimization of the functional (11.2) and the equations (11.3) with V replaced by V_h lead to the finite element equations

$$GU = F, \quad g_{\ell,i} := \int_D \text{grad } B_\ell \text{ grad } B_i, \quad f_\ell := \int_D f B_\ell \tag{11.9}$$

for the vector U of coefficients u_i .

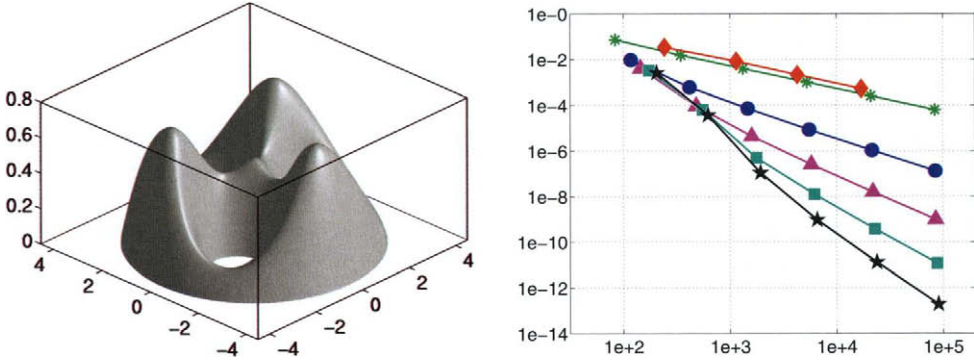


Figure 11.9. Displacement (magnified) of a membrane with constant load and error of piecewise linear and web-spline approximations of order $n = 2, \dots, 6$.

Figure 11.9 shows on the left the solution of (11.1) with $f \equiv 1$ on a circular domain with 3 holes, representing, e.g., the displacement of a membrane with constant load. Numerical approximations were computed with the web-spaces S_{we} and, for comparison, also with hat-functions on a triangulation of the domain. For orders 2, 3, 4, 5, 6 (markers $*$, \bullet , \blacktriangle , \blacksquare , \star , respectively), the diagram on the right shows the L_2 -error versus the number of basis functions. Compared to hat-functions (\diamond marker), web-splines yield highly accurate solutions. For example, with 496 web-splines of order 4 the relative error in the L_2 -norm $\|u - u_h\|_0 / \|u\|_0$ is $9.1705e-05$ (in the H^1 -norm $7.8161e-04$) whereas 17602 standard finite elements are needed to achieve a relative L_2 -error of $5.1614e-04$.

The Galerkin matrix G_{we} for the space S_{we} can be computed with simple row and column operations from the larger matrix G_w for S_w . More precisely, $G_{we} = PG_wP^t$ with $p_{i,j} = (\delta_{i,j} + e_{i,j})/w(x_i)$, as is apparent from Definition 11.3.1. The transformation P , corresponding to the stabilization of the weighted B-spline basis, substantially reduces the condition number of the Galerkin system. In the example in Figure 11.9,

$$\text{cond}_\infty(G_w) = 2.3884e+12, \quad \text{cond}_\infty(G_{we}) = 226.24$$

for $h = 0.35$ and bilinear splines ($n = 2$). The size of the system is reduced from 496 to 351 variables. For smaller grid width and splines of higher order, the difference is even more dramatic. For example, for $h = 0.0875$ and $n = 5$, the condition estimate

of MATLAB [25] yields $\text{cond}_\infty(G_w) = 9.3315\text{e}+61$, while the condition of the reduced system is $3.0532\text{e}+10$. This shows that the parameter reduction by the transformation P significantly improves the stability of the Galerkin system, thus permitting more accurate solutions and better convergence rates of iterative solvers.

Of course, Galerkin's method applies to equations with variable coefficients as well. It is sufficient to assume that the boundary value problem can be described by the variational equations

$$a(u, v) = \langle f, v \rangle, \quad \forall v \in V = H_0^1(D), \quad (11.10)$$

where a and $\langle f, \cdot \rangle$ are continuous bilinear and linear forms on $H_0^1(D)$ and a is elliptic, i.e.

$$a(u, u) \succeq \|u\|_1^2$$

(cf. [12] for several examples). The Lax-Milgram Lemma implies existence and uniqueness of weak solutions in this very general setting. Again, the finite element approximation $u_h \in V_h$ is obtained by replacing V by V_h in (11.10).

Problems with prescribed boundary data, $u = g$ on ∂D , can be treated as well. They are reduced to the homogeneous case via the substitution $u =: v + \tilde{g}$, where \tilde{g} is an extension of g to all of D . In effect, we seek an approximation for u from the affine space $\tilde{g} + V_h$.

11.4.2. Natural boundary conditions

As a typical problem, we consider the Laplace equation with Neumann boundary conditions,

$$\Delta u = 0 \text{ in } D, \quad \frac{\partial u}{\partial n} = g \text{ on } \partial D, \quad (11.11)$$

assuming that the compatibility condition $\int_{\partial D} g = 0$ is satisfied. The weak formulation of (11.11) is

$$\int_D \text{grad } u \text{ grad } v = \int_{\partial D} g v, \quad \forall v \in H^1(D).$$

Since no boundary conditions are imposed on the test functions v , the finite element formulation is much simpler than for problems with essential boundary conditions. No weight function is required, and we can work with the spline spaces S or S_e .

As an example, Figure 11.10 illustrates the computation of incompressible flow through a channel with three obstacles. In this case, u is the potential and $-\text{grad } u$ the velocity. On the right of the Figure, we visualize $\|\text{grad } u\|$ for constant flow rate at the in- and outlet of the channel ($\frac{\partial u}{\partial n} = \pm v_0$ on the vertical boundaries). The finite element solution u_h was calculated with extended B-splines of order 4 ($u_h \in S_e$). As for essential boundary conditions, the results are much superior to mesh-based approximations. For 9709 basis functions the relative L_2 -error is $2.2499\text{e}-08$, which is by a factor 35290 smaller than for an approximation with 9835 hat-functions and a triangulation with mesh width $1/16$. Moreover, as shown on the left of Figure 11.10, to construct the basis for S_e , only very few B-splines have to be extended. For most of the inner shifts (marked with blue dots) the web-splines B_i coincide with the standard B-splines b_i .

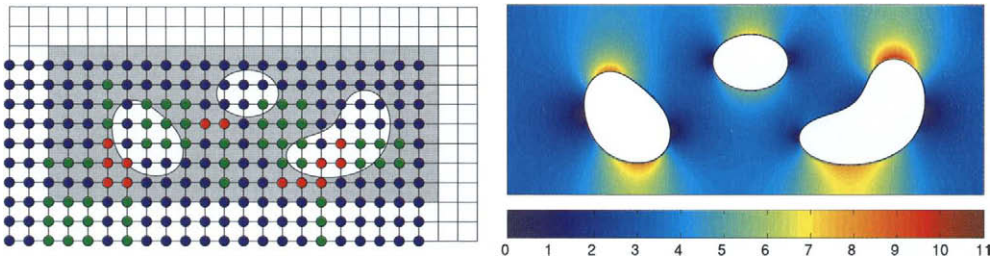


Figure 11.10. Incompressible flow, modeled with extended B-splines.

The more general Neumann problem,

$$-\operatorname{div}(a \operatorname{grad} u) + bu = f \text{ in } D, \quad \frac{\partial u}{\partial n} = g \text{ on } \partial D,$$

with $a(x), b(x) > 0$, can be approximated in a completely analogous fashion. The variational form is

$$\frac{1}{2} \int_D a \operatorname{grad} u \operatorname{grad} u + bu^2 - \int_D fu - \int_{\partial D} gu \rightarrow \min,$$

and we can use any of the spaces $S(D)$, $S_e(D)$, or $S(\mathbb{D})$ to compute finite element approximations.

11.4.3. Mixed and higher order boundary conditions

As an example of a mixed problem, we consider the system of linear elasticity

$$-\mu \Delta u - (\lambda + \mu) \operatorname{grad} \operatorname{div} u = f \text{ in } D$$

with the boundary conditions

$$u = \mathbf{0} \text{ on } \Gamma \subset \partial D, \quad \sigma(u)\xi = g \text{ on } \partial D \setminus \Gamma,$$

where Γ has positive $(m - 1)$ -dimensional measure. This system determines the displacement $(u_1(x), u_2(x), u_3(x))$ of a three-dimensional elastic body, fixed at Γ , under a volume force with density f and a force applied to $\partial D \setminus \Gamma$ with density g . The constants λ and μ are the Lamé coefficients, the matrix σ is the stress tensor, and ξ denotes the outward boundary normal. Without going into details, minimizing the potential energy over

$$\{u \in H^1(D)^3 : u(x) = \mathbf{0}, x \in \Gamma\}$$

yields a natural variational formulation. Hence, if w_Γ is a weight function equivalent to the distance to Γ , each of the spaces

$$S_w(D)^3, S_{we}(D)^3, S_w(\mathbb{D})^3$$

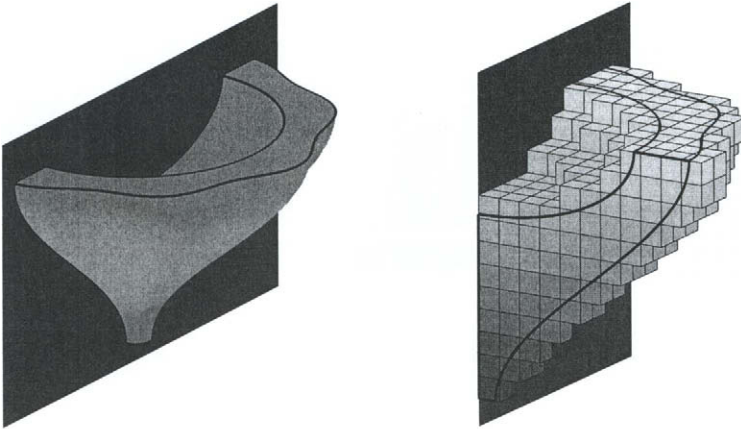


Figure 11.11. B-spline grid for computing the deformation of an elastic structure, fixed at a planar surface.

can be used to construct finite element approximations. An example is shown in Figure 11.11, where an elastic structure is attached to a planar surface $\Gamma = \{x : x_1 = 0\}$. Here, the obvious choice $w_\Gamma(x) = x_1$ yields very simple weighted finite element subspaces.

Higher order boundary conditions arise, e.g., in the biharmonic problem,

$$\Delta^2 u = f \text{ in } D, \quad u = g_0, \quad \frac{\partial u}{\partial n} = g_1 \text{ on } \partial D. \tag{11.12}$$

For homogeneous boundary conditions ($g_0 = g_1 = 0$), (11.12) describes the equilibrium position of a clamped plate under the action of a transverse force. In this case, finite elements should vanish to second order at the boundary. Hence, we choose a weight function, which, for smooth boundaries, is equivalent to the square of the distance function. For example, if D is the unit disc,

$$w = w_1^2, \quad w_1(x) = \frac{1}{2}(1 - x_1^2 - x_2^2).$$

We can also use the weight function to construct affine finite element subspaces, which satisfy the inhomogeneous boundary conditions. If $w = w_1^2$ and $\partial w_1 / \partial n = -1$ on the boundary, as in the above example, then

$$v = \tilde{g}_0 - w_1(\tilde{g}_1 + \text{grad } w_1 \text{ grad } \tilde{g}_0),$$

where $\tilde{\psi}$ denotes an extension of ψ to D , satisfies the boundary conditions. Hence, each of the affine spaces

$$v + S_w(D), \quad v + S_{w_e}(D), \quad v + S_w(\mathbb{D})$$

provides admissible approximations.

11.4.4. Error estimates

For most elliptic boundary value problems, the order of convergence of finite element approximations is determined by the order of the basis functions. Essentially, this is also true for all spline spaces described in Section 11.3. For the norm associated with the variational formulation, the arguments are particularly simple and familiar from classical finite element analysis [48,12]. As an illustration, we consider the Dirichlet problem for Poisson's equation, discussed in Subsection 11.4.1. The generalization to the abstract variational problem (11.10) is straightforward.

Theorem 11.4.1 *If u_h is a finite element approximation to a solution u of (11.1) from a linear subspace $V_h \subset H_0^1(D)$, then*

$$\|u - u_h\|_1 \preceq \inf_{v_h \in V_h} \|u - v_h\|_1.$$

This estimate is a direct consequence of the variational formulation. By (11.9), $u_h = \sum_i u_i B_i$ satisfies

$$\int \text{grad } u_h \text{ grad } v_h = \int f v_h, \quad \forall v_h \in V_h.$$

Combining this with (11.3) yields $\int \text{grad } (u - u_h) \text{ grad } v = 0$ for all $v \in V_h$. Hence, choosing $v := u_h - v_h$,

$$\int \text{grad } (u - u_h) \text{ grad } (u - u_h) = \int \text{grad } (u - u_h) \text{ grad } (u - v_h),$$

where the left-hand side is $\succeq \|u - u_h\|_1^2$ by Friedrich's inequality, and the right-hand side is $\preceq \|u - u_h\|_1 \|u - v_h\|_1$. Dividing by $\|u - u_h\|_1$, the Theorem follows.

Theorem 11.4.1 implies convergence of the finite element approximations from S_{we} and therefore also from the larger space S_w . Moreover, with the aid of the canonical projector

$$P_h u := \sum_i \left(\int \Lambda_i u \right) B_i, \tag{11.13}$$

associated with the dual functions for the web-basis, we can derive the following standard error bound [26,27].

Theorem 11.4.2 *If the weight function w satisfies (11.7) and the solution u as well as u/w are smooth, then*

$$\|u - u_h\|_1 = O(h^{n-1})$$

for finite element approximations u_h from S_{we} .

As shown in Figure 11.12 the numerical results for the model problem (11.1) live up to the theoretical predictions. For the example considered in Figure 11.9 the H^1 -error is plotted versus the number of web-splines on the left-hand side. The right-hand side shows the ratio between consecutive logarithmic errors in order to make the rate of convergence apparent. As is to be expected, the use of higher order approximations pays off.

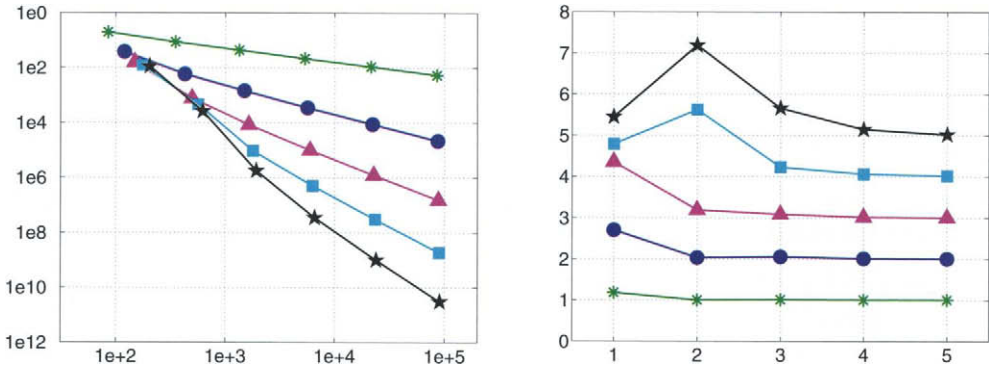


Figure 11.12. H^1 -error of web-approximations of order ≤ 6 and rate of convergence.

11.4.5. Implementation

A typical finite element simulation with splines consists of the following major steps:

- construction of the basis;
- assembly of the finite element system;
- iterative solution.

Each of these components will now be described in turn. For simplicity, we consider only the spaces $S(D)$, $S_w(D)$, and $S_{we}(D)$, which do not involve adaptive refinement.

Construction of the basis. The domain is enclosed in a bounding box and covered by a regular grid. Then the grid cells are classified into interior, boundary, and exterior cells (cf. Figure 11.2). This minimal information is sufficient to determine all parameters for constructing the various bases. Among all B-splines b_ℓ with support overlapping the bounding box, we mark the inner B-splines b_i and the outer B-splines b_j , checking whether their support contains at least one interior grid cell Q_i or merely boundary cells. The union of these B-splines, b_k , $k \in K = I \cup J$, forms a basis for S .

To construct the web-basis B_i for S_e , we first determine for each $j \in J$ the closest array $I(j)$ among the inner shifts i . Then we associate with each i the set $J(i)$ and the coefficients $e_{i,j}$. Since $e_{i,j}$, $i \in I(j)$, only depend on the relative position of the array $I(j)$ with respect to the outer shift j , and $I(j)$ is usually close to j , tabulated values can be used in most cases.

For the spaces S_w and S_{we} , we have to specify the weight function. If permitted by the description of D , the R -functions method can be used, and the weight function w is specified explicitly via an algorithmic expression. In general, w has to be constructed with the aid of distance functions and must be evaluated numerically.

Assembly of the finite element system. For the entries of the finite element matrices we have to compute integrals over grid cells with integrands that involve values and derivatives of the weight function, B-splines, and functions occurring in the formulation of the

boundary value problem. Good results are usually obtained with tensor product Gauß formulas. For interior grid cells, the integration is straightforward. However, boundary grid cells have to be divided into subcells, which can be mapped smoothly to standard domains. Integrating directly would result in a loss of accuracy due to the non-smooth boundary of the intersections $Q \cap D$. While the subdivision in two dimensions just requires cuts by straight lines (cf., e.g., [42]), in three dimensions a number of different possibilities have to be considered; Figure 11.13 shows a few typical examples. To facilitate the visualization of the subdivision, arrows indicate the transformation to standard integration areas.

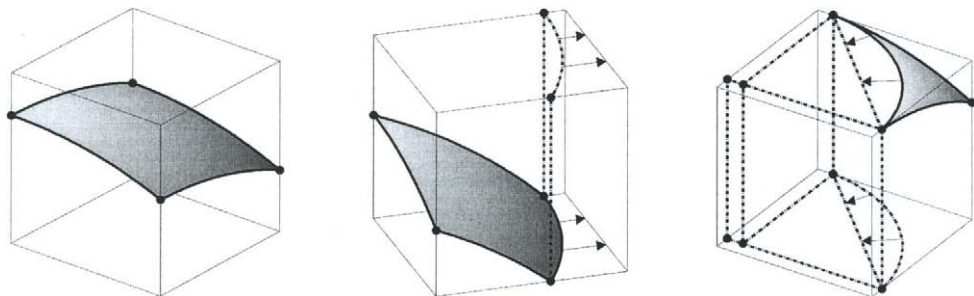


Figure 11.13. Subdivision of boundary cells.

The integrands can be evaluated efficiently. The most complicated part is computing derivatives of the weight function at the Gauß points. If w is defined via R -functions, automatic differentiation can be used in conjunction with the algorithmic definition [24, 47]. The numerical evaluation of w involves Newton's method. However, only very few iterations are necessary since for neighboring points good start values exist. Moreover, for weight functions with plateau, the additional computational effort is required only near the boundary.

Iterative Solution. The finite element systems can be solved with any of the standard iterative schemes. For example, as is illustrated in Figure 11.14, conjugate gradients, preconditioned with SSOR, give very good results. For the Dirichlet problem, considered in Figure 11.9, we have plotted on the left-hand side the number of iterations versus the dimension of the spline space S_{we} . The diagram on the right-hand side shows the total computing time in seconds, measured on a Pentium II/400 MHz class personal computer. There is only a fairly moderate growth in the required computational work. Systems with more than 600,000 unknowns could be handled without any stability problems.

The spline approximations on uniform grids are also ideally suited for multigrid techniques. Subdivision algorithms [8,15] and the standard projector (11.13) provide natural grid transfer operators. For the standard multigrid w-cycle optimal grid independent convergence rates can be established [28]. Hence, the solution time remains proportional to the dimension of the spline spaces. Such techniques are particularly important for large

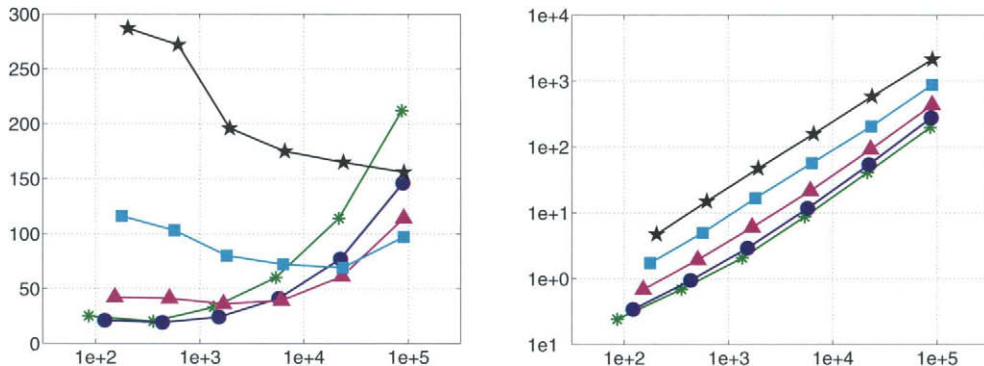


Figure 11.14. Convergence of cg-iterations and total computing time in seconds.

scale three-dimensional applications.

11.5. SUMMARY

Weighted spline approximations provide a natural link from geometric modeling to finite element simulation. They do not require any grid generation, thereby eliminating an often time-consuming preprocessing step. Moreover, software for manipulating B-splines can be used to assemble Galerkin matrices and to visualize numerical results. Unlike for mesh-based methods, there are no limitations on the smoothness of the basis. Therefore, highly accurate approximations are possible with relatively few parameters.

An important feature of weighted finite elements is the simplicity of the geometry description. For domains defined in terms of implicit equations, the associated weight functions can be constructed with the R-functions method. This technique is applicable to many engineering problems, in particular those involving constructive solid geometry models. Efficient algorithms are also available for general NURBS-domains, where weight functions have to be constructed numerically. Hence, the methods combine well with standard boundary representations in CAD-systems.

Acknowledgement. I am very grateful to Joachim Wipper for extensive numerical computations. Moreover, I would like to thank Alexander Fuchs and Jörg Hörner for helping with comparisons of spline- and mesh-based methods.

REFERENCES

1. R.A. Adams. *Sobolev Spaces*. Academic Press, 1978.
2. J.H. Argyris. *Energy Theorems and Structural Analysis*. Butterworths, London, 1960.
3. J.H. Argyris. *Recent Advances in Matrix Methods of Structural Analysis*. Pergamon Press, Elmsford, N.Y., 1964.
4. I. Babuška. The finite element method with penalty. *Mathematics of Computation*, 27(122):221–228, 1973.

5. I. Babuška. The finite element method with Lagrangian multipliers. *Numer. Math.*, 20:179–192, 1973.
6. T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. Meshless methods: An overview and recent developments. *Comput. Methods Appl. Mech. Eng.*, 139(1-4):3–37, 1996.
7. P. Bochev and M. Gunzburger. Finite element methods of least squares type. *SIAM Review*, 40:789–837, 1998.
8. W. Böhm. Inserting new knots into B-spline curves. *Computer-Aided Design*, 12:199–201, 1980.
9. C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, 1978.
10. C. de Boor, K. Höllig, and S. Riemenschneider. *Box Splines*. Springer-Verlag, 1993.
11. I.G. Bubnow. Rezension über die mit dem Schurawski-Preis ausgezeichneten Abhandlungen von Prof. Timoschenko, Sammlung des Instituts für Verkehrswesen, Heft 81, SPB, 1913.
12. P.G. Ciarlet. *The Finite Element Method for Elliptic Problems*. North-Holland, Amsterdam, 1978.
13. P.G. Ciarlet and J.P. Lions, editors. *Handbook of Numerical Analysis*. North Holland, Amsterdam, 1991.
14. R.W. Clough. The finite element method in plane stress analysis. *Proc. 2nd ASCE Conf. on Electronic Computation*, Pittsburgh, 1960.
15. E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics Image Proc.*, 14:87–111, 1980.
16. R. Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bull. Amer. Math. Soc.*, 49:1–23, 1943.
17. G.E. Farin, editor. *Geometric Modeling: Algorithms and New Trends*. SIAM, 1987.
18. G.E. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1988.
19. G.E. Farin, editor. *NURBS for Curve and Surface Design*. SIAM, 1991.
20. A. Fuchs. Almost regular Delaunay-triangulations. *Internat. J. Numer. Meths. Eng.*, 40:4595–4610, 1997.
21. A. Fuchs. *Optimierte Delaunay-Triangulierungen zur Vernetzung getrimmter NURBS-Körper*, PhD thesis, Universität Stuttgart, 1999.
22. A. Fuchs. Almost regular triangulations of trimmed NURBS-solids. *Engineering with Computers*, 17:55–65, 2001.
23. B.G. Galerkin. Stäbe und Platten; Reihen in gewissen Gleichgewichtsproblemen elastischer Stäbe und Platten, *Vestnik der Ingenieure*, 19:897–908, 1915.
24. A. Griewank. Evaluating derivatives: principles and techniques of algorithmic differentiation. *Frontiers in Applied Mathematics 19*, SIAM, 1999.
25. N.J. Higham and F. Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM J. Matrix Anal. Appl.*, 20(4):1185–1201, 2000.
26. K. Höllig, U. Reif, and J. Wipper. Weighted extended B-spline approximation of Dirichlet problems. *SIAM J. Num. Anal.*, 39:442–462, 2001.
27. K. Höllig, U. Reif, and J. Wipper. Error estimates for the web-method. In T. Lyche

- and L.Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, Oslo 2000, pages 195–209. Vanderbilt University Press, Nashville, 2000.
28. K. Höllig, U. Reif, and J. Wipper. Multigrid methods with web-splines. *Numer. Math.*, 91(2):237-256, 2002.
 29. K. Ho-Le. Finite element mesh generation methods: a review and classification. *Computer-Aided Design*, 20(1):27–38, 1988.
 30. J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. A.K. Peters, 1993.
 31. K.H. Huebner, E.A. Thornton, and T.G. Byrom. *The Finite Element Method for Engineers*. John Wiley & Sons, 1994.
 32. L.W. Kantorowitsch and W.I. Krylow. *Näherungsmethoden der Höheren Analysis*. VEB Deutscher Verlag der Wissenschaften, Berlin, 1956.
 33. H. Kardestuncer, editor. *Finite Element Handbook*. McGraw-Hill, New York, 1987.
 34. R. Kraft. *Adaptive und linear unabhängige multilevel B-Splines und ihre Anwendungen*. PhD thesis, Universität Stuttgart, 1998.
 35. J. Mackerle. *Finite Element Methods: A Guide to Information Sources*. Elsevier Science Publishers B.V., 1991.
 36. S.J. Owen. A survey of unstructured mesh generation technology. *Proceedings of 7th International Meshing Roundtable*, pages 239–268. Sandia National Laboratories, 1998.
 37. L.B. Rall. *Automatic Differentiation*. Springer Lecture Notes in Computer Science 120, 1981.
 38. J.W.S. Rayleigh. *The Theory of Sound*, 2nd edition., London. 1894 and 1896.
 39. W. Ritz. Über eine neue Methode zur Lösung gewisser Variationsprobleme in der Mathematischen Physik. *J. reine angew. Math.*, 135:1–61, 1908.
 40. V.L. Rvachev and T.I. Sheiko. *R-functions in boundary value problems in mechanics*. *Appl. Mech. Rev.*, 48(4):151–188, 1995.
 41. V.L. Rvachev, T.I. Sheiko, V. Shapiro, and I. Tsukanov. On completeness of RFM solution structures. *Comp. Mech.*, 25:305–316, 2000.
 42. V.L. Rvachev, A.N. Shevchenko, and V.V. Veretel'nik. Numerical integration software for projection and projection-grid methods. *Cybernetics and Systems Analysis*, 30:154–158, 1994.
 43. I.J. Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. *Quart. Appl. Math.*, 4:45–99 and 121–141, 1946.
 44. L.L. Schumaker. *Spline Functions: Basic Theory*. Wiley-Interscience, 1980.
 45. V. Shapiro. Theory of *R-functions* and applications: A primer. Technical Report CPA88-3, Cornell Programmable Automation, Sibley School of Mechanical Engineering, Ithaca, NY., 1988.
 46. V. Shapiro and I. Tsukanov. Meshfree simulation of deforming domains. *Computer-Aided Design*, 31:459–471, 1999.
 47. A.N. Shevchenko and V.N. Rokityanskaya. Automatic differentiation of functions of many variables. *Cybernetics and Systems Analysis*, 32(5):139–158, 1996.
 48. G. Strang and G.J. Fix. *An Analysis of the Finite Element Method*. Prentice-Hall, 1973.
 49. M.J. Turner, R.W. Clough, H.C. Martin, and L.C. Topp. Stiffness and deflection

- analysis of complex structures. *J. Aeronaut. Sci.*, 23(9):805–823, 854, 1956.
50. O.C. Zienkiewicz and R.L. Taylor. *Finite Element Method – Basic Formulation and Linear Problems, Vol. I, II, III*. Butterworth & Heinemann, 2000.

Chapter 12

Subdivision Surfaces

Malcolm Sabin

This chapter deals with the technology of subdivision surfaces, how they can be defined and how schemes are analysed for desirable properties, thus allowing tuning of the details of the scheme for optimum behaviour.

12.1. SUBDIVISION SURFACE DEFINITIONS

The basic idea is that an initial 2-manifold network of vertices, edges and facets (often now referred to as the *control polyhedron*, even though the facets need not be planar, or sometimes as the *mesh*) can be refined by computing new vertices and joining them up to form a new polyhedron. This operation can be repeated indefinitely, and as the process continues, if the refinement construction is suitable, the polyhedron converges towards a limit surface. The local nature of the process means that the polyhedron need not be limited to rectangularity, and surfaces of any topological genus can be defined.

Alternative approaches include the definition of n -sided patches, possibly by use of toric variety theory, and the assembly of polynomial patches meeting n at a vertex. The latter is described in Chapter 8 of this handbook.

However, the development of the subdivision approach into a practical tool, capable of succeeding NURBS as the surface description method of choice throughout animation and engineering design is receiving enormous academic attention. This chapter cannot hope to describe all that is going on. The reader is also directed to the course notes [39] which support the tutorials given on this subject at the SIGGRAPH conferences in recent years.

12.2. INTRODUCTION – SUBDIVISION CURVES

12.2.1. The Chaikin construction

Although significant analysis had been carried out earlier by de Rham[9], the first subdivision construction noticed by the CAGD community was that proposed by Chaikin[6] as

an ad hoc way of constructing a smooth curve, given a sequence of points.

Suppose that the sequence of points is

$\dots \mathbf{a}_{i-1}, \mathbf{a}_i, \mathbf{a}_{i+1}, \mathbf{a}_{i+2} \dots$

Each span from this sequence then has two new points constructed on it, at $(3\mathbf{a}_i + \mathbf{a}_{i+1})/4$ and at $(\mathbf{a}_i + 3\mathbf{a}_{i+1})/4$, and these new points are reconnected in the obvious way to form a new sequence. If we regard the sequence as defining a polygon, the new polygon is essentially the old one with the corners cut off, and if this is done often enough the polygon becomes a smooth curve.

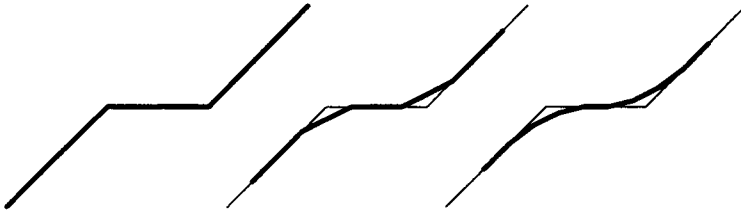


Figure 12.1. Chaikin subdivision

This material was presented at the first major conference on CAGD[3], but unfortunately did not appear in the conference proceedings. However, at the conference both Riesenfeld[28] and Forrest[16] were stimulated to investigate the mathematical nature of these curves, and discovered that the limit curve was an equal interval quadratic B-spline.

12.2.2. Higher degree splines

This observation immediately led to the generalisation that every equal interval B-spline has its own subdivision construction. Adding knots to create a new denser knot vector from the old one can be done on a binary basis (one new knot in each old knot interval), or ternary or higher.

Each span of an equal interval n^{th} degree B-spline has a constant n^{th} derivative, numerically equal to the n^{th} divided difference of the control polygon. Suppose that each old span is divided by knot insertion into p equal pieces. After the knot insertion those new pieces will all have the same n^{th} derivative as the original span and thus the same divided differences. We can deduce from this that their n^{th} differences will be $1/p^n$ of those of the original span.

We can recover the new polygon from these differences by a process of summation, the exact inverse of differencing. In particular, if we apply the processes of differencing n times, repeating each of the differences p times and dividing by p^n , and then anti-differencing n times, to cardinal data, we generate the coefficients appearing in the templates.

We take as example here the equal interval cubic B-spline with a binary subdivision.

The generalisation of the de Casteljau[8] construction gives another way of considering high degree schemes. The knot insertion process can be thought of as a sequence of stages, each stage creating a polygon with just one extra vertex, where every new or repositioned vertex is a weighted mean of just two precursor vertices. This has the important advantage that high-degree schemes may be thought of in terms, not just of one big template, but of a set of smoothing steps involving just a few terms at a time.

12.2.3. The 4-point interpolatory scheme

Dyn, Levin and Gregory[14] produced a new idea, that new points could be added into the existing sequence. Because the original points remain as members of all subsequent sequences, they become members of the limit curve, which therefore interpolates them.

The coefficients for the template for the new midpoint in each span in their 4-point scheme become $[-1, 9, 9, -1]/16$.

Whereas the schemes derived from the B-splines have piecewise polynomial limit curves, with well defined continuity levels between adjacent pieces, it can easily be shown that this 4-point scheme, and indeed all C1 interpolating schemes with fixed support cannot have polynomial pieces. Consider the curve resulting from a polygon which has abscissae, $x_i = i$ at the integers and ordinates y_i zero except at $x = 0$ (*cardinal data*).

This curve must have regions of negative y , since the limit curve must cross the x -axis at $x_i \neq 0$. However, the curve is also self similar at different scales, because it can be expressed as the sum of copies of itself scaled in x by the arity of the scheme and in y by the coefficients of the refinement. Each such copy has images of the negative regions, and the iterated refinement process makes it clear that these must cluster towards the ends of the support. In the limit there are an infinite number of such changes of sign in the finite support of the limit function, and no piecewise polynomial has an infinite number of roots.

Further, the continuity level just fails to reach C2 because the 2nd derivative is just undefined. This can be seen by inventing a composite operator, which applies the 4-point scheme and then rescales the polygon by a factor of 2 in x and 4 in y . This composite operator gradually increases the second difference without bound as we keep applying it. This behaviour is now understood in terms of Jordan blocks in the eigenfactorisation of the subdivision operator, as will be explained below.

The points constructed in this construction lie on the Lagrange polynomial through four successive points of the polygon. Deslauriers and Dubuc[15] approached subdivision from a fractal background, and carried out a systematic analysis of such schemes.

12.3. BOX-SPLINES

The simplest generalisation of B-splines to surfaces is via tensor product constructions, where bivariate basis functions are formed from products of univariate functions. It is straightforward to see, from the commuting of subdivision in the two directions, that we can describe such surfaces in subdivision terms by just taking the tensor product of the refinement operators.

A rather broader generality is by considering the box-splines, described in Chapter 10. Because a box (the outer product of a set of intervals) can always be subdivided into smaller boxes by subdividing those intervals, we can always express a box-spline basis

function over a coarse grid in terms of the sum of basis functions over a nested finer grid.

12.4. GENERALIZATIONS TO ARBITRARY TOPOLOGY

Once such a bivariate scheme has been set up, its derivation can be forgotten. It becomes a way of making a new, refined polyhedron from a coarser one, and this leads to a generalisation of profound importance. We can leave the rigid rectangularity of tensor product systems behind, and create schemes which define smooth surfaces from polyhedra with arbitrary connectivity. The first two of these were the Doo-Sabin[11][12] and Catmull-Clark[4] schemes, but many more have been derived since then.

12.5. SOME SPECIFIC SCHEMES

12.5.1. The Doo-Sabin quadratic scheme

The biquadratic B-spline gives a scheme in which a new vertex is created for each old corner of each old facet, as a linear combination of the old corners of that facet. New facets are created linking these new vertices 'inside' the old facets, across the old edges and across the old vertices. The four coefficients of the old vertices in a new one are just the tensor products of the $[1, 3], [3, 1]/4$ quadratic templates from table 12.1, $[1, 3, 9, 3]/16$ taken in cyclic order around the old facet.

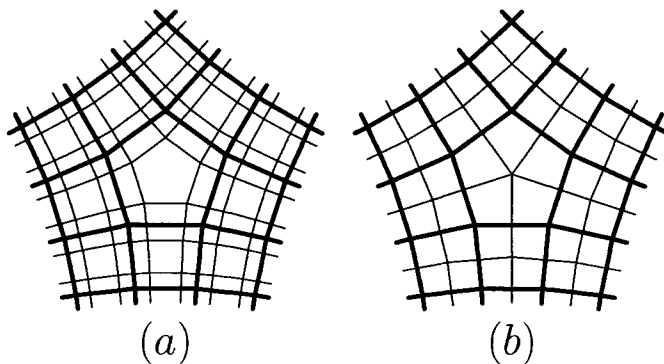


Figure 12.2. Topologies of the (a) Doo-Sabin and (b) Catmull-Clark schemes.

To generalise this we need only define how new vertices are constructed in old facets of other than four sides. In [12] the coefficients

$$a_{ij} = \begin{cases} \frac{(n+5)}{4n}, & i = j \\ \frac{3+2\cos(2\pi(i-j)/n)}{4n} & i \neq j \end{cases}$$

are supplied. These lead to a surface which is C1 everywhere and which has well behaved curvatures. Being only a quadratic scheme it cannot give C2, but all curvatures are bounded to values sensible in the context of the polyhedron itself.

12.5.2. The Catmull-Clark cubic scheme

The bicubic B-spline gives a scheme in which a new vertex is created at the centroid of each old facet (the tensor product of the two $[4, 4]/8$ templates), another is created close to the midpoint of each old edge, a particular weighted mean of the vertices of the two adjacent facets, and another is created close to each old vertex, a particular weighted mean of the vertices of the adjacent old facets.

If we look at this as a sequence of smoothings, as suggested at the end of section 12.2.2 above, this can be re-expressed as taking the mean of the vertices of an old facet to give the centroids of the old facets; then we imagine joining the old vertices to the new ones to give a square mesh rotated 45 degrees from the original; then we repeat the taking of the means to give the new ‘edge-vertices’; and finally the old vertices are smoothed to new positions which are linear combinations of the adjacent new ones. The effect on facets is that each old (four-sided) facet is divided into four new ones.

The Catmull-Clark surface[4] generalises this by carrying out the same three steps. The first two are just means, which are well-defined however many sides the facet has, and the third merely requires the choice of appropriate weights to use when there are other than four edges meeting at a vertex. The net result is that each old n -cornered face is divided into n new four-sided faces.

In [4] two possible weighting schemes for giving new ‘vertex-vertices’ are given. Both give C1 surfaces. Neither gives C2. Indeed, in both cases curvatures near old n -valent vertices become unbounded rather sharply as refinement proceeds when $n \neq 4$. This problem was addressed by Ball and Storry [35] who found vertex weights for the final stage ameliorating the problem, and by Sabin [31] who defined weights depending on valency for all three stages which lead to bounded curvature, though not C2 continuity, for all values of valency.

12.5.3. The Loop triangular mesh scheme

Loop[20] applied the box-spline subdivision approach to the 3-direction quartic C2 box-spline. It is defined over any polyhedron with triangular faces. It is at first sight rather similar to Catmull-Clark, without its first stage. A new vertex is computed near the midpoint of each old edge, a linear combination (derived from the box-spline) of the old vertices at the ends of the edge and the old vertices across the adjacent triangles. New vertices are also computed near the old vertices, being a linear combination of the old vertex itself and the mean of the new vertices around it. The new facets divide each old facet into four pieces.

12.5.4. The Butterfly interpolatory scheme

Another scheme based on a triangular mesh is the Butterfly, an interpolating scheme from Dyn and Levin[13].

It has the same topology as Loop’s scheme, with each triangular facet being divided into four subfacets, but now the original vertices are retained. The new ones, near the midpoints of the original edges, are defined as a linear combination of the ends of the edge, the two vertices across the two adjacent triangles and the four vertices across the triangles adjacent to them.

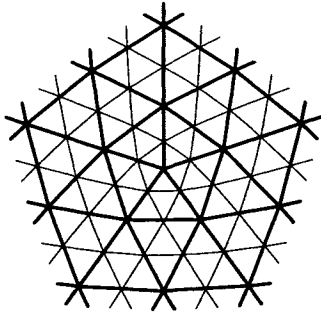


Figure 12.3. Topology of the Loop scheme.

In figure 12.4 the new vertex \mathbf{x} near the midpoint of \mathbf{de} is given by

$$\mathbf{x} = \frac{\mathbf{d} + \mathbf{e}}{2} + \frac{\mathbf{b} + \mathbf{g}}{8} - \frac{\mathbf{a} + \mathbf{c} + \mathbf{f} + \mathbf{h}}{16}$$

12.6. ANALYSIS OF CONTINUITY AT THE SINGULARITIES

The main mathematical challenge in understanding the nature of these generalised surfaces has been seen as determining the behaviour around certain singular points.

This analysis goes as follows:-

12.6.1. Support

Within any scheme, there are a finite number of points at level $i + 1$ which are altered in position by moving a single vertex \mathbf{v} at level i . Let the furthest of these from \mathbf{v} be distance d away. Then the largest step at the next level will be kd with $k < 1$, and the furthest influence any vertex can have is $d \sum_{j=0}^{\infty} k^j$ which is equal to $d/(1 - k)$. For any particular scheme we can apply sharper arguments in which k and d depend on direction, to determine a boundary outside which a particular vertex has no effect.

12.6.2. Regular regions

First we observe that every scheme has a natural regular topology, which is preserved under subdivision. Typically after one step, or at most two, either every vertex has the same valency or every face has the same number of sides. Further, the number of entities with other than the standard number does not alter as a result of subdivision. For example, in Catmull-Clark, after one iteration every face is four-sided, and the number of non-4-valent vertices is invariant. Each n -valent vertex is derived either from another at the previous step, or from an n -sided face.

As subdivision proceeds, the surface between such points gets divided into finer facets, all meeting regularly, and after relatively few steps, the singularities become isolated in the sense that moving one of them (in the refined mesh) will not influence the shape of the surface near another.

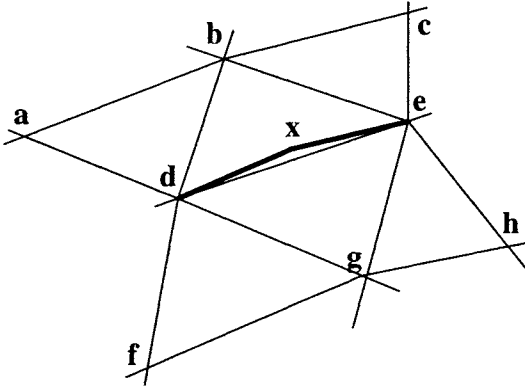


Figure 12.4. Butterfly subdivision

For the spline-based systems, if we look at the regular regions, we realise that we know the exact form of the limit surface, because it consists of pieces each influenced by a finite number of control points, and if those control points are all regularly connected, the limit surface of the subdivision must itself be the spline surface. Its level of continuity is thereby well-defined.

For systems which are not spline-based we cannot make this deduction straight away, but in fact there is an analogous statement that we can make after applying the irregular-point analysis to our regular points.

12.6.3. Neighbourhoods of singularities

Once the singularities are isolated, we can look at the immediate vicinity of the singularity as having at least topological rotational symmetry. The configuration is regular, with a single irregularity at the centre. After one step, the configuration is exactly similar, and we can label the vertices with a common numbering system before and after.

If \mathbf{C}_0 is the column vector of those points in (or on the edge of) whose domain of influence the singularity lies, and \mathbf{C}_1 is the column of corresponding points after one subdivision step, we can express all the linear combinations involved in deriving the positions of \mathbf{C}_1 as a matrix equation

$$\mathbf{C}_1 = \mathbf{M} \cdot \mathbf{C}_0$$

The main tool to explore the behaviour in the neighbourhood of the singularity is eigenanalysis, which is relevant because, if we can factorise the subdivision matrix \mathbf{M} into pre-eigenvectors \mathbf{R} , a diagonal eigenvalue matrix \mathbf{L} and post-eigenvectors \mathbf{S} so that $\mathbf{M} = \mathbf{R}\mathbf{L}\mathbf{S}$, the effect of a very large number of refinements will be $\mathbf{M}^n = \mathbf{R}\mathbf{L}^n\mathbf{S}$, which will clearly be dominated by the largest eigenvalues, with their eigenvectors telling us what happens at a very small scale.

There are two numbering sequences which give rise to particular structures for \mathbf{M} which give qualitative insight into the behaviour. The first is a sequence which starts at the

singularity itself and then works out in rings. This gives \mathbf{M} a block lower triangular structure with only a finite number of blocks on the diagonal.

The only non-zero eigenvalues of \mathbf{M} are then the eigenvalues of the diagonal blocks, and as soon as we raise \mathbf{M} to any power greater than 1, we see that all terms to the right of the rightmost diagonal non-zero become zero. The right eigenvectors therefore have a limited number of terms, and we can carry out most of the eigenanalysis with only that number of elements in each of the vectors. This is another way of determining the support. We can remove from consideration any original vertices which have zero columns in \mathbf{M}^2 . For example,

$$\mathbf{M}^2 = \begin{bmatrix} A & 0 & 0 & 0 \\ B & C & 0 & 0 \\ D & E & 0 & 0 \\ F & G & H & 0 \\ \dots & & & \dots \end{bmatrix}^2 = \begin{bmatrix} A^2 & 0 & 0 & 0 \\ BA + CB & C^2 & 0 & 0 \\ DA + EB & EC & 0 & 0 \\ FA + GB + HD & GC + HE & 0 & 0 \\ \dots & & & \dots \end{bmatrix}$$

The other numbering system to use is one which works around the centre, working out in each sector. \mathbf{M} then has a block circulant structure (see Davis[7] for tutorial material), with each block itself being block lower triangular, with a finite number of sub-blocks on the diagonal. If there is a vertex, rather than a facet at the singularity, there needs to be a technical fix of replacing that vertex by a set of identical vertices there, one per sector. This does not affect the logic of the process.

The blocks structure itself does not depend on the valency of the singularity. This enables us to treat the eigenanalysis of the subdivision matrix in a way which leaves the valency as an algebraic variable. Because of the strong connection between circulant matrices and finite Fourier analysis, it also enables us to split the system into the different rotational 'frequencies', which are present in the system, solving each independently, enabling us to carry out algebraic eigenanalysis for high valencies.

If we imagine that the process of subdivision is actually carried out in three stages, the first stage is to multiply the column \mathbf{C}_0 by the right eigenvectors. The appropriate scaling for the eigenvectors is that the sum of their elements should equal either 1 or 0. Each eigenvector summing to one multiplies the points in \mathbf{C}_0 as a weighted mean, giving a point: each eigenvector summing to zero multiplies the points in \mathbf{C}_0 as a zero-sum linear combination, giving a displacement vector invariant under translation of the entire configuration. These products of the right eigenvectors with \mathbf{C}_0 thus give an abstraction of \mathbf{C}_0 into a set of *triggers*.

The triggers are then multiplied by the eigenvalues themselves, those corresponding to larger eigenvalues being made relatively larger thereby, and the scaled triggers are then multiplied by the left eigenvectors, so that each trigger has a characteristic effect on the configuration \mathbf{C}_1 and on all subsequent configurations.

12.6.4. Limit point

It turns out that because every new vertex is given by a weighted mean of old vertices, the rows of \mathbf{M} all sum to unity, and this in turn means that there is a single unit eigenvalue. For the spline-based systems, all the weights are positive and less than unity, from which

we can deduce that all other eigenvalues have absolute value less than 1. In fact this also holds for other systems so far explored.

The corresponding left eigenvector consists of a column of 1s, meaning that the entire configuration converges to a single point, the limit position of the singularity itself. The right eigenvector is the only one which sums to 1, and it gives the coefficients of a linear combination which determines this position in terms of the original vertices.

This eigencomponent always comes from the zeroth frequency component.

12.6.5. Natural configuration and characteristic map

If we use as the origin exactly this point, then there is no contribution to this component because the trigger is zero, and the behaviour becomes dominated by the next eigenvalues. In well-behaved schemes we find two equal eigenvalues λ , with distinct eigenvectors, both arising from the unit frequency component. All other eigenvalues are strictly smaller in absolute value. In [34] Storry observed that if we take as our x and y coordinate axes the trigger vectors obtained by multiplying the actual configuration by the right eigenvectors, and multiply all coordinates after every step by the reciprocal of the eigenvalue, the configuration converges towards planar (the tangent plane), with all the control points converging to what he called the *Natural Configuration*.

Reif [27] went further and, using the fact that in the outer parts of the natural configuration we have regular regions which can be parametrised, he identified the *Characteristic Map* which determines how the parameters in the regular regions map to position in the tangent plane. He observed that it is possible to set up schemes in which the characteristic map is not 1:1, giving sharp creases in the limit surface. He and Peters [21] gave as an example a modified version of Doo-Sabin with extremely perturbed weights which actually showed this effect.

12.6.6. Curvatures

Using the natural configuration x - and y -axes, and a z -axis perpendicular to them, the z -coordinates of all the control points shrink as fast as the next eigenvalue with an eigenvector outside the eigenspace of those that determine the natural configuration.

In well-behaved schemes, after 1, λ and λ , there come three eigenvalues of equal value μ , one from the zero-frequency component and two from the 2-frequency component, and all remaining eigenvalues are strictly smaller.

By looking at estimators of curvature we see that if $\mu > \lambda^2$, those estimators are multiplied at every step by μ/λ^2 and therefore diverge, giving infinite curvatures in the limit. If $\mu < \lambda^2$ they converge to zero, giving 'flat-spots' in the limit. Only if $\mu = \lambda^2$ can the curvature at the singularity be well-behaved.

Even this, however, is not enough, because we can take the ideas of the natural configuration and characteristic map a stage further. The eigenvectors corresponding to λ give the x - and y -coordinates of limiting configurations. If we take the z -coordinates from one of the μ eigenvectors, we find one of three *second order natural configurations*, to which correspond *second order characteristic maps*, shapes which span the space of possible immediate neighbourhoods of the singularity. For well defined curvature over this neighbourhood, each of those three shapes must be a paraboloid. In fact the 0-frequency shape is an elliptic paraboloid $z = x^2 + y^2$, and the two 2-frequency shapes are hyperbolic

paraboloids, $z = x^2 - y^2$ and $z = 2xy$.

The regular parts of the natural configuration imply well-defined surface pieces, and for true C2 they must all lie exactly in a single quadric for each of the μ -components. If $\mu = \lambda^2$ and the eigenvectors do not satisfy this condition, the curvatures at the singularity are not precisely defined, although they can be bounded.

12.6.7. Tuning subdivisions for better behaviour

So far we do not know a straightforward subdivision scheme which satisfies all these conditions at singularities, although some methods have been contrived for overriding the local curvature behaviour.

One of these approaches, pioneered by Prautzsch in [23,24] is to take an initial scheme and eigenfactorise it. The eigenvectors are retained, but the eigenvalues are replaced by more desirable ones, thus giving a new subdivision. This has the problem that we no longer guarantee that all the weights are positive, and that the templates for subdivision are not necessarily quite so local as in the starting point. In fact Prautzsch and Umlauf are content in their modifications of the Catmull-Clark (in [25]) and Loop and Butterfly (in [26]) schemes, with achieving continuity by accepting flat spots. Clearly, in order to achieve well-defined curvatures it may be necessary also to modify the eigenvectors.

Another approach, usable when the regular regions have a known limit surface and good behaviour, is to assume that the weights are dependent on the valency n only. The eigenanalysis is carried through for generic valency with these weights w_i associated with the n -valent vertex as algebraic variables, thus turning the conditions $\mu = \lambda^2$ into equations in w_i and the valency itself, which can be solved for the w_i as functions of n . There is no guarantee that the weights will be positive, but this becomes fairly apparent quite quickly. It is also necessary to check that the characteristic map remains 1:1, and to test whether the second order characteristic map is actually quadratic. This method was used in deriving the scheme reported in [31], which fails the second test by a small amount. The locality of the templates is maintained by definition.

12.6.8. Jordan blocks

Some subdivision matrices do not have a neat eigenfactorisation. The diagonalisation process leads to an irreducible block diagonal structure. When this happens we get two equal eigenvalues with coincident eigenvectors. When this geometric degeneracy occurs among the μ eigencomponents, it leads to the local curvature estimators diverging logarithmically. This behaviour occurs in the 4-pt interpolating curve scheme of [14].

It is possible for an even milder form of this behaviour to occur when there are close eigenvalues with close eigenvectors. At the original scale of the configuration the curvatures look reasonable, but at this scale the apparent curvatures stem from two large almost equal terms. As refinement proceeds, one of these terms gradually decays, leaving the other contributing a very large, though bounded, curvature. This behaviour occurs in a C2 ternary variant of the scheme of [14]. This variant has $\mu = \lambda^2$ and all other eigenvalues strictly less than μ , but one of these others is almost as large as μ .

12.6.9. Discontinuities of curvature

In order to find a discontinuity of curvature at the singularity we need again to have multiple μ eigenvalues, but with distinct eigenvectors. The extra μ components will typically come from higher frequency parts of the circulant system. This behaviour occurs in the Doo-Sabin scheme.

12.6.10. Higher derivatives

When we do achieve well-defined C2 behaviour at singularities, the analysis of higher derivatives (which will presumably be even more challenging) will take a clearly visible path. We will either subtract the 2nd degree shape from the actual configuration, or limit ourselves to configurations which have zero curvature. The next eigenvalues will be four equal values ν , two from the 1-frequency component and two from the 3-frequency component, ν will need to equal λ^3 and the corresponding characteristic maps will need to lie exactly in $z =$ some cubic function of (x, y) . The pattern obviously extends to higher derivatives still, but will probably not be seen as important for some years.

12.6.11. Precision set

The precision set is the set of surfaces S for which the final surface lies in S if all the control points lie in S . We can deduce this if, for an initial configuration in S the next configuration also lies in S . For the spline-based systems the precision set is only the set of planes.

For interpolating schemes with finite support, the conditions for well-defined curvatures demands that the precision set should include paraboloids.

For spline-based systems there is an analogous demand, identified by Adi Levin[5], that if the initial control point x - and y -coordinates are spaced as in the characteristic map, and the z -coordinates lie in a quadratic function of x and y , then the same property should hold after one iteration, and the quadratic after should have the same quadratic terms as before.

12.7. FIRST STEP ARTIFACTS

A new issue, becoming visible as subdivision surfaces start to be used for commercial work in commercial systems, is that the original topology of the polyhedron shows through, in the form of ripples whose spatial frequency is that of the original vertices. By Shannon's sampling theorem[33] only components of spatial frequency up to half the vertex frequency are justified by the data, and so these ripples are definitely artifacts.

They are called first-step artifacts because it is the first subdivision step which makes them explicit.

Tools for analysing these are not yet well-developed, but we can see that spatial Fourier analysis will be one of the important ones. This already confirms that for most schemes the first step does 8 or 16 times as much damage as the second, and so fixes will probably take the form of modifying the coefficients for the first iteration or possibly the first two, in ways which depend on the actual geometry, and are designed to minimise the artifacts.

12.8. CURRENT RESEARCH DIRECTIONS

As well as the problem of analysis of artifacts, there are a few new directions in which active research is ongoing.

12.8.1. Square root of 3 scheme

This scheme was devised by Kobbelt in [19] and has also been looked at independently by Guskov[17].

The regular mesh is triangular and the refinement step places a new vertex at the centroid of each triangle. Each such vertex is joined to its neighbours across the original edges and also to new vertices near the original vertices, but smoothed by taking a linear combination of the original vertex and its new neighbours. The topology of refinement is shown in figure 12.5(a).

The rotation of the mesh in each step gives rise to complex eigenvalues, but if we compose the numbering schemes of two steps the rotation can be cancelled out and we can work in terms of real eigenvalues again. In the circulant view, complex eigenvalues correspond to phase shifts in the Fourier analysis. Twisting the numbering scheme one way in the first step and the other way in the second means that we are composing a scheme with its complex conjugate, giving real eigenvalues for the combination.

A topologically regular grid has quadratic precision and a C2 limit surface. However, the support region has a fractal boundary and normally there is no closed form limit surface even piecewise. The scheme is not generated by any box-spline.

There is an interpolating equivalent, in which the new vertices near the centres of the original facets are given by a linear combination of the three vertices of the facet and the three far vertices of the adjacent triangles. In [17] Guskov investigates both this and the more general case where each vertex-vertex also depends on the original vertex and all its neighbours.

The claimed advantage of such schemes is that they provide for smoother gradation of density if part of the polyhedron is refined and other parts are not.

12.8.2. Subdivision over semiregular lattices

Some of the box-splines give schemes over semiregular lattices. For example, the 4-direction quadratic box spline described by Sabin in [30] and independently by Peters and Reif in [22]. This turns out to have a really simple construction: create a new vertex at the midpoint of each old edge; join these new vertices by a new face in each old face and a new face across each old vertex. The resulting topology is shown in figure 5(b).

However, another scheme by Velho[36] deliberately uses semi-regularity in the same hunt for less abrupt density transitions, and designs its coefficients towards those ends. Figure 12.6(b) shows how easy this approach makes the regional variation of subdivision level.

12.8.3. Dual schemes and contact elements

The term *dual* is sometimes used to denote a scheme like Doo-Sabin or those generated by the other even degree tensor product splines. This is dual in the topological graph-theory sense. However, it is not clear how the square root of 3 and semiregular schemes fit into this classification.

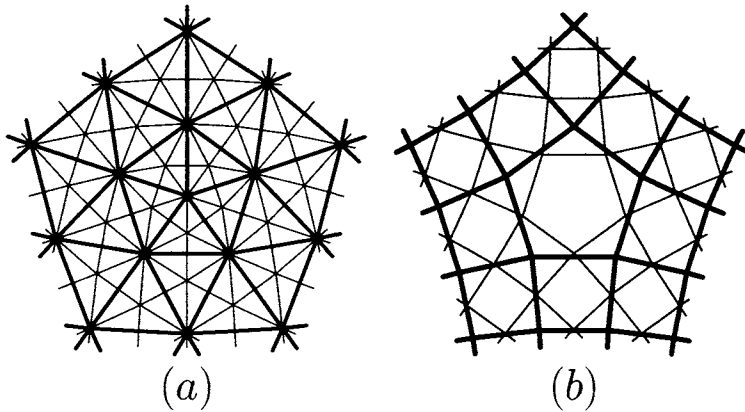


Figure 12.5. Topologies of the (a) $\sqrt{3}$ and (b) 'simplest' schemes.

There are also dual schemes in the algebraic geometric sense, where instead of new vertices being generated from linear combinations of old ones, with limit points and tangent planes being derived from the eigenanalysis, new planes are generated from linear combinations of old ones, and new planes and tangent points are derived from the eigenanalysis. The first of these was described by Wartenburg [37], who showed that the dual of the Doo-Sabin scheme gave an interpolating scheme for convex data.

It is also possible that schemes in which initial contact elements (the combination of a point and its tangent neighbourhood) are combined to give new ones may give a better trade-off between degree and size of template than current schemes.

12.8.4. Unequal intervals

In the use of standard 'rectangular' B-spline surfaces, the ability to make short-wavelength changes in a long surface without affecting the surface in the large is very valuable. In the subdivision surface context something similar is offered by the ability to modify the vertices at a late stage in the subdivision. This is called multi-resolution editing and is described in chapter 14. However, this does not have quite the flexibility and ease of control offered by knot insertion in the unequal interval B-spline context.

There have been a number of attempts to enhance the subdivision process in this direction. deRose[10] focussed on the ability to make certain edges within the refined surface relatively sharp, following the ideas of [18] which allows specific edges to be turned into sharp creases. Sederberg et al. [32] described a more general technique for associating 'knot intervals' with the edges of the mesh and for propagating these to successive refinements. There are still problems in the detail of this last technique, notably in the behaviour round singularities of high valency [29], but I expect to see variants emerge which tune the propagation to solve this problem.

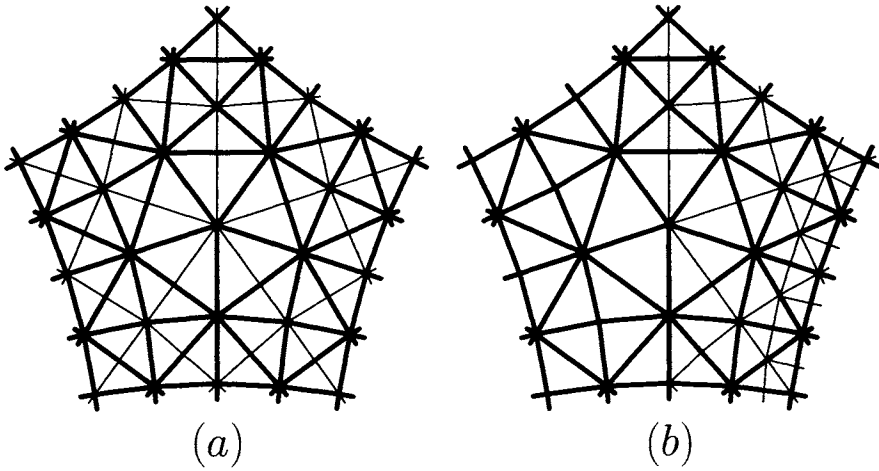


Figure 12.6. Topology of 4-8 semiregular subdivision

12.8.5. Artifact analysis

Artifacts are clearly the result of aliasing effects, but we cannot get rid of them by the usual graphics techniques of super-sampling, because the data on the coarse grid is all that we have.

In general some kinds of bivariate Fourier analysis will be applicable.

However, there appear to be three cases through which we can approach the analysis of artifacts in subdivision surfaces. The first we can call *longitudinal artifacts*, which is essentially the univariate artifact problem: data points equally spaced around a circle give a limit curve with maxima and minima of curvature. This can be addressed by looking at the result of subdividing *extruded data* on a regular grid, where all the values on any one of the generators have the same value.

We can also address *lateral artifacts* the same way, but looking to see whether the result of subdivision is still extruded data. There are schemes (mostly interpolating square root schemes) where the curvature across the dinosaur’s back can cause ripples along it.

The third case will undoubtedly take most work, either in finding an elegant widely applicable analysis technique or else in working through a large number of particular cases. This is the *rotational artifact* problem, where ripples appear around an extraordinary point of high valency. The essential problem is that because artifacts are mainly caused at the first iteration, we cannot assume that the extraordinary points are isolated.

12.9. CONCLUSIONS

Recursive refinement may not be the only technology addressing the need to provide automatic maintenance of continuity over surfaces of arbitrary topology – alternatives are to define *n*-sided patches suitable for incorporation in B-spline-like networks or to arrange the joins of conventional patches other than four at each vertex – but it currently appears

to be the contender in which most interest is being taken.

It also provides fascinating mathematical problems in the analysis of continuity at singularities and in the devising of new schemes giving a better trade-off between continuity, artifacts and template size.

REFERENCES

1. A.A. Ball and D.J.T. Storry. A matrix approach to the analysis of recursively generated B-splines. *Computer-Aided Design*, 18:437–442, 1986.
2. A.A. Ball and D.J.T. Storry. Conditions for tangent plane continuity of recursively generated B-splines. *ACM Transactions on Graphics*, 7:83–102, 1988.
3. R.E. Barnhill and R.F. Riesenfeld, editors. *Computer Aided Geometric Design*. Academic Press ISBN 0-12-079050-5, 1974.
4. E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, 1978.
5. Cavaretta, W. Dahmen, and C. Micchelli. Stationary subdivisions. Vol. 453 of *Memoirs of AMS* 1991 as reported to me by A. Levin, personal communication, 1999.
6. G.M. Chaikin. An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.
7. P. Davis. *Circulant Matrices*. John Wiley, ISBN 0-471-05771-1, 1979.
8. G. Farin. Chapter 3 of *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, ISBN 0-12-249050-9, 1988, and later editions.
9. G. de Rham. Collected works pages 89–97 and 690–727, 1947–1957 as reported by C. de Boor in Cutting corners always works. *Computer Aided Geometric Design*, 4:125–132, 1987.
10. T. de Rose, M. Kass and T. Truong. Subdivision surfaces in character animation. *SIGGRAPH*, 85–94, 1998.
11. D.W.H. Doo. A subdivision algorithm for smoothing down irregular shaped polyhedrons. *Proc. Interactive Techniques in Computer Aided Design*, pages 157–165, (IEEE Bologna), 1978.
12. D.W.H. Doo and M.A. Sabin. Analysis of the behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10:356–360, 1978.
13. N. Dyn, J. Gregory, and D. Levin. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9:160–169, 1990.
14. N. Dyn, D. Levin, and J.A. Gregory. A 4-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, 4:257–268, 1987.
15. G. Deslauriers and S. Dubuc. Symmetric iterative interpolation processes. *Constructive Approximation*, 5:49–68, 1989.
16. A.R. Forrest. Notes of Chaikin's algorithm. *CGM74-1*, University of East Anglia. 1974.
17. I. Guskov. Multivariate subdivision schemes and divided differences. *Program for Applied and Computational Mathematics*, Princeton University, Princeton NJ08544, 1998 available from <http://www.cs.caltech.edu/~ivguskov>
18. H. Hoppe, T. deRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. *Computer Graphics*,

- 28:295–302, 1994.
19. L. Kobbelt. $\sqrt{3}$ -Subdivision. *SIGGRAPH*, 103–112, 2000.
 20. C.T. Loop. *Smooth Subdivision Surfaces Based on Triangles*. Master's thesis, University of Utah, 1987.
 21. J. Peters and U. Reif. Analysis of algorithms generalizing B-Spline subdivision. *SIAM J. of Numerical Analysis*, 35:728–748, 1998.
 22. J. Peters and U. Reif. The simplest subdivision scheme for smoothing polyhedra. *ACM Transactions on Graphics*, 16:34–73, 1997.
 23. H. Prautzsch. Freeform splines. *Computer Aided Geometric Design*, 14:201–206, 1997.
 24. H. Prautzsch. Smoothness of subdivision surfaces at extraordinary points. *Adv. Comp. Math*, 9:377–389, 1998.
 25. H. Prautzsch and G. Umlauf. A G^2 -subdivision algorithm. In G. Farin, H. Bieri, G. Brunnett, and T. deRose, editors, *Geometric Modelling*, pages 217–224, Computing Supp. 13, Springer Verlag, 1998.
 26. H. Prautzsch and G. Umlauf. A G^1 and a G^2 subdivision scheme for triangular nets. *J. of Shape Modelling*, 6:21–35, 2000.
 27. U. Reif. A unified approach to subdivision algorithms near extraordinary points. *Computer Aided Geometric Design*, 12:153–174, 1995.
 28. R.F. Riesenfeld. On Chaikin's algorithm. *IEEE Computer Graphics and Applications*, 4:304–310, 1975.
 29. K. Qin and H. Wang. Eigenanalysis and continuity of non-uniform Doo-Sabin surfaces. *Proc. Pacific Graphics '99*, pages 179–186 and 324, IEEE Computer Society PR00293 ISBN 0-7695-0293-8, 1999.
 30. M.A. Sabin. Recursive division. In J.A. Gregory, editor, *The Mathematics of Surfaces I*, pages 269–282, Clarendon Press ISBN 0-19-853609-7, 1986.
 31. M.A. Sabin. Cubic recursive division with bounded curvature. In P.J. Laurent, A. Le Mehaute, and L.L. Schumaker, editors, *Curves and Surfaces*, pages 411–414, Academic Press, ISBN 0-12-438660-1, 1991.
 32. T.W. Sederberg, J. Zheng, D. Sewell, and M. Sabin. Non-uniform recursive subdivision surfaces. *SIGGRAPH*, 387–394, 1998.
 33. C.E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, ISBN 0-252-72548-4, 1949.
 34. D.J.T. Storry. *B-spline Surfaces over an Irregular Topology by Recursive Subdivision*. PhD thesis, Loughborough University, 1985.
 35. D.J.T. Storry and A.A. Ball. Design of an n -sided surface patch. *Computer Aided Geometric Design*, 6:111–120, 1989.
 36. L. Velho. Quasi 4-8 subdivision. *Computer Aided Geometric Design*, 18:345–357, 2001.
 37. I. Wartenburg. Dual Doo-Sabin subdivision surfaces. *Curves and Surfaces*, presented at AFA conference, St Malo, 1999.
 38. D. Zorin, P. Schroder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. *SIGGRAPH*, 189–192, 1996.
 39. D. Zorin and P. Schroder. Subdivision for modeling and animation. *SIGGRAPH Course Notes*, 1999.

Chapter 13

Interrogation of Subdivision Surfaces

Malcolm Sabin

This chapter deals with how surfaces may be interrogated by the subdivision paradigm, in which anything too complicated to deal with simply is divided into sub-problems, and the same approach applied to the subproblems.

13.1. SUBDIVISION SURFACE INTERROGATIONS

Interrogation is the process of determining from a surface various properties which are required for analysis of various aspects of the product which the surface helps to describe, and for creating data to assist in the manufacturing process. The survey paper [13] covers a range of computational approaches.

Display is the first example which comes to mind, being useful both for aesthetic analysis and for marketing. Simple images can be generated by faceting the surface and sending the facets to a Z-buffer display: highly realistic renderings need ray-casting, the calculation of the points where rays from the eye first meet the surface.

Sharper graphical analysis has as examples the use of images colour modulated to show the variation of curvatures, or of nests of cross-sections (contours) which skilled loftsmen can use to assess the quality of a surface.

For analysis of strength and stiffness of the product we require to be able to generate appropriate grids across the surface, varying in local density with the expected spatial frequencies of the displacement under load, and being able to find nearest points on the surface to given points (estimated by the mesh generation process) is a key tool.

For manufacture, cross-sections again give the shapes of templates or of internal structural members, and intersections of surfaces are required to determine trimming of skin panels. Offset surfaces are traditionally required for the determination of tool-centre paths for numerically controlled machining, although the recent trend in NC software towards the use of dense triangulations as the master for tool-path determination may reduce the need for that a little.

Such requirements typically take much more CPU than the first setting up of a surface,

and so it is important that they be efficiently implemented. In the solid modelling context, it is also important that they be robust in their operation.

The literature is remarkably sparse in describing these issues, with a few tens of papers while there are hundreds dealing with surface descriptions and thousands dealing with the mathematics of splines.

In Geisow's seminal work[7] (more accessibly available in a summary by Pratt and Geisow[11]), various approaches to interrogation were described in some detail, the front-runner at the time being Newton iteration[17], used directly for multivariate interrogations such as nearest-point, and embedded in a marching procedure for determining univariate interrogations such as cross-sections and intersections. If a piece of surface were known to have only one open piece of intersection across it, the process of approximation by successive halving of the intersection curve was also robust.

A little later, Sederberg[14] compared different techniques for finding intersection points of 2D curves. At that time subdivision came out poorly: robust, but relatively slow.

There are reasons to believe that this judgement needs to be reconsidered. The computers on which we now base CAD/CAM systems are much faster than those available then, and, more significantly, the amounts of real memory are orders of magnitude larger. We are now approaching the era of the Gigabyte PC, compared with the tens of Kilobytes which were available when other methods were developed.

We can now afford the memory to subdivide a surface when it is loaded, so that the subsequent interrogations become much cheaper. There are quantitative arguments, based on the relative speed of disc access and processor arithmetic, which can determine the level of detail which should be retained on disk and other arguments can determine the optimum level of detail to be precomputed and retained in memory.

13.2. HISTORICAL BACKGROUND

One of the first ways in which subdivision impinged on the CAGD community was through graphics. The hidden surface problem spawned a large number of different approaches during the early 1970s, and one of the novelties then was the idea of Warnock[19], whose algorithm was essentially 'if the picture is simple, draw it; if not, use this same approach for a quarter at a time.'. This was an image space algorithm and the subdivision concept was soon applied in object space algorithms and then in parameter space algorithms for surfaces.[18,?] This was stimulated by the development of knot insertion algorithms, notably the Oslo algorithm [6] and the Boehm algorithm [2].

The motivation for these was based on two observations:-

1. That for any piecewise polynomial curve, typically represented by the combination of B-spline basis functions with point-valued coefficients, knots can be inserted at places where there are no actual discontinuities of derivative, and that if enough knots are inserted at the same place, the original curve falls into two pieces, each represented by its own subset of control points.
2. That such a curve always lies inside the convex hull of its control points.

If we wish to determine such things as the intersection points of curves, or the intersections of curves or surfaces with planes or with each other, these two properties give an

elegant recursive approach.

As a simple example, if we wish to calculate the points of intersection of a curve with a plane, we have a recursive algorithm exactly echoing the Warnock structure

‘If the curve is simple get the intersection directly; if not, apply this same algorithm to two halves of it, taking the union of any points which are found in the halves’

There is one addition, which makes it much more efficient than just dividing the curve into a large number of simple pieces ab initio and considering them one at a time. If, at each level, we test whether the convex hull crosses the plane, we can discover whether the curve lies entirely on one side of the plane, in which case there is no intersection, and no further subdivision need take place.

Such an algorithm promises both efficiency and robustness.

Algorithms in this style are capable of supporting the wide range of interrogations which are necessary in industrial use of surfaces represented on the computer; they give the very high levels of robustness which are essential if the surface handling is embedded inside a solid modelling context, and with the dramatic growth of real memories in the last decade or two are potentially the fastest available approach.

There is, however, a significant amount of sophistication which can and must be added to the basic idea in order to turn it into a widely applicable tool, and this chapter deals with that detailed technology by taking the simple approach and exploring how it needs to be tuned.

13.3. THE CONVEX HULL PROPERTY

The key factor in efficient use of subdivision in, for example, surface-surface intersections is the culling of subproblems which cannot contribute to the solution. This, carried out during the subdivision, is what avoids the cost of comparing everything with everything else for a lot of small surface pieces.

The property which, for Bezier and B-spline surfaces, supports this test, is that each piece of curve or surface is known to lie within the convex hull of its control points. If two convex hulls do not overlap, then those two pieces of surface cannot intersect.

13.3.1. Other hulls

Unfortunately, the convex hull even of a discrete set of points is not a trivial thing to represent or compute, and checking whether two such hulls intersect is moderately complex.

Bounding boxes

Most practical implementations of these ideas have therefore used a much simpler construct, the bounding box, which is trivial to represent and construct. Checking whether two boxes overlap is also trivial.

This simplicity is paid for by the fact that it is easy for two bounding boxes to overlap while the corresponding convex hulls do not. In such circumstances the test gives a *false positive* and the outer algorithm will think that it needs to subdivide when in fact it does not. The tree of subdivision gets bushier and time is wasted on exploring paths which do not contribute to the result.

The theorem which is being used implicitly here is that every point in the convex hull is also in the bounding box. This is simply proved by observing that every point in the intersection of two solids is also in each of those solids. The convex hull is the intersection of all supporting half-planes, while the bounding box is the intersection of the six supporting axially-oriented halfplanes. Intersecting with the other half-planes can only remove points of the bounding box, never add any.

Quantised hulls

That theorem leads to an observation that maybe something of the simplicity of the bounding box could be combined with the relative tightness of the convex hull by taking a larger set of fixed support directions.

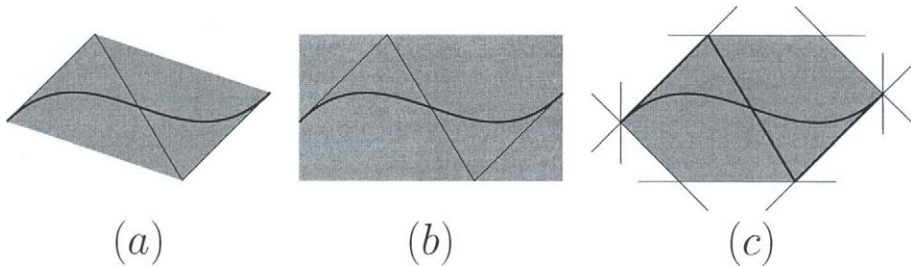


Figure 13.1. (a) convex hull, (b) bounding box, (c) 8-quantised hull of control points

Figure 13.1 shows the areas of the convex hull, the bounding box and the quantised hull with eight support directions for a typical Bezier curve. Note that in the particular geometry of figure 1(c) two of the eight support halfspaces do not actually trim the quantized hull. The probability of a positive ‘yes, it may intersect’ is in some sense proportional to the area, and it may be seen that the quantised hull removes some of the bounding box’s excess of area over the convex hull.

The representation of a quantised hull can just be an array H of support distances, one for each (oriented) direction. The construction algorithm for the quantised hull using n directions, \mathbf{v}_i , $i = 1 \dots n$ of a set of p points $\mathcal{P} = \{\mathbf{p}_j, j = 1 \dots p\}$ becomes

```

For each direction:  $\mathbf{v}_i$ 
Do  $h := -\infty$ 
  For each point  $\mathbf{p}_j$ 
  Do  $h := \max(h, \mathbf{v}_i \cdot \mathbf{p}_j)$ 
  Od
   $H[i] := h$ 
Od
  
```

The loops in this algorithm may be interchanged for higher performance if the points in \mathcal{P} are scattered in memory. The directions \mathbf{v}_i do not need to be unit vectors, and they

may be implicit, so that if they are regularly oriented (for example, the axial directions and the directions of cube face and solid diagonals) the inner products can be replaced by appropriate sums of the axial components.

Checking for overlap requires that we have the support directions occurring in oppositely oriented pairs. Suppose that the direction oppositely oriented to \mathbf{v}_i is $\mathbf{v}_{\bar{i}}$, and the support distances for two sets are held in H and \bar{H} . The overlap check then becomes

```

 $olap := true$ 
For each direction  $i$  while  $olap$ 
  Do If  $H[i] + \bar{H}[\bar{i}] < 0$ 
    Then  $olap := false$ 
  Fi
Od

```

Another algorithm important for high performance is that we can compute the quantised hull, HC , of the union of two quantised hulls, HA and HB , by just taking maxima of support distances.

```

For each support direction  $i$ 
  Do  $HC[i] = \max(HA[i], HB[i])$ 
Od

```

The bounding box is just a quantised hull with six support directions. Such experimental evidence as I have suggests that the tightness gained by adding more directions does not actually improve the overall performance of interrogations. The cost of comparing additional directions appears to outweigh the saving from fewer false positives, unless the cost of carrying out unnecessary subdivisions is very high. That experimental evidence is not particularly broadly based, and further experiment would be very valuable.

13.3.2. Hulls of non-positive bases

Although the convex hull enclosure property was initially seen as a major advantage of the Bezier and B-spline bases, Goldman and deRose [8] pointed out that all is not lost if the basis has local support.

Consider the condition number $L = \max_i \sum_j |\phi_j(t)|$ of a basis $\phi_i(t)$ for which $\sum_i \phi_i(t) = 1$. For positive bases this is equal to 1, while if some basis functions are somewhere negative it will exceed 1. We can determine a hull in which the curve itself lies by just expanding the hull of the control points by an offset ratio of $(L - 1)$ in each support direction.

13.3.3. Normal hulls

The hull of a pointset is a convex solid containing all the points in the set. It is also useful to have available some information relating to first derivatives, the tangent vector in the case of a curve, the normal vector in the case of a surface. Because we do not use parametric values when thinking of surfaces as subdivision surfaces, the most useful vectors are unit vectors. Although these lie on a (unit) sphere, often called the Gauss

Sphere, there is nothing preventing us from putting a box or other hull around them. Such a box is called a *Normal Hull* (or *Tangent Hull*), and allows us to make deductions based on local surface or curve orientation.

In principle, higher derivative information could also be captured into Hull form. For example, a *Binormal Hull* could bound binormal vectors, scaled by the local curvature, of a piece of curve. However, I have not seen such constructs in the literature.

13.3.4. Offset subdivision surfaces

Many enquiries need to address surfaces offset from some primary surface. The three main examples are

1. sections through the inside of a skin when the primary definition is of the outside.
2. machining paths computed as sections through the tool-centre surface offset by the cutter radii of a typical toroidal milling cutter.
3. intersection of two offset surfaces to determine the centre-line of a rolling-ball blend.

Such interrogations are not outside the domain of subdivision interrogations. A piece of offset surface can determine its hulls by reference to the primary definition. The trivial theorem here is that if a primary surface lies inside a hull H , the offset surface lies inside a version expanded by the appropriate offset distance in all directions.

However, this does not lead to efficient interrogations. As described in [12], it is necessary for the offset surface also to take the normals, either directly or through the normal hull, into account, and expand only by those parts of the offset form which are actually invoked by the normals in the piece of parent surface.

13.4. AN API FOR SUBDIVISION SURFACES

Just as the interrogation of parametric surfaces is facilitated by the identification of a small set of procedures which support a wide variety of interrogations, and which can be implemented relatively easily for any new surface formulation, we also need a programming interface for subdivision surfaces.

It is convenient to describe these in terms of methods for surface objects, though the actual implementation is not forced into the object-oriented paradigm.

The initial methods which a subdivision surface (or piece of surface), S , needs to support are:-

1. `simple: Surface → Boolean`
2. `approx: Surface|simple(.)=True → Polygon`
3. `hull: Surface → Convex Pointset`
4. `normal hull: Surface → Convex Vectorset`
5. `pieces: Surface → Set of surfaces`

Note that the pieces of a large surface do not need to hold copies of all their parent's data. These methods can be implemented using indirection into that data.

Curves have exactly analogous methods.

13.5. EXAMPLE INTERROGATIONS

In this section we take a sequence of interrogations, showing how each can be implemented in terms of the API, initially as described above, but enhanced when we encounter the need.

13.5.1. Z-buffer imaging

The problem is to divide the surface into polygons which are small where the surface is complex and large where it is almost flat. Those polygons are then sent to a Z-buffer display device using the procedure *polygondraw*(*polygon*).

The description here is in recursive terms

```

S.Draw
  If S.simple
  Then polygondraw(S.approx)
  Else For each  $S_i$  in S.pieces
    Do  $S_i$ .Draw
  Od
Fi

```

If we do not require the ability to use the representation stored in the display device to support dynamic rotation, or if the surface is part of the boundary of a solid (so that it cannot be seen from the inside), we can reduce the number of polygons sent to the display and also reduce the amount of subdivision by using the normal hull. If there is no ray from any part of the Hull in any of the directions in the Normal Hull which can meet the eye, that part of the surface need not be considered any further.

This is not a high-quality approach, since sending polygons of different resolution next to each other may result in glitches, where fragments of background are visible through the triangular holes between those polygons.

It is, however, extremely simple to implement, provided that a language is being used which supports the recursive calls.

13.5.2. Raycast

Here we seek the first point on a ray in which the ray penetrates the surface. We discover that recursion is not the best way to implement this function. Far better is to use a priority queue, maintained explicitly in the code. It then becomes possible to ignore completely those pieces of surface whose nearest point is further than the best intersection point already found, thus resulting in more effective culling of the tree.

The principle is that we are effectively doing a tree-search for a solution which is in some sense 'best'. Using recursion in the language forces a depth-first search, whereas heuristic search, well-described in chapter 3 of Nilsson[10] can limit the search to a small fraction of the tree.

We set up a pair of planes, more or less perpendicular, which intersect in the line, and also identify a direction along the ray. Each surface piece can be bounded (using its hull) with respect to that direction. If it lies behind the eyepoint it should be ignored. If it lies further than the nearest solution point found so far, it can also be ignored.

With this in mind we choose as prioritising criterion between pieces of surface the one whose nearest point in the ray direction is nearest. If that piece of surface is actually cut by the ray, it will have most effect on eliminating other pieces early later.

```

S.Raycut() → Point
  put S on priority queue
  set best so far to ∞
  set up ray planes and ray direction
  Until queue is empty
  Do  take piece S1 from queue
      If S1.hull is entirely behind the eye
      Then skip it
      Elseif S1.hull is entirely further than best so far
      Then skip it
      Elseif S1.hull does not intersect either plane
      Then skip it
      Elseif S1.simple
      Then P := cutpolygon(S1.approx)
          If P closer than best so far
          Then update best so far
          Fi
      Else For each piece S2 of S1.pieces
          Do add S2 to priority queue
          Od
      Fi
  Od

```

In fact it may be best to carry out the check for a piece being behind the eye before the piece is added to the queue, since this keeps the number of pieces on the queue to a minimum, which may reduce the cost of queue-maintenance. The comparison with best-so-far, however, must be carried out as the piece is taken off the queue, since best-so-far may have improved since it was added. The test could be carried out, of course, both when the piece is added and when it is taken off the queue.

This use of a priority queue is relevant whenever we are seeking the best of a number of candidates all of which would normally be found by the tree search. In fact it can be generally applied, as if the priority criterion is recentness of addition the queue becomes a stack, with very little overhead.

The optimum hull is a quantised hull with four directions, being the normals to the two planes which intersect in the ray. If all the rays are parallel it may be worth setting up hulls specially for this purpose. However, if that much pre-processing is being done, it would probably be better to be considering a scan-line organisation to exploit the spatial coherence in the data.

13.5.3. Plane section

The plane section routine is probably the most heavily exercised of all interrogations apart from the rendering of the shape.

The main issue here is that the output is a univariate point set, which requires a finite representation. There is much potential debate here, and so it will be assumed that the curve will be represented as the sequence of short straight line segments in which the plane intersects the flat approximations to parts of the surface. We need to be a little careful, since the intersection can have many loops, and so a simple array will not be adequate. During the computation there may be many open pieces of curve which get linked together subsequently.

Without going into details, a doubly linked list of pieces, each holding the coordinates of both ends, indexed appropriately to give fast access to unpaired ends during the matching process, is an adequate basis which can be optimised to suit the programming context.

When a new piece of curve is determined it needs to be linked with any neighbours which have already been computed. It is vitally important that this should happen during the unwinding of the recursion, since otherwise there is a large sorting to do.

The matching needs to be based on what pieces of surface were being cut to give the matched curve segments

```

S.PlaneCut(Plane:F)→ Curve
  set output curve C empty
  put S on priority queue
  Until queue is empty
  Do take piece S1 from queue
    If S1.hull does not intersect plane
    Then skip it
    Elseif S1.simple
    Then C1 := polygonsect(S1.approx,F)
      If C1 exists
      Then merge C1 into output C
      Fi
    Else For each piece S2 of S1.pieces
      Do add S2 to priority queue
      Od
    Fi
  Od

```

Which way to split ?

An interesting issue which comes alive in this context is the question of which way to split a parametric surface, since a given piece can be split in either the u -direction or the v -direction. The simple heuristic here is to split whichever direction is longer in the direction of the normal to the plane, since this will have most effect on minimising the hulls of the pieces. However, there is a counter-argument that suggests that the split should be to cut the direction which is contributing most to the non-simplicity of the piece, since this has most effect on helping to terminate the recursion. Yet another argument says

that just cutting the longest direction is good from most points of view, and if the pieces are being retained in memory in case of future usefulness (as will be suggested below) this is a reasonable compromise. However, experimental evidence on this topic would be valuable.

Switching to other methods

It was mentioned above that there are other methods which can be applied robustly if we know that the piece of surface has only one piece of cross-section and that piece is open. In particular, we can set up an approximation to the piece of curve, and progressively refine it using parametric access to the surface.

In order to exploit this we need to be able to use a simplicity criterion which is based on the normal hull. This was pointed out by Sederberg et al. in [15,?]. If the normal hull does not contain the origin, there is a direction D along which all normals have a positive dot product. This implies that there is another direction $D \wedge N_p$ along which all oriented tangents to the plane section have a positive dot product. If the tangent to a curve always has a positive component in some direction it cannot form a closed loop. It can consist of a number of open pieces, but we can find these by scanning round the boundary of the piece of surface.

13.5.4. Surface-surface intersection

The intersection of two surfaces is the virility symbol of CAGD systems. First described in the literature by Barnhill et al [1], though commercial systems had their own methods operational somewhat earlier, there have been many methods propounded.

The subdivision algorithm for this has a new feature, not present in the interrogations described above. This is that what goes on to the stack or priority list is not a piece of curve or surface, but a *sub-problem*. We can, in fact, take this as the general case, since in the previous cases the subproblem is characterised by the piece of curve or surface and the data which is common to all the subproblems.

```

Surfcut(Surface: $S_1, S_2$ ) → Curve
  set output curve  $C$  empty
  put [ $S_1, S_2$ ] on priority queue
  Until queue is empty
  Do  take problem [ $S_1, S_2$ ] from queue
      If  $S_1$ .hull does not intersect  $S_2$ .hull
      Then skip it
      Else If  $S_1$ .simple
          Then  $C_1 := \text{specialcase}(S_1.\text{approx}, S_2)$ 
              If  $C_1$  exists
              Then merge  $C_1$  into output  $C$ 
              Fi
          Elseif  $S_2$ .simple
          Then  $C_1 := \text{specialcase}(S_1, S_2.\text{approx})$ 
              If  $C_1$  exists
              Then merge  $C_1$  into output  $C$ 
              Fi

```

```

      Fi
      Else choose which surface to split
        If  $S_1$  to split
        Then For each piece  $S_{1i}$  of  $S_1$ .pieces
          Do add  $[S_{1i}, S_2]$  to priority queue
          Od
        Else For each piece  $S_{2i}$  of  $S_2$ .pieces
          Do add  $[S_1, S_{2i}]$  to priority queue
          Od
      Fi
    Fi
  Od

```

Note that in order always to generate pieces of curve with compatible orientations, the sequence of S_1 and S_2 is always maintained in the subproblems.

Which surface to split ?

The strategy here appears to be clear, that the larger of S_1 and S_2 should be the prime candidate, since if the hull of S_1 is entirely contained in S_2 splitting it will give two overlapping subproblems and a bushy tree. However, there is also the argument that the piece nearer simplicity should be split, so that the special case code, which intersects a surface piece with an approximation, can be entered sooner. Again, we need quantitative experimental evidence, which does not appear to be in the current literature.

Co-simplicity

Just as the plane section code has the option of leaving the subdivision paradigm early as soon as a condition on the normal hull is met, there is the possibility of terminating the recursion early here also. The condition is very similar, that the oriented tangents to a curve piece should have a positive dot product with some fixed direction. However, the oriented tangents are now the cross products of two normals, each of which lies somewhere in its own normal hull.

The hull of such vectors can in principle be evaluated in the case of B-spline surfaces, from the property that since the derivative of a B-spline is a B-spline, and the product of two B-splines is a B-spline, each normal is a B-spline and the cross-product of the two normals is a B-spline, so that we can make a hull by taking the hull of its control points.

However, this is not really practical, because the product B-splines are almost always strongly degenerate, typically being more-or-less multiBezier representations.

It is probably far better to make a hull of a cross-product from the hulls of the two vectors. This is slightly sloppier, but a great deal simpler, since the input hulls will have fewer points to deal with.

13.5.5. Silhouette

The silhouette is a curve containing points where the ray from an eye point is perpendicular to the local normal. It is included here in order to exemplify the normal hull being used

within the main test, not just in the test for simplicity.

Here the check that a subproblem cannot contribute to the solution is that the dot product of any ray from the eye point to a point in the position hull with any normal in the normal hull has a constant sign. Again, the sensible way to compute this is by taking the hull of the dot product of two hulls rather than by setting it up algebraically.

13.6. PERFORMANCE ISSUES

We have seen in the above how certain additions to the basic API can make interrogations more efficient.

- decide which to split

A good heuristic when either of two objects might be split is to split whichever has the hull of larger dimension.

- decide which way to split

Again, splitting the direction of larger dimension will normally be a good choice.

- decide where to split

In chapter 7 of Bowyer and Woodward's book [3] on solid modelling, spatial subdivision is used as a technique for interrogations, and it is pointed out that a good choice of split point is that which tries to get all of the result into one half, while making that half as small as possible.

- split to Bezier edge conditions

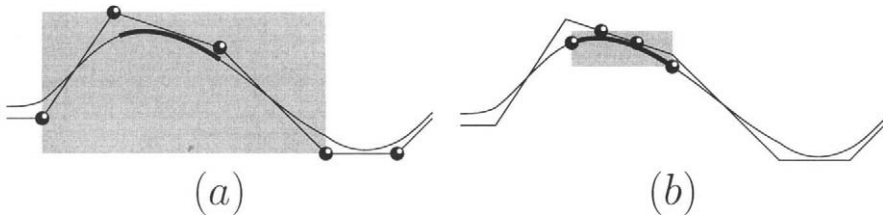


Figure 13.2. (a) cubic B-spline bounding box (b) cubic Bezier bounding box

A problem with B-spline control polygons is that a particular piece of curve or surface is defined to lie within the hull of a set of control points which span well outside the part of the curve or surface defined. Once the piece of surface is small enough to fall between the original knot lines of the surface, it is therefore better to use the Bezier representation rather than the B-spline. Figure 13.2 shows dramatically how much larger the Bspline box can be for the same piece of curve. At coarser subdivisions it is more expensive but tighter to split a B-spline surface into pieces by inserting enough knots to give the pieces Bezier end-conditions.

- keep pieces in case they are needed again

Particularly when relatively expensive subdivision (Bezier endconditions, normal hulls) are being used, the availability of large real memories suggests that, since often interrogations are made in large batches, it may be useful to retain the subdivided data and its hulls, so that a subsequent interrogation may bypass the recomputation.

To take full advantage of this would involve marking the pages in which this data is held, so that the operating system could avoid writing them to disc if the page is swapped out, leaving a mark, so that if the data is required again it can be recomputed. (Recomputation will typically be faster than writing it out and rereading it, and using a smaller swap-file may speed the time taken for the disc heads to find the right track.) Unfortunately operating systems do not normally permit their application programs such control over the use of real memory.

- update the hulls of parents when a split takes place.

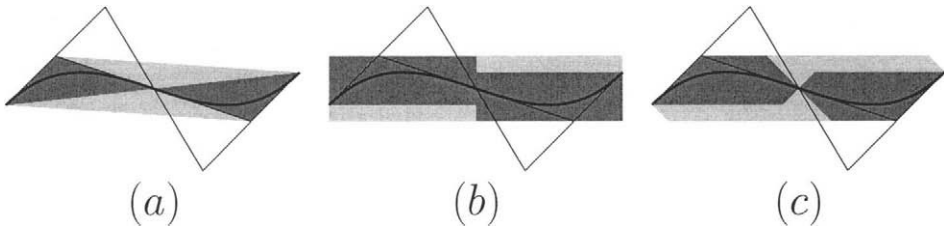


Figure 13.3. (a) convex hull, (b) bounding box, (c) 8-quantised hull updated

This concept is due to Cameron[4], who identified it in the context of solid modelling for robotics, calling it *S-bounds*. Figure 13.3 shows how in each case: convex hull, bounding box and quantised hull, the updating significantly reduces the area, and therefore the chance of a false positive. In fact in the limit the result is the true hull of the curve itself, rather than of its control points.

- reassemble pieces of result on the way out

If the pieces of a univariate curve result are not processed as soon as possible, a large sorting task may be required later. The best strategy is to label the ends of each sequence of line segments with the split which created the edge on which they lie, and then index these sequences from those splits. That way there is very little hunting to do in order to find the mate for a given end. Keeping all line segments oriented, so that it is only necessary to find a matching end for a given start or vice versa also reduces the number of comparisons which are necessary to make a given match. This issue is described in some detail in Nasri [9] pp190–196. Care in this area also covers the issue that two consecutive segments may be found with slightly

different end-points because the surfaces being cut may be subdivided to different extents. The output coordinates can always be from the finer refinement.

- use an explicit priority queue instead of an implicit stack

This is primarily for situations such as nearest point or ray-casting, where we want a particular solution among many that could exist. However, it can also help to reduce the reassembly costs if it is used to steer the solution into geometric coherence. For example, if the parts of an iso- x cross section are generated in sequence of increasing y , connections of low- y pieces will tend to be made earlier, before the high- y pieces are added to the result, thus speeding (slightly) the reassembly process.

- use normal hulls to switch to other methods

If further experiment shows that marching or homotopy methods really are faster than continuing the refinement, normal hulls provide the capability to make the decisions as to when to switch.

With all these ideas in mind, we can see that a sensible approach is to pre-process the data with a sensible subdivision sequence for each surface. It is certainly faster to do a certain amount of preprocessing and generate the hull data from the bottom up than to determine every hull by reference to the original data, and the S-bounding then happens automatically without complicating the substantive interrogations.

The higher level hulls then become essentially a multiresolution index for finding the relevant lower level pieces.

The creation of a fine-level model with its hulls from a coarse model is almost certainly faster than reading the fine data in from backing store. (But doing this probably precludes the use of variational refinement.) Alternatively we can view the subdivision on the fly as lazy evaluation.

13.7. CONCLUSIONS

The subdivision paradigm carefully implemented can form the basis for extremely robust and fast code for a wide range of interrogations.

For peak performance there are a number of trade-offs which are not as yet well-documented in the literature, and one hopes that experimental studies will be regarded as worth-while by researchers in the near future.

REFERENCES

1. R.E. Barnhill, G. Farin, M. Jordan, and B.R. Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4:3-16, 1987.
2. W. Boehm. Inserting new knots into B-spline curves. *Computer-Aided Design*, 12:199-201, 1980.
3. A. Bowyer and J. Woodwark. *Introduction to Computing with Geometry*. Information Geometers, Winchester, ISBN 1-874728-03-8, 1993.
4. S.A. Cameron. *Modelling Solids in Motion*. PhD thesis, Edinburgh University, pages 77-92, 1984.

5. E.A. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Report UTEC-CSc-74-133, University of Utah, 1974.
6. E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-splines and subdivision techniques in computer aided design and computer graphics. *Computer Graphics and Image Processing*, 14:87-111, 1980.
7. A.D. Geisow. *Surface Interrogations*. PhD thesis, School of Computing Studies, University of East Anglia, 1983.
8. R.N. Goldman and T.D. deRose. Recursive subdivision without the convex hull property. *Computer Aided Geometric Design*, 3:247-266, 1986.
9. A.H. Nasri. *Polyhedral Subdivision Methods for Free-Form Surfaces*. PhD thesis, School of Computing Studies, University of East Anglia, 1984.
10. N.J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw Hill, ISBN 07-046573-8, 1971.
11. M.J. Pratt and A.D. Geisow. Surface/surface intersection problems. In J.A. Gregory, editor, *The Mathematics of Surfaces*, Clarendon Press, Oxford, pages 117-142, 1986.
12. M.A. Sabin. Recursive division interrogation of offset surfaces. In J. Warren, editor, *Proceedings of SPIE 1830 Curves and Surfaces in Computer Vision and Graphics III*, pages 152-161, 1992.
13. M.A. Sabin. *Numerical Geometry of Surfaces*. Acta Numerica 1994 CUP, pages 411-466, 1994.
14. T.W. Sederberg and S.R. Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18:58-64, 1986.
15. T.W. Sederberg and R.J. Meyers. Loop detection in surface patch intersections. *Computer Aided Geometric Design*, 5:161-172, 1988.
16. T.W. Sederberg, H. Christiansen and S. Katz. Improved test for closed loops in surface intersections. *Computer-Aided Design*, 21:505-508, 1989.
17. N.E. South and J.P. Kelly. *Analytic Surface Methods*. Internal report, NC development unit, Ford Motor Company, Dearborn MI, 1965.
18. I.E. Sutherland, R.F. Sproull, and R.A. Schumaker. A characterisation of ten hidden surface algorithms. *Computing Surveys*, 6:1-55, 1974.
19. J.E. Warnock. *A Hidden-line Algorithm for Halftone Picture Representation*. Univ. of Utah, Computer Science Department, Report TR4-5, 1968.

Chapter 14

Multiresolution Techniques

Leif P. Kobbelt

The term *multiresolution techniques* refers to a class of algorithms that decompose a given geometry into its global shape and detail information on different levels of resolution. The representation of an object on several levels of detail which are defined relative to each other gives rise to a number of applications that exploit the hierarchical nature of the representation. In this Chapter we explain the theoretical background of the multiresolution transform and show how the basic concepts can be generalized to arbitrary freeform surfaces.

14.1. INTRODUCTION

One standard approach to facilitate the handling of large amounts of data is to introduce *hierarchical structures*. Hierarchies usually provide fast access to relevant parts of a dataset which increases the efficiency of any algorithm processing the data. In the context of geometric datasets, hierarchical representations provide, besides spatial decomposition, access to different *resolutions* of the underlying curve or surface. Depending on the specific application, the term resolution refers to a certain level of *complexity* or to the amount of *geometric detail* (cf. Fig. 14.1).

If the underlying surface representation is based on splines (cf. Chapter 6) or subdivision surfaces (cf. Chapter 12) then the *topological* level of detail characterizes the degree of refinement of the control mesh while the *geometric* level of detail rates the size of the smallest features and dents on the corresponding continuous surface. If the surface representation is based on polygonal meshes the distinction is more obvious since (topologically) refined meshes can be very smooth (low geometric detail) or highly detailed.

Multiresolution techniques exploit the additional information that becomes available through a level-of-detail (LoD) representation either by choosing an appropriate resolution for given quality (complexity) requirements or by considering the difference between two levels of detail as a separate geometric frequency band which contains the detail information that is added or removed when switching between hierarchy levels.

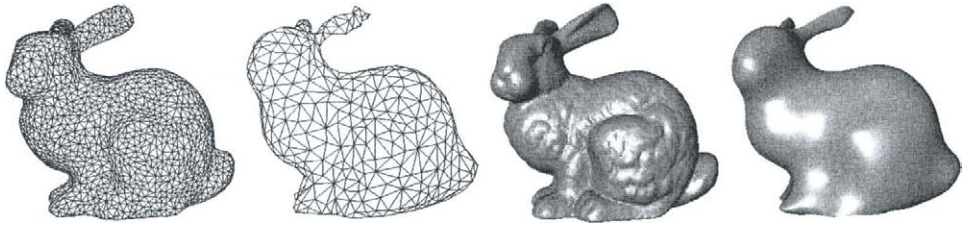


Figure 14.1. For geometric models we distinguish two different types of hierarchies. *Topological hierarchies* provide different levels of complexity (left) while *geometric hierarchies* provide different levels of geometric detail information. For spline representations, the link between both hierarchies is established by the basis functions (blending functions) which are associated with the control vertices. For pure polygonal mesh representations there is no canonical way to derive *smooth* meshes from *coarse* ones.

In this chapter we will first explain the general theoretic set-up for multiresolution representations for curves and surfaces. Based on this formal description we will then discuss specific algorithms and their applications. We will start with the classical representation of a freeform object as a vector-weighted superposition of scalar valued basis functions. Later we generalize this concept to non-nested hierarchies where explicit basis functions are no longer available.

14.2. MULTIREOLUTION REPRESENTATIONS FOR CURVES

We will introduce the notion of multiresolution analysis and wavelets where we focus on those aspects which are most relevant for geometric modeling applications. For a more detailed exposition we refer to standard books, e.g., [1,4,27,28]

As shown in Chapter 6, a standard representation for freeform curves is based on the uniform B-splines

$$\mathbf{f} = \sum_i \mathbf{c}_i \phi(\cdot - i).$$

The concept of subdivision curves (Chapter 12) has its origin in the observation that the basis function ϕ satisfies a *two-scale-relation*

$$\phi = \sum_i \alpha_i \phi(2 \cdot - i) \quad (14.1)$$

which implies the inclusion $V = \text{span}\{\phi(\cdot - i)\} \subset V' = \text{span}\{\phi(2 \cdot - i)\}$. Consequently we can find a refined representation

$$\mathbf{f} = \sum_i \mathbf{c}'_i \phi(2 \cdot - i)$$

where the new coefficients \mathbf{c}'_i are computed by the *subdivision rules*

$$\mathbf{c}'_i = \sum_j \alpha_{i-2j} \mathbf{c}_j. \quad (14.2)$$

For B-splines and more general subdivision basis functions we usually have only a finite number of coefficients $\alpha_i \neq 0$. Hence the sum (14.2) only computes a linear combination of a constant number of coefficients \mathbf{c}_i which leads to very efficient *subdivision algorithms*.

Equation (14.2) defines an identity map from the coarse space V to the fine space V' (with “twice” as many basis functions). Obviously this map is not surjective since V' contains functions \mathbf{f}' which are not in V . Instead of using the basis $\{\phi(2 \cdot - i)\}$ for V' , we try to *extend* the basis $\{\phi(\cdot - i)\}$ by additional basis functions $\{\psi(\cdot - i)\}$ such that

$$V' = \text{span}\{\phi(2 \cdot - i)\} = \text{span}\{\phi(\cdot - i)\} \oplus \text{span}\{\psi(\cdot - i)\} = V \oplus W.$$

Once we have such a basis function ψ , every function $\mathbf{f}' \in V'$ can be rewritten as

$$\mathbf{f}' = \sum_i \mathbf{c}'_i \phi(2 \cdot - i) = \sum_i \mathbf{c}_i \phi(\cdot - i) + \sum_i \mathbf{d}_i \psi(\cdot - i)$$

which can be considered as the *reconstruction* of the function \mathbf{f}' from a coarse scale approximation (defined by the \mathbf{c}_i) and the detail information (defined by the \mathbf{d}_i).

Since the basis functions $\psi(\cdot - i)$ also lie in the space V' , there exists a linear combination which satisfies

$$\psi = \sum_i \beta_i \phi(2 \cdot - i)$$

and as a consequence we obtain the complete reconstruction rule

$$\mathbf{c}'_i = \sum_j \alpha_{i-2j} \mathbf{c}_j + \sum_j \beta_{i-2j} \mathbf{d}_j. \tag{14.3}$$

In the context of multiresolution transforms, the function ϕ is usually called the *scaling function* and ψ is called the *wavelet*. Both functions are typically designed such that V captures the low frequency component of V' and W captures the high frequency parts. Depending on the application, additional properties of ϕ and ψ might be required.

The minimum requirement for this reconstruction to be useful is that for a given function \mathbf{f}' we have to be able to efficiently compute well-defined values $[\mathbf{c}_i, \mathbf{d}_i]$ from the coefficients $[\mathbf{c}'_j]$. This inverse reconstruction operation is called the *decomposition*. For arbitrary basis functions ϕ and ψ the decomposition requires to solve the linear system defined by (14.3) which is computationally much more expensive than the reconstruction itself. Hence, one tries to balance the computation costs and looks for specific pairs of basis functions $[\phi(\cdot - i), \psi(\cdot - i)]$ that lead to faster and more effective decomposition operators.

For example, if we can find another set of functions $[\tilde{\phi}(\cdot - i), \tilde{\psi}(\cdot - i)]$ for which the following conditions hold

$$\langle \phi(\cdot - i), \tilde{\phi}(\cdot - j) \rangle = \langle \psi(\cdot - i), \tilde{\psi}(\cdot - j) \rangle = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{14.4}$$

and

$$\langle \phi(\cdot - i), \tilde{\psi}(\cdot - j) \rangle = \langle \psi(\cdot - i), \tilde{\phi}(\cdot - j) \rangle = 0 \quad \forall i, j \tag{14.5}$$

then the coefficients \mathbf{c}_k and \mathbf{d}_k can easily be computed by

$$\mathbf{c}_k = \langle \mathbf{f}', \tilde{\phi}(\cdot - k) \rangle \quad \text{and} \quad \mathbf{d}_k = \langle \mathbf{f}', \tilde{\psi}(\cdot - k) \rangle .$$

This setting is called the *bi-orthogonal wavelet setting* since the conditions (14.4) and (14.5) indicate that the two bases $[\phi(\cdot - i), \psi(\cdot - i)]$ and $[\tilde{\phi}(\cdot - i), \tilde{\psi}(\cdot - i)]$ are bi-orthogonal (or *dual*) to each other.

The reformulation of the decomposition operation in terms of inner products is not necessarily more efficient than solving the linear system directly. However, if we choose special basis functions, the situation simplifies significantly since we do not have to evaluate the inner products by integration.

Assume that there exists a two-scale-relation for the dual basis functions as well

$$\tilde{\phi} = \sum_j \lambda_j \tilde{\phi}(2 \cdot - j)$$

and

$$\tilde{\psi} = \sum_j \mu_j \tilde{\phi}(2 \cdot - j).$$

then, based on these relations, the inner products reduce to simple linear combinations of control coefficients, e.g.,

$$\begin{aligned} \langle \mathbf{f}', \tilde{\phi}(\cdot - k) \rangle &= \left\langle \sum_i \mathbf{c}'_i \phi(2 \cdot - i), \sum_j \lambda_j \tilde{\phi}(2 \cdot - 2k - j) \right\rangle \\ &= \sum_{i,j} \mathbf{c}'_i \lambda_j \langle \phi(2 \cdot - i), \tilde{\phi}(2 \cdot - 2k - j) \rangle \end{aligned}$$

and hence

$$\mathbf{c}_k = \sum_j \lambda_{j-2k} \mathbf{c}'_j \tag{14.6}$$

and

$$\mathbf{d}_k = \sum_j \mu_{j-2k} \mathbf{c}'_j \tag{14.7}$$

respectively. If the dual two-scale-relations have only finitely many non-vanishing coefficients then the decomposition has the same computational complexity as the reconstruction. As we will see in the next section there is a simple technique to construct such pairs of dual refinable basis functions.

Besides the mere applicability of the decomposition, we usually require additional properties of the transform. The obvious requirement is that the coarse approximation \mathbf{f} of \mathbf{f}' should be as close as possible. Here, the optimal solution can be obtained if we find basis functions $\psi(\cdot - i)$ which are orthogonal to the basis functions $\phi(\cdot - i)$ since in this case the approximation error becomes minimal in the least squares sense. This setting, where $V' = V \oplus W$ is an orthogonal decomposition, is called *semi-orthogonal wavelet setting*.

From the theoretical point of view, the optimal representation for a function \mathbf{f} or \mathbf{f}' would be with respect to an orthonormal basis, i.e., not only are the $\phi(\cdot - i)$ orthogonal to the $\psi(\cdot - i)$ but in addition the integer shifts of the basis functions themselves are orthogonal to each other. If this is the case then the dual basis $[\tilde{\phi}, \tilde{\psi}]$ is identical to the

primal one and the magnitude of the coefficients \mathbf{d}_i is proportional to their impact on the shape.

Requiring the set $[\phi(\cdot - i), \psi(\cdot - i)]$ to be an orthonormal basis is a very strong condition which eliminates most of the degrees of freedom [4]. Additional properties such as smoothness (differentiability), symmetry, and local support of the basis functions $\phi(\cdot - i)$ cannot be satisfied simultaneously anymore. Hence, in many applications, the semi-orthogonal setting is preferred and the additional degrees of freedom are used to obtain smooth basis functions with local support.

However in practice, it often turns out that even the semi-orthogonal setting is quite difficult to establish. Therefore, an even weaker condition is imposed on the basis functions $\phi(\cdot - i)$ and $\psi(\cdot - i)$. The motivation for this weaker condition is that the space V usually contains some low degree polynomials up to order n , i.e.

$$\forall k = 0, 1, \dots, n \quad \exists p_{i,k} \quad (\cdot)^k = \sum_i p_{i,k} \phi(\cdot - i),$$

to guarantee a certain approximation power. Instead of requiring that the basis function $\psi(\cdot - i)$ be orthogonal to *all* functions from V we can restrict ourselves to requiring that the basis functions $\psi(\cdot - i)$ be at least orthogonal to these low degree polynomials

$$\langle (\cdot)^k, \psi(\cdot - i) \rangle = \int_{-\infty}^{\infty} x^k \psi(x - i) dx = 0 \quad \forall i \quad \forall k = 0, 1, \dots, n$$

This property is called *vanishing moments*. For the basis functions $\psi(\cdot - i)$ to deserve the name “wavelets” we typically have to guarantee at least one vanishing moment (= average function value is zero).

14.3. LIFTING

Lifting [29–31] is a simple technique to construct a set of operators that perform a multiresolution decomposition and reconstruction. The underlying basis functions and their duals correspond to the bi-orthogonal setting and the lifting technique can be used to increase, e.g., the number of vanishing moments of the wavelets.

The starting point for the construction is an arbitrary refinable scaling function ϕ whose integer shifts $\phi(\cdot - i)$ span a coarse space V . For simplicity we assume that ϕ is *interpolatory*, i.e., $\phi(0) = 1$ and $\phi(i) = 0$ for all integers $i \neq 0$.

The squeezed basis functions $\phi(2 \cdot - i)$ span the refined space V' and we have $V \subset V'$ due to the two-scale-relation (14.1).

Suppose we are given a function

$$\mathbf{f}' = \sum_i \mathbf{c}'_i \phi(2 \cdot - i)$$

in the refined space V' . The simplest way to decompose \mathbf{f}' into a coarser approximation

$$\mathbf{f} = \sum_i \mathbf{c}_i \phi(\cdot - i)$$

plus detail information

$$\mathbf{f}' - \mathbf{f} = \sum_i \mathbf{d}_i \psi(\cdot - i)$$

is to apply *subsampling* to the sequence of coefficients, i.e.,

$$\mathbf{c}_i := \mathbf{c}'_{2i}. \quad (14.8)$$

Applying the subdivision operator (14.2) corresponding to the basis function ϕ we obtain *predicted* values

$$\mathbf{p}'_{2i+1} = \sum_j \alpha_{2i+1-2j} \mathbf{c}_j$$

on the refined scale which in general differ from the original values \mathbf{c}'_{2i+1} . Hence the detail coefficients \mathbf{d}_i can be defined as the prediction error

$$\mathbf{d}_i := \mathbf{c}'_{2i+1} - \mathbf{p}'_{2i+1} = \mathbf{c}'_{2i+1} - \sum_j \alpha_{2i+1-2j} \mathbf{c}'_{2j} \quad (14.9)$$

Equations (14.8) and (14.9) define the decomposition operator for a multiresolution analysis. The corresponding reconstruction operator is given by

$$\mathbf{c}'_{2i} = \mathbf{c}_i \quad \text{and} \quad \mathbf{c}'_{2i+1} = \mathbf{d}_i + \sum_j \alpha_{2i+1-2j} \mathbf{c}_j \quad (14.10)$$

which shows that we implicitly set the detail function ψ to $\phi(2 \cdot -1)$.

The construction so far has all the formal properties that we required. We have a pair of basis functions $[\phi, \psi]$ based on which we derive efficient decomposition and reconstruction operators. The dual basis functions never appear explicitly although the coefficients of their two-scale-relations show up in the decomposition rules.

As we mentioned earlier, we would like to have additional properties such as vanishing moments of the wavelet ψ since this guarantees better approximation quality of the original function and consequently smaller detail coefficients.

In the lifting scheme these additional vanishing moments can be obtained by modifying the initial choice for the function ψ . For this we add a linear combination of scaling functions $\phi(\cdot - i)$, i.e.

$$\psi^* := \phi(2 \cdot -1) + \sum_j \gamma_j \phi(\cdot - j)$$

where we choose the γ_j such that ψ^* has vanishing moments. Obviously $\gamma_0 = \gamma_1 = -\frac{1}{4}$ and all other $\gamma_j = 0$ yield at least one vanishing moment and even two vanishing moments if the function ϕ is symmetric. More non-zero coefficients γ_j can give additional vanishing moments.

The reason for using the $\phi(\cdot - i)$ to enhance the wavelet is that it enables a very simple implementation of the multiresolution transform. For the modified wavelet ψ^* we get a new decomposition operator

$$\begin{aligned} \mathbf{c}_i &\leftarrow \mathbf{c}'_{2i} \\ \mathbf{d}_i &\leftarrow \mathbf{c}'_{2i+1} - \sum_j \alpha_{2i+1-2j} \mathbf{c}'_{2j} \\ \mathbf{c}_i &\leftarrow \mathbf{c}_i - \sum_j \gamma_{i-j} \mathbf{d}_j \end{aligned}$$

and the corresponding reconstruction operator is obtained by simply inverting the order of the update steps and changing the signs

$$\begin{aligned} \mathbf{c}_i &\leftarrow \mathbf{c}_i + \sum_j \gamma_{i-j} \mathbf{d}_j \\ \mathbf{c}'_{2i+1} &\leftarrow \mathbf{d}_i + \sum_j \alpha_{2i+1-2j} \mathbf{c}'_{2j} \\ \mathbf{c}'_{2i} &\leftarrow \mathbf{c}_i. \end{aligned}$$

Obviously both operators have the same computation cost. Moreover, since the coarse scale coefficients \mathbf{c}_i and the detail coefficients \mathbf{d}_i are used for mutual updating we can overwrite the old values in each line of the implementation. Hence the whole computation can be done “in place” without using additional memory [31].

The original lifting scheme as proposed by Sweldens [29,30] is much more general than the construction presented here. In fact, every uniform wavelet transform can be factorized into a number of lifting steps [5]. Moreover, lifting can be applied even in non-uniform settings where the spaces V and V' are no longer spanned by uniformly spaced shifts of the same basis functions. For more details on the lifting scheme and its usage in different practical applications cf. [31].

14.4. GEOMETRIC SETTING

So far we considered the functional setting, i.e., the geometry was given as the graph of a scalar valued function defined over the real line (or plane). In a more general setup, we have to use parametric representations where the geometry is given by a vector valued function which maps some planar or non-planar parameter domain into 3-space, i.e., each of the coordinates is defined by a separate scalar valued function. As a consequence, control coefficients and detail coefficients are also vector valued.

If we are considering decomposition and reconstruction only then the processing of vector valued functions is done by simply applying the same operators simultaneously to all three coordinate functions. However, if the decomposition is used for *multiresolution modeling*, i.e., if the position of the control points is changed then a “more geometric” definition of the detail information is necessary.

A typical multiresolution modeling step consists of three stages. First the original geometry is decomposed into global (low frequency) shape and detail information. Then the global shape is modified and finally the detail information is added back by the reconstruction operator. If the detail information is vector valued then the reconstruction will often lead to counter intuitive results since the rotation of the global shape’s tangent is neglected (cf. Fig. 14.2).

In order to avoid this effect, Forsey and Bartels [9,10] introduced the notion of *local frame representation* of the detail coefficients: Instead of storing the detail vectors \mathbf{d}_i with respect to some global reference frame \mathbf{F} , one rather stores $\mathbf{d}'_i = \mathbf{F}_i^{-1}(\mathbf{F}(\mathbf{d}_i))$ with individual frames \mathbf{F}_i which depend on the local surface geometry of the low frequency geometry. For example, a local frame that consists of the normal vector and the tangent(s) automatically stays aligned to the underlying curve or surface.

After the modification, the low frequency geometry has changed and hence we find new local frames $\hat{\mathbf{F}}_i$. The detail coefficients which are used for the reconstruction are then $\hat{\mathbf{F}}_i(\mathbf{d}'_i)$ which guarantees intuitive detail preservation (cf. Fig. 14.2).

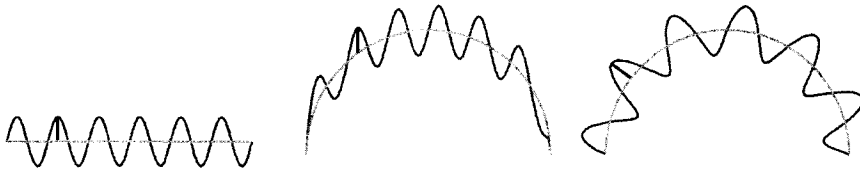


Figure 14.2. If the detail vectors are defined with respect to a global coordinate frame then the reconstruction after a deformation of the global shape (gray line) does lead to artifacts (center). A more intuitive detail preservation is achieved, if the detail is defined with respect to local frames that stay aligned to the global shape.

14.5. MULTIREOLUTION REPRESENTATIONS FOR SURFACES

The formal description of the multiresolution decomposition and reconstruction so far heavily relies on the regular structure of the control polygon or control mesh. In a regular mesh, all vertices can be labeled by a unique pair of indices, $\mathbf{c}_{i,j}$, and the bivariate subdivision operator computes the new control vertices by

$$\mathbf{c}'_{i,j} = \sum_{k,l} \alpha_{i-2k,j-2l} \mathbf{c}_{k,l}$$

which combines four different averaging rules according to the parity of i and j . The corresponding super- and subsampling operations that are based on the parity of the global indices can only work if the topological neighborhood of each vertex is identical. It is well-known that this requires each inner vertex of a triangle mesh to have exactly six neighbors (or each inner vertex of a quad mesh to have exactly four neighbors). This restriction is too strong for practical modeling applications since the class of possible shapes only contains surfaces which are homeomorphic to (a part of) a torus.

In order to apply multiresolution techniques to more general classes of freeform surfaces we have to extend the concepts of Section 14.2 to irregular meshes and non-planar parameter domains, e.g., a closed genus zero surface can be represented as a regular map from the unit sphere into 3-space while there is no regular map from any planar domain.

The standard procedure for finding operators with multiresolution functionality on surfaces with arbitrary topology is to first generalize the sub- and supersampling operations and then define decomposition and reconstruction operators that have properties similar to the original transformations on regular meshes. Ideally the generalized operators coincide with the original ones on regular meshes.

14.5.1. Coarse-to-fine hierarchies

Based on a generalized two-scale-relation, subdivision schemes provide the means to reconstruct a smooth surface from a coarse control mesh with arbitrary topology. The idea is to extend the knot-insertion operation for splines to irregular control meshes. Starting with the initial control mesh \mathcal{M}_0 consisting of the control vertices $\mathbf{c}_1^0, \dots, \mathbf{c}_{n(0)}^0$, we compute refined control meshes \mathcal{M}_m with control vertices $\mathbf{c}_1^m, \dots, \mathbf{c}_{n(m)}^m$ on the m th refinement

level. The geometric location of each new control vertex \mathbf{c}_i^{m+1} is computed by weighted averaging of nearby vertices \mathbf{c}_j^m from the unrefined mesh (cf. Chapter 12). For triangle meshes we usually have two different types of averaging rules. One for the new vertices $\mathbf{c}_{n(m)+1}^{m+1}, \dots, \mathbf{c}_{n(m+1)}^{m+1}$ and one for updating the old vertices $\mathbf{c}_1^{m+1}, \dots, \mathbf{c}_{n(m)}^{m+1}$.

The subdivision operator can serve as the *reconstruction operator* in a multiresolution representation of a freeform surface. When applying the subdivision operator to a given control mesh \mathcal{M}_m , we obtain *predicted* locations for the control vertices \mathbf{c}_i^{m+1} on the next level. Detail coefficients \mathbf{d}_i^m act like translation vectors which move the vertices back to their original location, i.e., in the reconstruction they undo the prediction error [34]. For the non-functional, geometric setting those displacement vectors have to be encoded with respect to a local coordinate frame (cf. Section 14.4).

The multiresolution representation of a surface by subdivision plus displacement on every refinement level mimics most of the important properties that we saw in the previous sections. The displacement vectors \mathbf{d}_i^m are local detail coefficients where the length of the vector indicates the significance of the detail and the refinement level on which the displacement is applied indicates the frequency band to which it belongs (cf. Fig. 14.3).

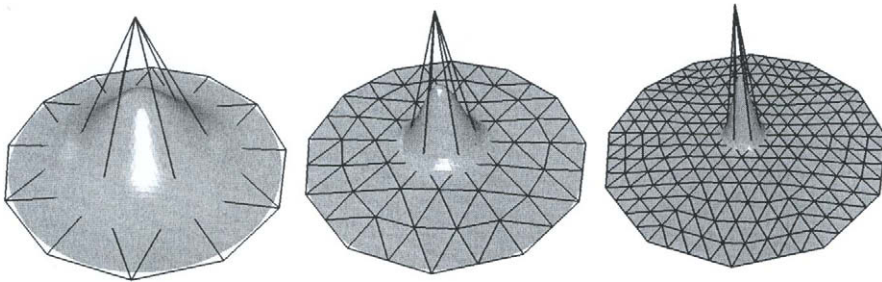


Figure 14.3. The effect of a detail coefficient \mathbf{d}_i^m depends on the associated refinement level. The region of the surface that is affected by one detail coefficient corresponds to the support of the basis function associated with the displaced control vertex. Since the support of the basis functions decreases with each refinement step, we obtain a proper decomposition of the geometric shape into different frequency bands.

So far we only considered the reconstruction operator which consists of subdivision plus displacement. For a complete multiresolution functionality we also have to construct a compatible *decomposition operator* that inverts the reconstruction. The easiest way to obtain such an operator is to apply subsampling to the original mesh \mathcal{M}_{m+1} then apply the subdivision scheme to obtain a mesh \mathcal{M}'_{m+1} which is a smoothed version of the original. The detail vectors are found by computing the shift of the vertices caused by this procedure.

Notice that in this case the number of detail coefficients equals the number, $n(m+1)$, of vertices in the fine mesh. This is quite different from the classical wavelet setting where

the number of detail coefficients, $n(m+1) - n(m)$, equals the number of vertices that are newly introduced by the reconstruction operator. In principle this does not affect the space and frequency localization properties of the decomposition but it introduces some redundancy since multiple detail coefficients are assigned to the same vertex on different refinement levels. This redundancy can be avoided if we use interpolatory subdivision (cf. Chapter 12).

A limitation of the subdivision based multiresolution representation is that it cannot be applied to arbitrary meshes. Because the decomposition is constructed for a prescribed type of reconstruction operator, the subsampling only applies to the special output of that operator. The specific connectivity of the meshes \mathcal{M}_m which are generated by the application of a uniform refinement operator is called *semi-regular* or *subdivision connectivity*. Semi-regular meshes consist of patches with regular mesh structure and extraordinary vertices (with valence $\neq 6$) only occur at the corners of these patches (cf. Fig. 14.4). Meshes to which we want to apply the “inverse subdivision” subsampling have to have this special connectivity.

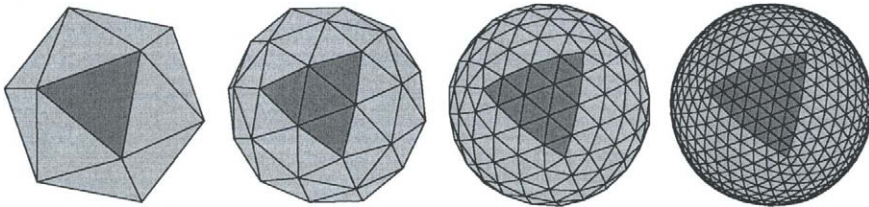


Figure 14.4. Although the base mesh \mathcal{M}_0 can be chosen arbitrarily in the coarse-to-fine setting, the refined resolutions \mathcal{M}_m must have subdivision connectivity. This is due to the uniform refinement of the reconstruction operator. The “inverse subdivision” cannot be applied to arbitrary meshes.

We call this type of hierarchy *coarse-to-fine hierarchy* since the structure of the meshes is determined by the coarsest level \mathcal{M}_0 (which can be arbitrary). All finer meshes \mathcal{M}_m are generated by iterative refinement and the decomposition operator only undoes this refinement. If a surface is given by an unstructured triangle mesh which does not have a semi-regular connectivity, remeshing techniques [8,19,21] have to be applied which resample the original surface to generate a semi-regular mesh that approximates the original geometry.

The mere displacement of individual control vertices after the refinement corresponds to the initial choice of the wavelet basis function in (14.10). Since the lifting scheme is not restricted to the uniform setting, we can apply it to improve the generalized reconstruction operator in a similar fashion. This leads to more involved reconstruction operators where a detail coefficient associated with one control vertex also affects the position of neighboring control vertices on the same level. Schröder and Sweldens use

this technique to design multiresolution decompositions for genus zero surfaces (parameterized over the unit sphere) [26]. Applying the lifting scheme also leads to improved decompositions with non-interpolatory basis functions but does not introduce redundant detail coefficients since the factorization of the transform can be computed in-place.

Another technique to improve the approximation behavior of the decomposition operator is presented in [23]. The construction starts with the nested sequence of spaces V_m which are spanned by the subdivision basis functions ϕ_i^m on the m th level (each function ϕ_i^m is associated with the corresponding control vertex \mathbf{c}_i^m). On the m th level, the basis $\phi_1^m, \dots, \phi_{n(m)}^m$ is extended to a basis of V_{m+1} by including the pre-wavelets $[\psi_{n(m)+1}^m, \dots, \psi_{n(m+1)}^m] := [\phi_{n(m)+1}^{m+1}, \dots, \phi_{n(m+1)}^{m+1}]$. Since the pre-wavelets are related to control vertices from the $(m + 1)$ st refinement level, they are associated with the *edges* of the mesh \mathcal{M}_m .

These pre-wavelets are then modified to make them orthogonal to the space V_m since this guarantees that the decomposition operator finds the best low frequency approximation in the least squares sense (*semi-orthogonal setting*). The orthogonalization is achieved by subtracting the least squares approximation σ_i^m of the function ψ_i^m from the space V_m , i.e.,

$$\psi_i^{m*} := \psi_i^m - \sigma_i^m = \psi_i^m - \sum_j s_{i,j} \phi_j^m.$$

It turns out that the least squares approximant σ_i^m happens to be globally supported in general which means that all $s_{i,j}$ are non-vanishing. Since this diminishes the efficiency of the decomposition and reconstruction operator, one tries to find a locally supported approximation of ψ_i^{m*} that is “as orthogonal as possible” to the space V_m . For this one prescribes a support Ω_i^m that is centered around the edge of \mathcal{M}_m where the vertex \mathbf{c}_i^{m+1} is going to be inserted. Then one finds the least squares approximation of ψ_i^m by using only those basis functions ϕ_i^m whose support lies within the interior of Ω_i^m . By increasing the size of Ω_i^m the resulting basis converges to the semi-orthogonal setting.

14.5.2. Fine-to-coarse hierarchies

The first attempt to generalize the concept of multiresolution representations to freeform surfaces worked from coarse to fine, i.e., we started with the reconstruction operator and the decomposition operator was derived by inverting the reconstruction. The second approach works the opposite way: We start with a decomposition operator building the hierarchy from *fine to coarse* and then derive the matching reconstruction operator.

The advantage of the fine-to-coarse approach is that it can be applied to arbitrary meshes, no special connectivity is required. The disadvantage is that we no longer have a simple representation of the surface geometry by a superposition of smooth basis functions (as they emerge from subdivision schemes) since the different hierarchy levels are non-nested and hence a proper two-scale-relation cannot be defined.

Given an arbitrary triangle mesh with vertices $\mathbf{c}_1^m, \dots, \mathbf{c}_{n(m)}^m$, we reduce its complexity by applying *mesh decimation algorithms*, e.g. [2,11,14,17,22,25]. Such algorithms remove vertices from the mesh according to some application specific criterion. A typical example for incremental decimation is edge collapsing where one vertex is removed at a time by shifting it into its neighbor’s position and eliminating degenerate triangles (cf. Fig. 14.5).

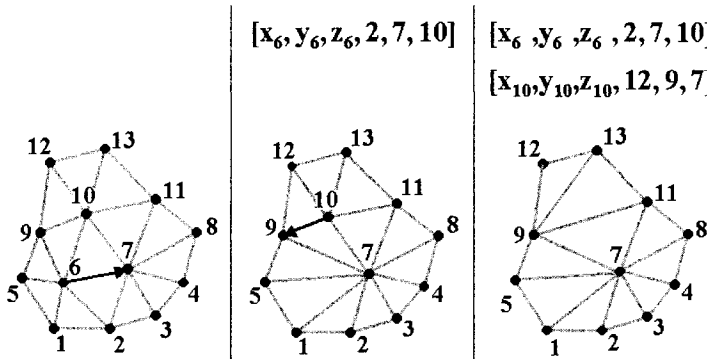


Figure 14.5. The edge collapse operation reduces the mesh complexity by one vertex and two triangles. Its inverse, the vertex split, can easily be performed if the local neighborhood relations are stored.

This operation reduces the mesh complexity by one vertex and two triangles and can be considered as a basic subsampling step. If we use edge collapsing to remove an independent set of vertices $\mathbf{c}_{n(m-1)+1}^m, \dots, \mathbf{c}_{n(m)}^m$, i.e., a set of vertices which are not connected by an edge then we achieve a global subsampling which does not require any regularity of the mesh connectivity (cf. Fig. 14.6).

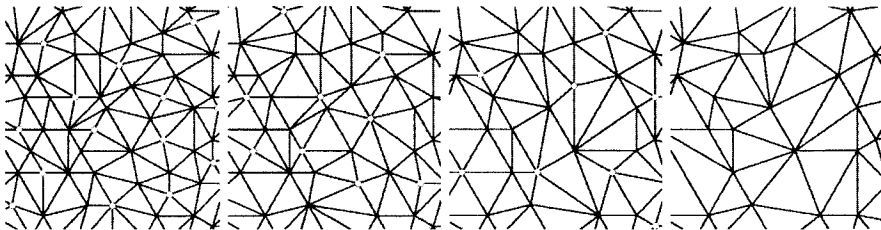


Figure 14.6. Removing an independent set of vertices (hollow dots) from the mesh \mathcal{M}_m has the effect of global subsampling but without requiring a regular structure of the mesh connectivity.

The original position of the removed vertices $\mathbf{c}_{n(m-1)+1}^m, \dots, \mathbf{c}_{n(m)}^m$ represents the detail information that is separated from the global shape by the decomposition operation. There are various ways to encode those positions relative to local frames which are aligned to the remaining geometry [20]. The decomposition does not produce redundancy since the number of geometric coefficients (one chunk per vertex) remains constant.

Hoppe [14] first observed that the edge collapsing can easily be inverted by vertex split operations. For this we only have to store little extra information about the local connectivity. Hence we immediately find a reconstruction operator which undoes the decomposition. Hoppe used this technique for the progressive transmission of complex meshes by first sending the decimated base mesh to the receiver and then sending a sequence of vertex splits which allow the client to run the mesh decimation backwards until the original model is recovered.

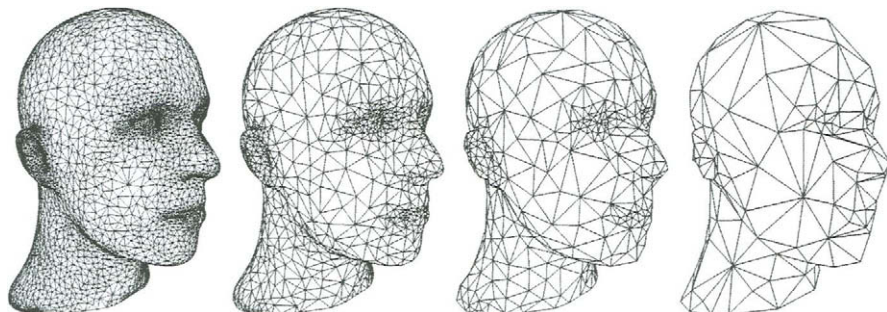


Figure 14.7. A sequence of meshes generated by a mesh decimation algorithm. We can go from left to right by performing edge collapses and we can go from right to left by splitting vertices (*progressive meshes*).

In the context of multiresolution techniques the combination of mesh decimation and progressive refinement yields the necessary pair of basic operators to switch between levels \mathcal{M}_m in a multiresolution representation for arbitrary meshes (cf. Fig. 14.7). However, so far we cannot access the smooth low frequency part of the geometry because we cannot refine the mesh without adding back the detail coefficients. For the full multiresolution functionality we have to be able to refine the mesh while suppressing the detail information since otherwise a geometric modification on a coarse scale will not lead to a smooth global deformation of the surface (cf. Fig. 14.8) [18].

In the coarse-to-fine setting of the last section the reconstruction without detail is achieved by simply applying the plain subdivision operator without displacement. In the fine to coarse setting, however, we no longer have such smooth basis functions associated with the vertices. As a consequence we have to find a more general prediction scheme that computes the expected position of the new vertices when they are introduced by the supersampling.

The most promising approaches to generate smooth geometry with unstructured triangle meshes are curvature flow techniques [6,12,32] and constrained optimization [16,18,24]. Both approaches lead to similar filter algorithms where every vertex of the mesh is shifted to a new position that is computed by a weighted average of its neighbors. The specific weights for these filters are derived from a discrete approximation of some continuous



Figure 14.8. A deformation of the global shape is not only characterized by a large support (topologically coarse) but also by its smoothness (low geometric frequency). The object on the left is globally deformed by using a (non-smooth) piecewise linear function in the center and a smooth function on the right.

curvature measure and depend on the local connectivity and edge lengths [7].

Based on these techniques we can define the reconstruction operator as follows: Re-insert a subsequence of previously removed vertices $\mathbf{c}_{n(m-1)+1}^m, \dots, \mathbf{c}_{n(m)}^m$ by vertex splits in reverse order. Determine the predicted position of the new vertices by applying a filter operation. Move the vertices to their final position by adding the detail vectors \mathbf{d}_i^n

Omitting the last step leads to topologically refined meshes without high frequency detail (cf. Fig. 14.9). If the smoothing filter is applied to the position of the coarse-scale vertices $\mathbf{c}_1^m, \dots, \mathbf{c}_{n(m-1)}^m$ as well, we can improve the quality of the low frequency geometry but we have to store redundant detail coefficients.

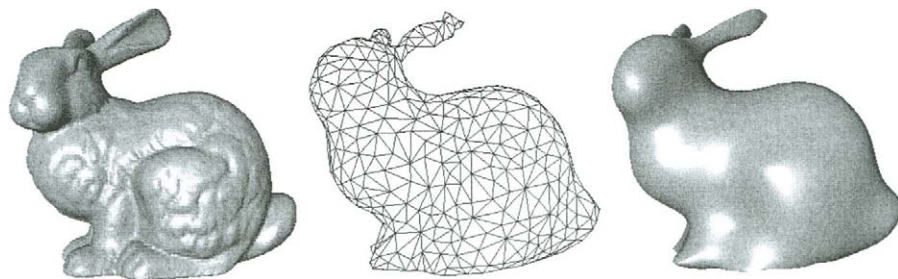


Figure 14.9. Starting with the original mesh \mathcal{M}_m on the left, we apply mesh decimation to build up a fine-to-coarse hierarchy. The coarsest level \mathcal{M}_0 is shown in the center. When re-inserting the vertices, we can suppress the detail information by computing new (predicted) vertex positions with some smoothing filter. The resulting mesh \mathcal{M}'_m on the right has the same connectivity as the original mesh \mathcal{M}_m but the geometry does not contain any high frequency details.

14.6. APPLICATIONS

Multiresolution representations of geometric models decompose the overall shape into detail information from different scales or frequency bands. These representations are hierarchical in the sense that besides the “horizontal” ordering of the geometric coefficients according to the surface topology (or mesh connectivity) we obtain a “vertical” ordering of the coefficients according to significance or feature size. This vertical ordering is very intuitive since it enables to directly access the shape of an object on various levels of detail.

There are plenty of applications where the augmented structure of hierarchical representations is used to increase the performance by allowing algorithms to focus the processing resources on the significant part of the geometry or by adapting the amount of detail information to the required accuracy. Typical examples for this type of applications are data compression and progressive transmission.

Another class of applications does not rate the detail coefficients according to their significance but tries to exploit the *semantic* information that emerges from the decomposition. Defining the detail information relative to the global shape is what designers usually do when assembling complex CAD objects. The rationale behind this is that local features are (semantically and physically) attached to the main body of an object and if the main body’s shape is altered then the local features should follow accordingly. The decomposition operator in a multiresolution scheme automatically recovers this type of hierarchical structure from the final shape. While the decomposition cannot identify the functional parts in a CAD model, it can at least distinguish between different feature sizes. Various metaphors for multiresolution editing can be implemented based on this structure.

14.6.1. Multiresolution editing

The general procedure for multiresolution editing is a three step process. First the decomposition operator is applied to separate detail information and global shape. Then the global shape is modified and finally the detail information is added back by the reconstruction operator. The detail reconstruction will be intuitive if the vector valued detail coefficients are encoded with respect to local frames (cf. Section 14.4).

For coarse to fine hierarchies, the multiresolution editing is quite simple since we have well-defined subdivision basis functions associated with each control vertex on each refinement level. Early works by Forsey and Bartels [9,10] already used these technique for hierarchical spline surfaces and Zorin et al. [34] generalized it to subdivision surfaces.

An interesting difference between the two approaches is that for hierarchical splines the control vertices on the different hierarchy levels are considered completely independent while Zorin et al. propagate modifications on the fine levels down to the coarser ones. The goal of this propagation is to keep the detail coefficients on each level as small as possible. Although this makes the reconstruction operator numerically more stable, the strict separation of the detail levels is not preserved.

In [12] Guskov et al. propose a technique for multiresolution decomposition of arbitrary meshes. The decomposition is based on a mesh decimation technique and the reconstruction operator uses a sophisticated smoothing filter for the prediction, i.e., each vertex split

during the reconstruction is followed by applying a low-pass filter to a local vicinity. The resulting multiresolution representation is highly redundant since for each vertex split one has to store detail coefficients for all neighboring vertices that are affected by the local smoothing.

One drawback of the above mentioned approaches is that the basis functions which are associated with the mesh vertices are fixed and their definition cannot be adapted to a specific design goal. Therefore in [18] a multiresolution metaphor is presented that builds up the hierarchy on demand. The designer can choose the location and the support of a modification and a fine to coarse hierarchy is then built by applying mesh decimation to the region of the mesh that is covered by the support. Because the resulting decomposition is custom made for one specific editing operation, one does not have to explicitly use the intermediate hierarchy levels but one can restrict to a two-level decomposition (cf. Fig. 14.10).

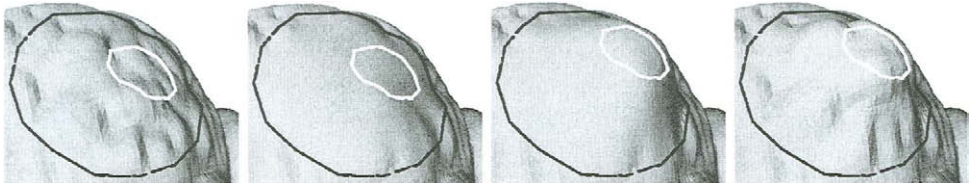


Figure 14.10. A flexible metaphor for multiresolution edits. On the left, the original mesh is shown. The black line defines the region of the mesh which is subject to the modification. The white line defines the handle geometry which can be moved by the designer. Both boundaries can have an arbitrary shape and hence they can, e.g., be aligned to geometric features in the mesh. The boundary and the handle impose C^1 and C^0 boundary conditions to the mesh and the smooth version of the original mesh is found by applying discrete fairing while observing these boundary constraints. The center left shows the result of the curvature minimization (the boundary and the handle are interpolated). The geometric difference between the two left meshes is stored as detail information with respect to local frames. Now the designer can move the handle polygon and this changes the boundary constraints for the curvature minimization. Hence the discrete fairing generates a modified smooth mesh (center right). Adding the previously stored detail information yields the final result on the right. Since we can apply fast multi-level smoothing when solving the optimization problem, the modified mesh can be updated with several frames per second during the modeling operation. Notice that all four meshes have the same connectivity.

14.6.2. Geometry compression

Techniques for lossy compression of geometry data often exploit the fact that multiresolution decompositions imply an ordering of the detail coefficients according to their

significance. If we want to obtain a prescribed compression ratio we can simply remove a certain percentage of the detail coefficients starting with the least significant ones. If we want to stay within a prescribed approximation tolerance, we can remove the least significant detail coefficients as long as we do not violate that tolerance.

Sophisticated multiresolution representations improve the effectiveness of such algorithms. If we choose the right basis functions ϕ and ψ then the approximation quality of the low frequency component increases and hence the prediction error (= detail coefficients) during the reconstruction becomes smaller. This is the reason why one usually aims at the (approximate) semi-orthogonal setting.

In [23] Lounsbery et al. constructed piecewise linear wavelet functions such that they are, for a given support, as orthogonal as possible to the space of subdivision basis functions. Based on the lifting scheme, Schröder and Sweldens constructed wavelets with vanishing moments for various subdivision schemes and compared their approximation properties [26].

Guskov et al. [13] and Khodakhowski et al. [15] additionally exploit the geometric coherence of a meshed surface by storing the detail coefficients as scalar valued normal displacements instead of vector valued local frame displacements. They achieve this by resampling the original geometry such that the tangential component of the displacement vectors vanishes.

All the above multiresolution compression schemes are based on coarse to fine hierarchies. The reason for this is that the availability of subdivision basis functions and their corresponding wavelets allows to adapt the theoretical concepts from the regular functional setting.

In [3] Cohen-Or et al. propose a compression scheme which is based on a fine to coarse hierarchy. Their technique combines ideas from lossless non-hierarchical mesh compression with progressive reconstruction of fine to coarse hierarchies. In every subsampling step they remove an independent set of vertices and retriangulate the resulting holes by triangle strips. In order to keep the detail vectors as small as possible they use a linear prediction scheme that is similar to the low pass filter operations mentioned in Section 14.5.2.

Taubin describes a progressive compression scheme in [33]. Here the fine to coarse hierarchy is generated by rather complex “forest splits” which are a generalization of the vertex split operation. The scheme is optimized for connectivity compression.

REFERENCES

1. C. Chui. *An Introduction to Wavelets*, Academic Press, San Diego, 1996.
2. P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, pages 37–54, 1998.
3. D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. *IEEE Visualization Conference Proceedings*, pages 67–72, 1999.
4. I. Daubechies. *Ten Lectures on Wavelets*, CBMS-NSF Regional Conf. Series in Appl. Math., Vol. 61. SIAM, Philadelphia, PA, 1992.
5. I. Daubechies and W. Sweldens. Factoring Wavelet Transforms into Lifting Steps. *J. Fourier Anal. Appl.*, pages 245–267, 1998.
6. M. Desbrun, M. Meyer, P. Schröder, and A. Barr. Implicit fairing of irregular meshes

- using diffusion, curvature flow. *Computer Graphics (SIGGRAPH 99 Proceedings)*, pages 317–324, 1999.
7. M. Desbrun, M. Meyer, P. Schröder, and A. Barr. *Discrete Differential-Geometry Operators in nD* , Preprint.
 8. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. *Computer Graphics (SIGGRAPH 95 Proceedings)*, pages 173–182, 1995.
 9. D. Forsey and R. Bartels. Hierarchical B-spline refinement. *Computer Graphics (SIGGRAPH 88 Proceedings)*, pages 205–212, 1988.
 10. D. Forsey and R. Bartels. Surface fitting with hierarchical splines. *ACM Transactions on Graphics*, pages 134–161, 1995.
 11. M. Garland and P. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics (SIGGRAPH 97 Proceedings)*, pages 209–216, 1997.
 12. I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. *Computer Graphics (SIGGRAPH 99 Proceedings)*, pages 325–334, 1999.
 13. I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. *Computer Graphics (SIGGRAPH 2000 Proceedings)*, pages 95–102, 2000.
 14. H. Hoppe. Progressive meshes. *Computer Graphics (SIGGRAPH 96 Proceedings)*, pages 99–108, 1996.
 15. A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression, *Computer Graphics (SIGGRAPH 2000 Proceedings)*, pages 271–278, 2000.
 16. L. Kobbelt. Discrete fairing, *Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces*, pages 101–131. Information Geometers, Winchester, 1997.
 17. L. Kobbelt, S. Campagna, and H-P. Seidel. A general framework for mesh decimation, *Graphics Interface Proceedings*, pages 43–50, 1998.
 18. L. Kobbelt, S. Campagna, J. Vorsatz, and H.P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. *Computer Graphics (SIGGRAPH 98 Proceedings)*, pages 105–114, 1998.
 19. L. Kobbelt, J. Vorsatz, U. Labsik, and H-P. Seidel. A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum 18 (1999)*, Eurographics '99 issue, C119–C130.
 20. L. Kobbelt, J. Vorsatz, and H-P. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry: Theory and Applications*, 14:5–24, 1999.
 21. A. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: multiresolution adaptive parameterization of surfaces. *Computer Graphics (SIGGRAPH 98 Proceedings)*, pages 95–104, 1998.
 22. P. Lindstrom and G. Turk. Fast, memory efficient polygonal simplification. *IEEE Visualization '98 Conference Proceedings*, pages 279–286, 1998.
 23. M. Lounsbery, T. DeRose, and J. Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, 1997.
 24. R. Schneider and L. Kobbelt. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design*, 18(4): 359–379, 2001.
 25. W. Schroeder, J. Zarge, and W. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH 92 Proceedings)*, pages 65–70, 1992.
 26. P. Schröder and W. Sweldens. Spherical wavelets: efficiently representing functions on

- the sphere. *Computer Graphics (SIGGRAPH 95 Proceedings)*, pages 161–172, 1995.
27. E. Stollnitz, T. DeRose, and D. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers Inc., San Francisco, 1996.
 28. G. Strang and T. Nguyen. *Wavelets, Filter Banks*. Wellesley Cambridge Press, 1996.
 29. W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Appl. Comput. Harmon. Anal.*, pages 186–200, 1996.
 30. W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, pages 511–546, 1997.
 31. W. Sweldens and P. Schröder. Building your own wavelets at home. *Wavelets in Computer Graphics*, ACM SIGGRAPH Course notes, pages 15–87, 1996.
 32. G. Taubin. A signal processing approach to fair surface design. *Computer Graphics (SIGGRAPH 95 Proceedings)*, pages 351–358, 1995.
 33. G. Taubin, A. Gueziec, W. Horn, and F. Lazarus. Progressive forest split compression. *Computer Graphics (SIGGRAPH 98 Proceedings)*, pages 123–132, 1998.
 34. D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. *Computer Graphics (SIGGRAPH 97 Proceedings)*, pages 259–268, 1997.

Chapter 15

Algebraic Methods for Computer Aided Geometric Design

Thomas W. Sederberg and Jianmin Zheng

The concepts and methods of algebra and algebraic geometry have found significant applications in many disciplines. This chapter presents a collection of gleanings from algebra or algebraic geometry that hold practical value for the field of computer aided geometric design. We focus on the insights, algorithm enhancements and practical capabilities that algebraic methods have contributed to CAGD. Specifically, we examine resultants and Gröbner basis, and discuss their applications in implicitization, inversion, parametrization and intersection algorithms. Other topics of CAGD research work using algebraic methods are also outlined.

15.1. INTRODUCTION

CAGD draws from several branches of mathematics and computer science, such as approximation theory, differential geometry, and numerical analysis. This chapter reviews some of the tools of algebra and algebraic geometry that have been brought to bear on problems in CAGD [11,17,27,28,33,37,45].

Most of the free-form curves and surfaces used in CAGD are given by parametric equations. Planar rational curves in CAGD are typically defined as

$$x = \frac{a(t)}{c(t)}, \quad y = \frac{b(t)}{c(t)} \tag{15.1}$$

where $a(t)$, $b(t)$, and $c(t)$ are polynomials in the Bernstein basis for rational Bézier curves or in the B-spline basis for NURBS. Algebraic methods most commonly use polynomials in the power basis: $a(t) = a_0 + a_1t + \cdots + a_n t^n$, etc. Polynomials can be converted from Bernstein basis to power basis, although some algebraic methods such as resultants can be formulated using the Bernstein basis directly [26].

Parametric surfaces in CAGD are defined

$$x = \frac{a(s, t)}{d(s, t)}, \quad y = \frac{b(s, t)}{d(s, t)}, \quad z = \frac{c(s, t)}{d(s, t)} \quad (15.2)$$

where $a(s, t)$, $b(s, t)$, $c(s, t)$ and $d(s, t)$ are polynomials.

Surfaces and plane curves can also be defined using implicit equations. One contribution that algebraic methods make to CAGD is in solving the problem of implicitization and inversion of parametric curves and surfaces. For any parametric curve given by (15.1), an implicit equation $f(x, y) = 0$ (where $f(x, y)$ is a polynomial) exists that describes exactly the same curve. Likewise, for any parametric surface given by (15.2), there exists an implicit equation $f(x, y, z) = 0$ that describes exactly the same surface. The process of finding the implicit equation of a parametric curve or surface is called *implicitization*. Implicitization is of value in CAGD because the problem of determining whether a given point lies on a curve or surface is addressed much more easily using the implicit form than the parametric form. Curve implicitization is discussed in Section 15.4.

An *inversion* formula for a parametric curve (15.1) is of the form $t = \frac{g(x, y)}{h(x, y)}$ where g and h are polynomials. If the parametrization of a curve is a generally one-to-one map between parameter values and points on the curve, the inversion formula returns the parameter value t corresponding to a point (x, y) that lies on the curve. Inversion is discussed also in Section 15.4.

The process of finding the rational parametric equations of implicitly defined algebraic curves and surfaces is called *parametrization*. Some methods for parametrizing plane algebraic curves are shown in Section 15.5.

Algebraic methods also can facilitate the design of algorithms for computing intersections between curves and surfaces. The curve intersection problem is surveyed in Section 15.6.

The problems of implicitization, parametrization and intersection for surfaces are discussed in Section 15.7. Some other important applications of algebraic methods to CAGD are listed in Section 15.8.

Many of the algebraic methods reviewed in this chapter come from classical analytic geometry [31,32,46,48]. The twentieth century witnessed a marked shift from the constructive approach to non-constructive [5]. Section 15.2 gives a brief overview of polynomial ideals, varieties and Gröbner bases, and Section 15.3 introduces three popular resultant formulations.

15.2. POLYNOMIALS, IDEALS, AND VARIETIES

This section first introduces the notation and terminology which will be used later, and then presents the fundamental concepts of ideals and varieties and suggests some ways how these topics fit into CAGD. An excellent treatment of ideals and varieties and their application to CAGD can be found in [17].

15.2.1. Notation and terminology

In general, a polynomial in n variables x_1, \dots, x_n is defined

$$f(x_1, x_2, \dots, x_n) = \sum_{i=1}^{\tau} c_i x_1^{e_{1,i}} x_2^{e_{2,i}} \dots x_n^{e_{n,i}}. \quad (15.3)$$

Each summand $c_i x_1^{e_{1,i}} x_2^{e_{2,i}} \cdots x_n^{e_{n,i}}$ is called a *term*, $x_1^{e_{1,i}} x_2^{e_{2,i}} \cdots x_n^{e_{n,i}}$ is a *monomial*, and c_i is the *coefficient* of the monomial. By convention, any given monomial occurs in at most one term in a polynomial.

$k[x_1, \dots, x_n]$ signifies the set of all polynomials in the variables x_1, \dots, x_n whose coefficients belong to a field k . For example, $R[x, y]$ is the set of all polynomials

$$\sum c_i x^{e_{1,i}} y^{e_{2,i}} \quad (15.4)$$

where $c_i \in R$ and $e_{1,i}, e_{2,i} \in \{0, 1, 2, \dots\}$. Thus, " $f \in R[x, y, z]$ " means that f is a polynomial whose variables are x, y and z and whose coefficients are real numbers. All polynomials in this chapter have coefficients that are real numbers.

It is often useful to list the terms of a polynomial in decreasing order, beginning with the *leading term*. This is done using a *term order* — a way to compare any two distinct terms of a polynomial and declare which is "greater".

For linear polynomials, term order amounts to merely declaring an order on the variables. For example, the terms of the polynomial

$$2x + 3y - 4z$$

are in proper order if we declare $x > y > z$. If we declare $y > z > x$, the proper order would be $3y - 4z + 2x$. For non-linear polynomials, we begin by declaring an order on the variables, and then we must also choose one of several schemes that decide how the exponents in a polynomial influence term order. One such scheme is called *lexicographical order* (nicknamed *lex*), defined as follows. If the variables of a polynomial are ordered $x_1 > x_2 > \dots > x_n$, then given two distinct terms $T_i = c_i x_1^{e_{1,i}} x_2^{e_{2,i}} \cdots x_n^{e_{n,i}}$ and $T_j = c_j x_1^{e_{1,j}} x_2^{e_{2,j}} \cdots x_n^{e_{n,j}}$, $T_i > T_j$ if

1. $e_{1,i} > e_{1,j}$, or if
2. $e_{1,i} = e_{1,j}$ and $e_{2,i} > e_{2,j}$, or, in general, if
3. $e_{k,i} = e_{k,j}$ for $k = 1, \dots, m - 1$ and $e_{m,i} > e_{m,j}$.

For example, the polynomial

$$3x^2y^2z + 4xy^3z^2 + 5x^3z + 6y^2 + 7xz^3 + 8$$

using *lex* with $x > y > z$ would be written $5x^3z + 3x^2y^2z + 4xy^3z^2 + 7xz^3 + 6y^2 + 8$ and its leading term is $5x^3z$. Using *lex* with $z > x > y$ it would be written $7z^3x + 4z^2xy^3 + 5zx^3 + 3zx^2y^2 + 6y^2 + 8$ and the leading term would be $7z^3x$. Or using *lex* with $y > z > x$ it would be written $4y^3z^2x + 3y^2zx^2 + 6y^2 + 7z^3x + 5zx^3 + 8$ and the leading term would be $4y^3z^2x$.

Another choice for term order is the *degree lexicographical order* (abbreviated *deglex*). If the variables are ordered $x_1 > x_2 > \dots > x_n$, then using *deglex*, $T_i > T_j$ if

1. $e_{1,i} + e_{2,i} + \dots + e_{n,i} > e_{1,j} + e_{2,j} + \dots + e_{n,j}$, or
2. $e_{1,i} + e_{2,i} + \dots + e_{n,i} = e_{1,j} + e_{2,j} + \dots + e_{n,j}$ and $T_i > T_j$ with respect to *lex*.

Using deglex with $x > y > z$, the terms of $3x^2y^2z + 4xy^3z^2 + 5x^3z + 6y^2 + 7xz^3 + 8$ would be ordered $4xy^3z^2 + 3x^2y^2z + 5x^3z + 7xz^3 + 6y^2 + 8$.

As observed in the lex and deglex examples, term orders ignore the coefficient of a term, so a term order might more properly be called a monomial order.

Other term orders can also be defined, such as degree reverse lexicographical order. The precise requirements for any term order are discussed in reference [6], page 18.

The n -dimensional real affine space is denoted R^n and is the set of n -tuples:

$$R^n = \{(a_1, \dots, a_n) : a_1, \dots, a_n \in R\} \quad (15.5)$$

15.2.2. Ideals and varieties

The *polynomial ideal* generated by $f_1, \dots, f_s \in k[x_1, \dots, x_n]$, denoted $\langle f_1, \dots, f_s \rangle$, is defined

$$\langle f_1, \dots, f_s \rangle = \{p_1f_1 + \dots + p_sf_s : p_i \in k[x_1, \dots, x_n]\}.$$

The polynomials f_1, \dots, f_s are called the *generators* of this ideal.

Consider a set of polynomials $f_1, f_2, \dots, f_s \in k[x_1, \dots, x_n]$. Let (a_1, \dots, a_n) be a point in k^n satisfying $f_i(a_1, \dots, a_n) = 0$, $i = 1, \dots, s$. The set of all such points (a_1, \dots, a_n) is called the *variety* defined by f_1, \dots, f_s , and is denoted by $V(f_1, \dots, f_s)$:

$$V(f_1, \dots, f_s) = \{(a_1, \dots, a_n) \in k^n \mid f_i(a_1, \dots, a_n) = 0, i = 1, \dots, s\}. \quad (15.6)$$

A variety defined by a single polynomial—called a *hypersurface*—is the most familiar instance of a variety. A hypersurface in R^2 is a planar curve defined using an implicit equation, and a hypersurface in R^3 is what is normally called an implicit surface in CAGD. For example, $V(x^2 + y^2 - 1)$ is a circle defined in terms of the implicit equation $x^2 + y^2 - 1 = 0$ and $V(2x + 4y - z + 1)$ is the plane whose implicit equation is $2x + 4y - z + 1 = 0$.

A variety $V(f_1, \dots, f_s)$ defined by more than one polynomial ($s > 1$) is the intersection of the varieties $V(f_1) \dots V(f_s)$.

15.2.3. Gröbner bases

It can be very useful to devise alternative generators for an ideal. Necessary and sufficient conditions for $\langle f_1, \dots, f_n \rangle = \langle g_1, \dots, g_m \rangle$ are $f_1, \dots, f_n \in \{g_1, \dots, g_m\}$ and $g_1, \dots, g_m \in \{f_1, \dots, f_n\}$.

A *Gröbner basis* of an ideal I is a set of polynomials $\{g_1, \dots, g_t\}$ such that the leading term of any polynomial in I is divisible by the leading term of at least one of the polynomials g_1, \dots, g_t . This, of course, requires that a term order be fixed for determining the leading terms: different term orders produce different Gröbner bases. Several excellent books have been written on Gröbner bases that do not presuppose that the reader has advanced training in mathematics [6,10,17]

A Gröbner basis is a particularly attractive set of generators for an ideal, as illustrated by two familiar examples. If $\{f_1, \dots, f_s\}$ are polynomials in one variable, the Gröbner basis of $\langle f_1, \dots, f_n \rangle$ consists of a single polynomial: the greatest common divisor (GCD) of f_1, \dots, f_s . If $\{f_1, \dots, f_s\}$ are linear polynomials in several variables, the Gröbner basis is an uppertriangular form of a set of linear equations. The Gröbner basis of these special cases provides significant computational advantage and greater insight, and the same is true of the Gröbner basis of a more general ideal.

undetermined. Letting $h(t)f(t) - k(t)g(t) = 0$ leads to $m + n$ linear equations with $m + n$ unknowns which are the coefficients of $h(t)$ and $k(t)$. The determinant of the coefficient matrix of these $m + n$ linear equations is exactly Sylvester's resultant. Obviously the determinant vanishes if and only if there exist nonzero polynomials $h(t)$ of degree not greater than $m - 1$ and $k(t)$ of degree not greater than $n - 1$ such that $h(t)f(t) = k(t)g(t)$ holds. This is equivalent to the existence of the common roots of polynomials $f(t)$ and $g(t)$. Therefore Euler's method shows that $R(f, g) = 0$ is not only the necessary but also sufficient condition for $f(t)$ and $g(t)$ to have common roots.

15.3.2. Bezout's resultant

Another popular resultant formulation for two univariate polynomials is Bezout's resultant. A nice derivation of Bezout's resultant is due to Cayley. Without loss of generality, we assume the degree of the polynomial $g(t)$ is less than the degree of $f(t)$, i.e., $m \leq n$. Construct a symmetric function

$$\delta(t, s) = \begin{vmatrix} f(t) & f(s) \\ g(t) & g(s) \end{vmatrix} / (s - t) = \frac{f(t)g(s) - f(s)g(t)}{s - t}.$$

Some algebraic manipulation shows that $\delta(t, s)$ is a degree $n - 1$ polynomial in s whose coefficients are polynomials of t :

$$\begin{aligned} \delta(t, s) &= f(t)(g(t) - g(s))/(t - s) - g(t)(f(t) - f(s))/(t - s) \\ &= \sum_{k=0}^{m-1} \left(f(t) \sum_{i=k+1}^m b_i t^{i-k-1} - g(t) \sum_{i=k+1}^n a_i t^{i-k-1} \right) s^k - \sum_{k=m}^{n-1} \left(g(t) \sum_{i=k+1}^n a_i t^{i-k-1} \right) s^k. \end{aligned}$$

The variety of the ideal generated by these n polynomials is the same as $V(\langle f(t), g(t) \rangle)$. Write these polynomials in matrix form:

$$\begin{bmatrix} f \sum_{i=1}^m b_i t^{i-1} - g \sum_{i=1}^n a_i t^{i-1} \\ \vdots \\ fb_m - g \sum_{i=m}^n a_i t^{i-m} \\ -g \sum_{i=m+1}^n a_i t^{i-m-1} \\ \vdots \\ -ga_n \end{bmatrix} = \begin{bmatrix} c_{00} & \cdots & c_{0,n-1} \\ \vdots & \ddots & \vdots \\ c_{n-1,0} & \cdots & c_{n-1,n-1} \end{bmatrix} \begin{bmatrix} 1 \\ t \\ \vdots \\ t^{n-1} \end{bmatrix} \tag{15.7}$$

with the entry $c_{ij} = \sum_{\substack{k \leq \min(i,j) \\ k+h=i+j+1}} (a_k b_h - a_h b_k)$ and the convention that $b_{m+1} = \cdots = b_n = 0$.

This coefficient matrix is called Bezout's matrix. If $V(\langle f, g \rangle)$ is non-empty, the determinant of Bezout's matrix must vanish. The converse is also true when $n = m$, the proof of which can be found in [22,26]. The determinant is therefore a resultant for f and g , known as Bezout's resultant. In general, Bezout's resultant has dimension $n \times n$ while Sylvester's resultant has dimension $(n + m) \times (n + m)$.

When $n > m$, Bezout's determinant has an extraneous factor of a_n^{n-m} . This extraneous factor can be removed by modifying Bezout's resultant as follows [19]: The first m

polynomials are the same as in (15.7); and the remaining $n - m$ polynomials are obtained from $t^{i-m}g(t)$, $i = m, \dots, n$. Thus $c_{ij} = b_{i+j-m}$ for $i \geq m, 0 \leq j \leq n - 1$. Look at the example of $f(t) = a_2t^2 + a_1t + a_0$ and $g(t) = b_1t + b_0$. The original Bezout's determinant is

$$\begin{vmatrix} a_0b_1 - a_1b_0 & -a_2b_0 \\ -a_2b_0 & -a_2b_1 \end{vmatrix} = a_2 \begin{vmatrix} a_0b_1 - a_1b_0 & -a_2b_0 \\ -b_0 & -b_1 \end{vmatrix}$$

and the variant of Bezout's resultant is $\begin{vmatrix} a_0b_1 - a_1b_0 & -a_2b_0 \\ b_0 & b_1 \end{vmatrix}$.

15.3.3. Dixon's resultant

Cayley's formulation can be extended to the case of three bivariate polynomials. Consider three polynomials:

$$f(s, t) = \sum_{i=0}^n \sum_{j=0}^m a_{ij}s^i t^j, \quad g(s, t) = \sum_{i=0}^n \sum_{j=0}^m b_{ij}s^i t^j, \quad h(s, t) = \sum_{i=0}^n \sum_{j=0}^m c_{ij}s^i t^j$$

Dixon observed that the expression

$$\delta(s, t, \alpha, \beta) = \begin{vmatrix} f(s, t) & g(s, t) & h(s, t) \\ f(s, \beta) & g(s, \beta) & h(s, \beta) \\ f(\alpha, \beta) & g(\alpha, \beta) & h(\alpha, \beta) \end{vmatrix} / (s - \alpha)(t - \beta)$$

is actually a polynomial of degree $2n - 1, m - 1, n - 1$ and $2m - 1$ in s, t, α, β respectively. Thus it can be written as $\delta(s, t, \alpha, \beta) = \sum_{ijkl} d_{ijkl}s^i t^j \alpha^k \beta^l$ where d_{ijkl} are expressions in a_{ij}, b_{ij} and c_{ij} .

For any $(s, t) \in V(\langle f, g, h \rangle)$, $\delta(s, t, \alpha, \beta)$ vanishes no matter what α and β are. Thus the coefficients of each $\alpha^k \beta^l$ must vanish at these (s, t) pairs. This gives $2mn$ polynomials, each of which has $2mn$ terms in s and t since s has degree $2n - 1$ and t is degree $m - 1$. The determinant of the coefficient matrix from these polynomials serves as a resultant for f, g and h , called Dixon's resultant [23].

Consider the example:

$$\begin{cases} f(t) = a_{21}s^2t + a_{11}st + a_{01}t + a_{20}s^2 + a_{10}s + a_{00} \\ g(t) = b_{21}s^2t + b_{11}st + b_{01}t + b_{20}s^2 + b_{10}s + b_{00} \\ h(t) = c_{21}s^2t + c_{11}st + c_{01}t + c_{20}s^2 + c_{10}s + c_{00} \end{cases}$$

Dixon's method gives

$$\delta(s, t, \alpha, \beta) = [1, \alpha, \beta, \alpha\beta] \begin{bmatrix} (00, 01, 10) & (00, 01, 20) & (00, 21, 10) & (00, 21, 20) \\ & +(00, 11, 10) & +(00, 11, 20) & \\ (00, 01, 11) & (00, 01, 21) & (01, 21, 10) & (01, 21, 20) \\ & +(01, 11, 10) & +(01, 11, 20) & \\ (00, 01, 20) & (10, 01, 20) & (00, 21, 20) & (10, 21, 20) \\ & +(00, 11, 20) & +(10, 11, 20) & \\ (00, 01, 21) & (00, 11, 21) & (01, 21, 20) & (11, 21, 20) \\ & +(10, 01, 21) & +(10, 11, 21) & \end{bmatrix} \begin{bmatrix} 1 \\ s \\ s^2 \\ s^3 \end{bmatrix}$$

where (ij, kj, pq) stands for the 3×3 determinant $(ij, kj, pq) = \begin{vmatrix} a_{ij} & a_{kl} & a_{pq} \\ b_{ij} & b_{kl} & b_{pq} \\ c_{ij} & c_{kl} & c_{pq} \end{vmatrix}$.

15.4. CURVE IMPLICITIZATION AND INVERSION

The algebraic tools of Gröbner bases and resultants empower us to solve several problems of interest to CAGD. This section looks at several examples of implicitization and inversion.

It is known from classical algebraic geometry that any degree n polynomial or rational parametric curve can be represented exactly using a degree n algebraic equation. For example, a circle can be expressed using the parametric equation

$$x = (1 - t^2)/(t^2 + 1), \quad y = 2t/(t^2 + 1) \tag{15.8}$$

or using the implicit equation $x^2 + y^2 - 1 = 0$. In the following we discuss three approaches for the conversion from the parametric equation to the implicit equation.

15.4.1. Resultant-based method

We have presented the resultant tool for determining whether two polynomials have a common root. We now apply that tool to converting the parametric equation of a curve given by (15.1) into an implicit equation of the form $f(x, y) = 0$.

We proceed by forming two auxiliary polynomials:

$$g(x, t) = c(t)x - a(t), \quad h(y, t) = c(t)y - b(t)$$

View $g(x, t)$ as a polynomial in t whose coefficients are linear in x , and view $h(y, t)$ as a polynomial in t whose coefficients are linear in y . If we compute the resultant of $g(x, t)$ and $h(y, t)$, we do not arrive at a numerical value, but rather a polynomial in x and y which we call $f(x, y)$. Note that $g(x, t) = h(y, t) = 0$ only for values of x, y and t which satisfy the relationships $x = a(t)/c(t), y = b(t)/c(t)$. Clearly, for these values of x, y and t , the resultant $f(x, y)$ must vanish. Conversely, any (x, y) pair for which $f(x, y) = 0$, causes the resultant of g and h to be zero. But, if the resultant is zero, then we know that there exists a value of t for which $g(x, t) = h(y, t) = 0$. In other words, all (x, y) for

which $f(x, y) = 0$ lie on the parametric curve and therefore $f(x, y) = 0$ is the implicit equation of that curve.

As discussed, the resultant is the determinant of a matrix of coefficients for a set of polynomials. Inversion—computing the parameter t for a point (x, y) known to lie on the curve—can be performed by solving such a set of polynomial equations using Gauss elimination or Cramer’s rule.

We illustrate with the circle parametrized by (15.8). We have

$$g = (x + 1)t^2 + (x - 1), \quad h = yt^2 - 2t + y$$

Using Sylvester’s resultant, we obtain a 4×4 determinant

$$f(x, y) = \begin{vmatrix} x + 1 & 0 & x - 1 & 0 \\ 0 & x + 1 & 0 & x - 1 \\ y & -2 & y & 0 \\ 0 & y & -2 & y \end{vmatrix} = 4(x^2 + y^2 - 1).$$

Bezout’s resultant provides a 2×2 determinant

$$f(x, y) = \begin{vmatrix} -2x + 2 & -2y \\ -2y & 2x + 2 \end{vmatrix} = -4(x^2 + y^2 - 1).$$

We could obtain an inversion equation by solving the equations:

$$\begin{bmatrix} -2x + 2 & -2y \\ -2y & 2x + 2 \end{bmatrix} \begin{bmatrix} 1 \\ t \end{bmatrix} = 0,$$

from which $t = y/(x + 1)$ or $t = (1 - x)/y$.

Two remarks should be made. First, if $a(t), b(t)$ and $c(t)$ in (15.1) are not relatively prime, the common factor should be removed before the resultant method is applied. Otherwise, the resultant will be identically zero, containing no information about the curve, since p and q have always common solutions for arbitrary (x, y) pair. Second, if the degrees of $p(x, t), q(y, t)$ with respect to t are not the same, the variant of Bezout’s resultant is used.

15.4.2. Gröbner basis technique

In order to use Gröbner basis method for implicitizing a rational parametric curve defined by (15.1) with $GCD(a(t), b(t), c(t)) = 1$ (otherwise, the common factor can be removed), we define the ideal

$$I = \langle c(t)x - a(t), c(t)y - b(t) \rangle \subset R[x, y, t]. \tag{15.9}$$

If $f(x, y) = 0$ is the implicit equation of (15.1), then $f \in I \cap R[x, y]$. To guarantee f appears in the Gröbner basis, we order the variables $t > x > y$, and then construct the Gröbner basis with the lexicographic ordering for the ideal I . The lexicographic ordering results in a Gröbner basis that has a triangular structure. Thus the Gröbner basis obtained will contain the curve’s implicit form—an element which does not involve t , and an inversion—an element which is linear in t .

In the example of the circle (15.8), $I = \langle (1 + t^2)x - (1 - t^2), (1 + t^2)y - 2t \rangle$. Using the computer algebra system MAPLE, we obtain the Gröbner basis $I = \langle -y + x + tx, x - 1 + yt, y^2 + x^2 - 1 \rangle$. Therefore the polynomial $y^2 + x^2 - 1$ gives the implicit equation $x^2 + y^2 - 1 = 0$. The two other polynomials $-y + x + tx$ and $x - 1 + yt$ are linear in t and thus provide the inversion $t = y/(1 + x)$ or $t = (1 - x)/y$.

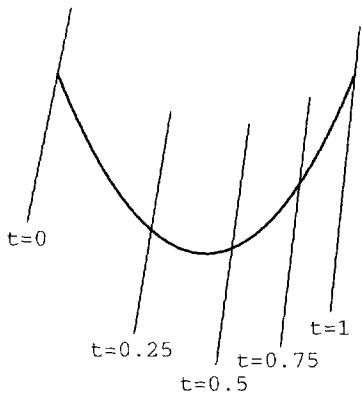


Figure 15.1. Parametric Curve and Moving Lines

15.4.3. Moving curve technique

A recent development in impliciting a planar rational parametric curve is the “moving curve” method [19,42,44]. A *moving curve* is defined as

$$C(x, y, w; t) := \sum_{i=0}^m f_i(x, y, w)t^i$$

where $f_i(x, y, w)$ is a homogeneous polynomial of degree d . Thus $C(x, y, w; t) = 0$ is a family of algebraic curves, with one curve corresponding to each t . In particular, when $d = 1$, $C(x, y, w; t) = 0$ is a family of implicitly defined lines. Therefore we call it a *moving line* of degree m . Likewise, $C(x, y, w; t) = 0$ is called a *moving conic* of degree m when $d = 2$. A moving curve $C(x, y, w; t) = 0$ is said to *follow* a planar rational curve (15.1) if $C(a(t), b(t), c(t); t)$ is identically zero, that is, if for all values of t , the point $(a(t)/c(t), b(t)/c(t))$ lies on the moving curve $C(x, y, w; t) = 0$. For example, each row of Bezout’s matrix or Sylvester’s matrix corresponds to a moving line following the curve. Figure 15.1 illustrates a parametric curve and a few moving lines.

The moving curve technique identifies $m + 1$ independent moving curves that follow a given rational curve. A square matrix can then be formed from the coefficients of these moving curves with respect to t and the determinant of the matrix gives the desired implicit equation. In general, such a collection of moving curves can be found by solving a set of linear equations [43]. For example, a degree $n - 1$ moving line

$$C(x, y, w; t) = \sum_{i=0}^{n-1} (A_i x + B_i y + C_i w)t^i = 0 \tag{15.10}$$

follows a rational curve (15.1) if

$$\sum_{i=0}^{n-1} (A_i a(t) + B_i b(t) + C_i c(t))t^i \equiv 0$$

which can be expressed as $2n$ linear equations with $3n$ unknowns

$$\begin{bmatrix} a_0 & b_0 & c_0 & 0 & \cdots & 0 & 0 & 0 \\ a_1 & b_1 & c_1 & a_0 & \cdots & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & a_1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_n & b_n & c_n & a_{n-1} & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & a_n & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & a_n & b_n & c_n \end{bmatrix} \begin{bmatrix} A_0 \\ B_0 \\ C_0 \\ \vdots \\ \vdots \\ A_{n-1} \\ B_{n-1} \\ C_{n-1} \end{bmatrix} = 0. \tag{15.11}$$

where a_i, b_i, c_i are the coefficients of the polynomials $a(t), b(t), c(t)$. Thus solving the equations for A_i, B_i, C_i yields n linearly independent moving lines of degree $n - 1$ that follows the curve.

Recently the problem of finding an appropriate set of moving lines was illuminated by the description of the μ -basis [19]. A μ -basis for a degree- n planar rational curve consists of two moving lines $p(x, y, t)$ and $q(x, y, t)$, of degree μ and $n - \mu$ respectively, which form an ideal basis for all moving lines that follow the curve. An efficient method of computing the μ -basis is given in [52]. Once the μ -basis of a curve is known, the problem of finding $m + 1$ linearly independent moving curves is greatly simplified. For example, consider the degree four curve:

$$x = \frac{t^4 + 2t^3 + t^2 + t + 1}{-t^4 - 2t^2 - 2}, \quad y = \frac{-t^3 + t^2 + 2t}{-t^4 - 2t^2 - 2}.$$

This curve has a μ -basis of

$$p = (x + y + 1)t^2 + t + y, \quad q = (x + 1)t^2 + 2t + 2x + y + 1.$$

Thus four moving lines of degree 3 are $p, t p, q, t q$. Moreover, two moving conics of degree 1 can be obtained by taking Bezout's resultant of p and q . Each row in Bezout's matrix corresponds to a moving conic. Therefore the implicit equation can be expressed as a 2×2 determinant whose elements are quadratic in x and y . In general, for a degree n rational curve, using a variant of Bezout's resultant on the μ -basis, we can write the implicit equation of the rational curve as the determinant of an $(n - \mu) \times (n - \mu)$ matrix with μ rows whose elements are quadratic in x and y , and the remaining $n - 2\mu$ rows with elements linear in x and y , while conventional implicitization methods generate the determinant of an $n \times n$ matrix [22,34].

15.5. CURVE PARAMETRIZATION

15.5.1. Planar algebraic curves

Implicitization shows that a degree n parametric curve can be represented using a degree n implicit equation. Any implicit equation that can be obtained by implicitizing a parametric curve is said to be *rational* (in other words, a rational curve is any curve which can be parametrized using rational functions). All algebraic quadratic curves have rational

quadratic parametrizations, whereas algebraic curves of degree greater than two are not generally rational.

A degree n algebraic curve is defined by a degree n polynomial $f(x, y)$, which has $(n + 1)(n + 2)/2$ terms. For example, a degree two algebraic curve has six terms: $a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6 = 0$. However, any one coefficient can be specified by scaling all of the other coefficients. (For example, the coefficient of x^2 in the quadratic example can be set to 1 by dividing all the coefficients by a_1). Thus, there is an $(n + 1)(n + 2)/2 - 1 = n(n + 3)/2$ dimensional family of degree n algebraic curves. Geometrically, this means a degree n algebraic curve can be forced to interpolate $n(n + 3)/2$ points in general position.

The parametric equation of a rational degree n curve has $3(n + 1)$ coefficients. However, any one of these can be specified by scaling all of the other coefficients. Also, three other coefficients can be specified by changing the parametrization by a rational linear transformation $s = \frac{at+b}{ct+d}$, and consequently there is a $3n - 1$ dimensional family of degree n rational curves.

15.5.2. Genus and rationality

The condition under which an implicit algebraic curve can be parametrized using rational polynomials is that its *genus* must be zero [49]. Basically, the genus of a curve is given by the formula $g = \frac{(n-1)(n-2)}{2} - d$ where g is the genus, n is the degree, and d is the number of double points. There are some subtleties involved in this equation if the singularities are not simple double points, but we will not concern ourselves with them.

A double point on a curve is a point for which $f(x, y) = f_x(x, y) = f_y(x, y) = 0$ where the subscripts x and y denote partial differentiations, and for a point of multiplicity k , all partials up to order $k - 1$ vanish. Geometrically, a double point means that any straight line through it intersects the curve at least twice at this point.

We see immediately that all curves of degree one and two have genus zero and thus can be parametrized using rational polynomials. A degree three algebraic curve is rational only if it has a double point.

An irreducible curve is one whose implicit equation $f(x, y) = 0$ cannot be factored. Rational curves (that can be parametrized using a single parametric equation) are irreducible, and an irreducible curve of degree n can have at most $(n - 1)(n - 2)/2$ double points. Thus, a rational curve has the most double points possible for a curve of its degree.

15.5.3. Parametrizing curves

One way to parametrize a degree two algebraic curve is to transform the conic into the standard form which has already a parametrization, and then to transform the standard parametrization back. The standard equations of conics are: $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ for an ellipse, $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$ for a hyperbola, and $y^2 = 2px$ for a parabola. They can be parametrized as $(a\frac{1-t^2}{1+t^2}, b\frac{2t}{1+t^2})$, $(a\frac{1+t^2}{1-t^2}, b\frac{2t}{1-t^2})$, and $(\frac{t^2}{2p}, t)$, respectively. Therefore the main step is to find a nonsingular affine coordinate transformation. This can be carried out as follows.

Suppose the conic equation is $Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Ey + F = 0$. First we convert the equation into the form

$$\bar{A}\bar{x}^2 + \bar{C}\bar{y}^2 + 2\bar{D}\bar{x} + 2\bar{E}\bar{y} + \bar{F} = 0. \quad (15.12)$$

If $B = 0$, it is done. If $A = C = 0$ and $B \neq 0$, then set $x = \bar{x} + \bar{y}$ and $y = \bar{x} - \bar{y}$.

Otherwise, if $A \neq 0, B \neq 0$, then

$$A\left(x + \frac{B}{A}y\right)^2 + \left(C - \frac{B^2}{A}\right)y^2 + 2Dx + 2Ey + F = 0.$$

Thus we only need to set $\bar{x} = x + \frac{B}{A}y$ and $\bar{y} = y$. A similar transformation can be derived if $C \neq 0$ and $B \neq 0$.

Second, if one of \bar{A} and \bar{C} in (15.12) is zero, the curve is a parabola. Assume that $\bar{A} \neq 0$ and $\bar{C} = 0$. Then $\bar{A}\left(\bar{x} + \frac{\bar{D}}{\bar{A}}\right)^2 = -2\bar{E}\bar{y} - \bar{F} + \frac{\bar{D}^2}{\bar{A}}$. Thus setting

$$x' = \bar{x} + \frac{\bar{D}}{\bar{A}}, \quad y' = \bar{y} + \frac{\bar{F} - \bar{D}^2/\bar{A}}{2\bar{E}}$$

arrives at the standard parabola equation. A similar process deals with the case of $\bar{A} = 0, \bar{C} \neq 0$. If both of \bar{A} and \bar{C} are nonzero, we have $\bar{A}\left(\bar{x} + \frac{\bar{D}}{\bar{A}}\right)^2 + \bar{C}\left(\bar{y} + \frac{\bar{E}}{\bar{C}}\right)^2 = \frac{\bar{D}^2}{\bar{A}} + \frac{\bar{E}^2}{\bar{C}} - \bar{F}$. Therefore we can take a translation of $x' = \bar{x} + \frac{\bar{D}}{\bar{A}}$ and $y' = \bar{y} + \frac{\bar{E}}{\bar{C}}$ to make the curve equation in the standard form. Composing the transformations in the above two steps gives the required coordinate transformation.

Another method to parametrize a conic is to establish a one-one correspondence between points on the curve and a family of lines through a point on the curve, which is called a *pencil-of-lines*. This pencil-of-lines method is most easily illustrated by translating the curve so that it passes through the origin, such as does the curve

$$x^2 - 2x + 4y^2 = 0$$

which is an ellipse centered at $(1, 0)$. We next make the substitution $y = tx$ and solve for x as a function of t : $x^2(1 + 4t^2) - 2x = 0$. Then

$$x = \frac{2}{1 + 4t^2}, \quad y = tx = \frac{2t}{1 + 4t^2}$$

Notice that $y = tx$ is a family of lines through the origin. The variable line $y = tx$ intersects the curve once at the origin, and at exactly one other point (because of Bezout's theorem: two algebraic curves of degree m and n intersect at either mn points or else they have common components [49]). Thus, we have established a one-one correspondence between points on the curve and values t which correspond to lines containing that point and the origin. The ellipse parametrized in this manner is shown in Figure 15.2.

To parametrize a genus zero cubic curve, one must first find its double point, which is done by solving $h(x, y) = h_x(x, y) = h_y(x, y) = 0$. Once the location of the double point is determined, one can translate the curve so that the double point lies on the origin. Then the same trick in the pencil-of-lines approach for conics can be played with this cubic curve since the curve now has an equation involving terms of degree two and three only.

Consider this example of the cubic curve

$$f(x, y) = -21 + 46x - 13x^2 + x^3 + 25y - 23xy + 3x^2y - 9y^2 + 3xy^2 + y^3 = 0$$

for which

$$f_x(x, y) = 46 - 26x + 3x^2 - 23y + 6xy + 3y^2$$

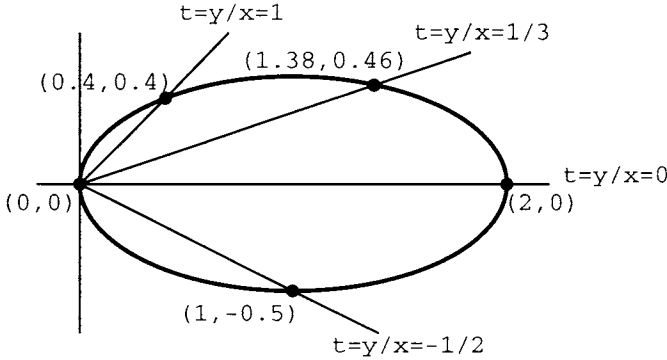


Figure 15.2. Parametrizing an Ellipse

and

$$f_y(x, y) = 25 - 23x + 3x^2 - 18y + 6xy + 3y^2$$

We compute the x coordinates of the intersections of $f_x = 0$ and $f_y = 0$ by taking the resultant of f_x and f_y with respect to y :

$$\text{Resultant}(f_x, f_y, y) = 174 - 159x + 36x^2$$

whose roots are $x = 2$ and $x = \frac{29}{12}$. Likewise the y coordinates of the intersections of $f_x = 0$ and $f_y = 0$ are found by taking the resultant of f_x and f_y with respect to x :

$$\text{Resultant}(f_x, f_y, x) = 297 - 207y + 36y^2$$

whose roots are $y = 3$ and $y = \frac{11}{4}$. From these clues, we find that the only values of (x, y) which satisfy $f(x, y) = f_x(x, y) = f_y(x, y) = 0$ are $(x, y) = (2, 3)$. This is therefore the double point.

The double point can also be found by computing the Gröbner basis of $\langle f, f_x, f_y \rangle$ using lex ordering with $x > y$, the Gröbner basis is $\{x - 2, y - 3\}$.

This curve can be parametrized by translating the implicit curve so that the double point lies at the origin. This is done by making the substitution $x = \bar{x} + 2, y = \bar{y} + 3$, yielding

$$2\bar{x}^2 + \bar{x}^3 + 7\bar{x}\bar{y} + 3\bar{x}^2\bar{y} + 6\bar{y}^2 + 3\bar{x}\bar{y}^2 + \bar{y}^3 = 0$$

Parametrization is then performed using the method discussed earlier in this section,

$$\bar{x} = -\frac{6t^2 + 7t + 2}{t^3 + 3t^2 + 3t + 1}; \quad \bar{y} = -\frac{6t^3 + 7t^2 + 2t}{t^3 + 3t^2 + 3t + 1}$$

and the parametrized curve is translated back so that the doubled point is again at $(2, 3)$ (see Figure 15.3):

$$x = -\frac{6t^2 + 7t + 2}{t^3 + 3t^2 + 3t + 1} + 2 = \frac{2t^3 - t}{t^3 + 3t^2 + 3t + 1};$$

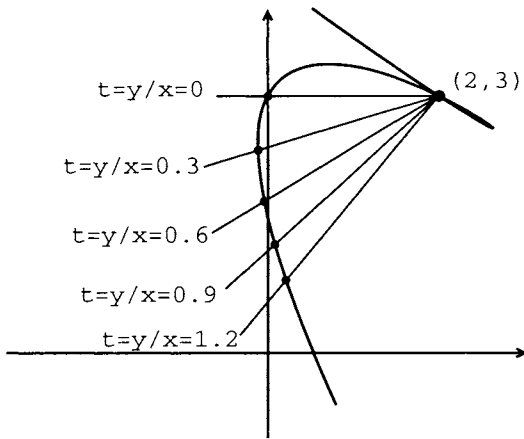


Figure 15.3. Parametrizing a Cubic Curve

$$y = -\frac{6t^3 + 7t^2 + 2t}{t^3 + 3t^2 + 3t + 1} + 3 = \frac{-3t^3 + 2t^2 + 7t + 3}{t^3 + 3t^2 + 3t + 1}$$

For a general algebraic curve, the parametrization problem involves two steps: determine whether it admits a rational parametric representation, and find one if so. The algorithms, in general, are not as simple as for conics or cubics. References [1–4,47] provide various computational techniques for parametrizing algebraic curves.

15.6. INTERSECTION COMPUTATIONS

We now consider how to compute the points at which two curves intersect. Intersection algorithms for two Bézier curves are commonly based on subdivision, or using some numerical algorithms such as a multivariate Newton method. The former takes advantage of the properties of Bézier or B-spline representations and focuses on the intersection points within the specified intervals. The latter method is not robust: it is difficult or impossible to assure that all intersection points have been found.

Algebraic methods provide a systematic way for computing intersections. As noted in Section 15.2, a variety $V(f_1, \dots, f_s)$ defined by more than one polynomial ($s > 1$) is the intersection of the varieties $V(f_1), \dots, V(f_s)$. Therefore, intersection computation is equivalent to determining a variety. Using either resultants or Gröbner bases, this finally reduces to the problem of finding the roots of a polynomial on one variable.

15.6.1. Parametric curve and implicit curve

Given one curve defined by the parametric equation $(x, y) = (x(t), y(t))$ and a second curve defined by the implicit equation $f(x, y) = 0$, we replace all occurrences of x and y in the implicit equation by $x(t)$ and $y(t)$, respectively. These substitutions create a polynomial $f(x(t), y(t)) = g(t)$ whose roots are the parameter values of the intersection points. The (x, y) coordinates of these intersection points can be easily obtained by substituting the parametric values into the equation of the parametric curve.

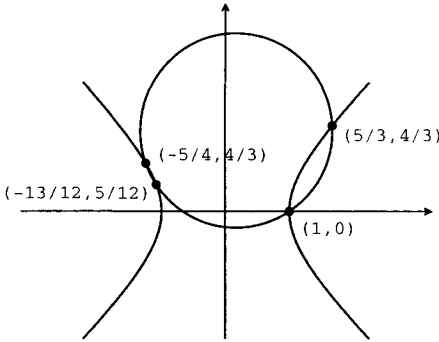


Figure 15.4. Circle and Hyperbola

15.6.2. Implicit curve and implicit curve

For two algebraic curves $f_1(x, y) = 0$ and $f_2(x, y) = 0$, the problem of computing the intersection amounts to computing the variety $V(f_1, f_2)$.

One direct method for computing the variety $V(f_1, f_2)$ is to take the resultant of f_1 and f_2 with respect to x or y . The x -resultant is computed by treating f_1 and f_2 as polynomials in x whose coefficients are polynomials in y . The x -resultant eliminates x and produces a polynomial in y whose roots are the y coordinates of the intersection points.

We illustrate with a circle $6x^2 + 6y^2 - 2x - 15y - 4 = 0$ and a hyperbola $x^2 - y^2 - 1 = 0$ (see Figure 15.4).

The x -resultant of these two implicit equations is $144y^4 - 360y^3 + 269y^2 - 60y$ whose roots are $y = 0$, $y = 4/3$, $y = 3/4$, and $y = 5/12$. These are the y -coordinates of the points of intersection of the two curves.

We can use the y -resultant to find the x -coordinates of the points of intersection. The y -resultant is $144x^4 - 48x^3 - 461x^2 + 40x + 325$ which has roots $x = 1$, $x = 5/3$, $x = -5/4$, and $x = -13/12$.

We now know the x and y components of the points of intersection, but we don't know which x goes with which y ! One way to determine that is simply to evaluate each curve equation with every x and every y to see which (x, y) pairs satisfy both curve equations simultaneously. A more clever way is to use Euclid's algorithm which computes the GCD of two polynomials. In fact, Euclid's algorithm spares us the trouble of computing both the x -resultant and the y -resultant.

Suppose we had only computed the y -resultant and we wanted to find the y -coordinate of the point of intersection whose x -coordinate is $5/3$. That is to say, we want to find a point $(\frac{5}{3}, y)$ which satisfies both curve equations. We substitute $x = 5/3$ into the circle equation to get $16/6 - y^2 = 0$ and into the hyperbola equation to get $6y^2 - 15y + 28/3 = 0$. We now simply want to find a value of y which satisfies both of these equations. Euclid's algorithm tells us that the GCD of these two is $3y - 4 = 0$, and thus one point of intersection is $(\frac{5}{3}, \frac{4}{3})$.

Gröbner bases also provide a systematic computational method with the assurance

that all intersection points have been found. The general strategy is based on the simple observation that $V(f_1, \dots, f_s) = V(\langle f_1, \dots, f_s \rangle)$. Consequently, if f_1, \dots, f_s and g_1, \dots, g_t are generators of the same ideal, then

$$V(f_1, \dots, f_s) = V(g_1, \dots, g_t).$$

Consider the previous example of a circle $V(6x^2 + 6y^2 - 2x - 15y - 4)$ and a hyperbola $V(x^2 - y^2 - 1)$. It can be verified that

$$\begin{aligned} \langle 6x^2 + 6y^2 - 2x - 15y - 4, x^2 - y^2 - 1 \rangle = \\ \langle -12x^2 + 2x + 15y + 10, 144x^4 - 48x^3 - 461x^2 + 40x + 325 \rangle \quad (\text{the Gröbner basis}) \end{aligned}$$

Since any point of intersection must be zeros of all generators of the ideal, the only possible x -coordinates for the intersection points must be roots of $144x^4 - 48x^3 - 461x^2 + 40x + 325 = 0$ (the roots are $\frac{5}{3}$, 1 , $-\frac{13}{12}$, and $-\frac{5}{4}$). The corresponding y coordinates can then be solved using $-12x^2 + 2x + 15y + 10 = 0$.

15.6.3. Parametric curve and parametric curve

If we begin with two parametric curves, we can first implicitize one of them, and then use the substitution method to compute the intersection points. We illustrate this process by intersecting the curve

$$(x, y) = \left(-\frac{1 + s^2}{1 - s^2}, \frac{2s}{1 - s^2} \right)$$

with the curve

$$(x, y) = \left(\frac{27t^2 - 239t - 116}{6t^2 + 81t + 165}, \frac{9t^2 + 96t - 33}{6t^2 + 81t + 165} \right)$$

(see Figure 15.5). The two curves intersect four times, which is the most that two quadratic curves can intersect. We implicitize the first curve and get the implicit equation $x^2 - y^2 - 1 = 0$. Substituting the parametric equation of the second curve into this implicit equation and clearing the denominator, we arrive at the intersection equation:

$$18t^4 - 459t^3 + 991t^2 + 1031t - 437 = 0.$$

We now compute the roots of this degree four polynomial, which are 23 , $1/3$, $19/6$ and -1 . These are the parametric values on the second curve for the intersection points. From the parametric equation of the second curve, the corresponding (x, y) coordinates can be easily found: $(5/3, 4/3)$, $(-1, 0)$, $(-5/4, 3/4)$, $(5/3, -4/3)$. The parametric values on the first curve for the intersection points can be found from the inversion formulas $s = y/(1 - x)$ and $s = -(1 + x)/y$. They are -2 , 0 , $1/3$ and 2 , respectively.

Tests indicate that this implicitization-based intersection algorithm is several times faster than subdivision methods for quadratic and cubic curves, but subdivision methods are faster for curves of degree five and greater [38].

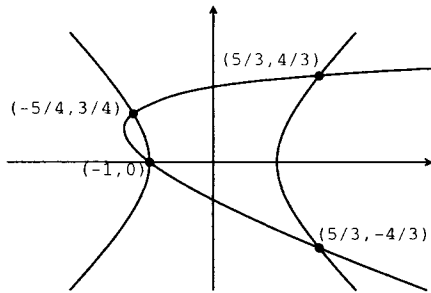


Figure 15.5. Intersections of Two Parametric Curves

Gröbner bases and resultants can also be used for finding the intersections between two parametric curves. The intersections of the curves $(x, y) = (a_1(s)/c_1(s), b_1(s)/c_1(s))$ and $(x, y) = (a_2(t)/c_2(t), b_2(t)/c_2(t))$ satisfy

$$\frac{a_1(s)}{c_1(s)} = \frac{a_2(t)}{c_2(t)}, \quad \frac{b_1(s)}{c_1(s)} = \frac{b_2(t)}{c_2(t)}.$$

This prompts us to compute the variety $V(a_1(s)c_2(t) - a_2(t)c_1(s), b_1(s)c_2(t) - b_2(t)c_1(s))$ in the (s, t) -space. The methods discussed in Section 15.6.2 can then be employed. For the previous example, the Gröbner basis using lexicographic ordering with $t > s$ is

$$\begin{aligned} & \langle 144t^2 + 1944t - 4389s^5 - 6448s^4 + 32169s^3 + 11072s^2 - 27780s - 664, \\ & 48s - 6t + 6s^2t + 2 + 14s^2 - 69s^3 - 16s^4 + 21s^5, -4s + 12s^2 + 5s^3 - 15s^4 - s^5 + 3s^6 \rangle. \end{aligned}$$

Solving the third polynomial in the Gröbner basis gives six roots: $0, 1, 2, 1/3, -2, -1$. Note that the solutions of 1 and -1 are actually the roots of the denominator $1 - s^2$ of the first curve, and thus should be discarded. We then substitute the rest four roots into the second polynomial in the Gröbner basis and solve for the corresponding values of t . These values of s and t are the parametric values of the intersection points on the first and second parametric curves.

15.7. SURFACES

This section briefly overviews some applications of algebraic methods to surfaces. A rational parametric surface is usually defined by (15.2). We denote the maximum of the total degrees of the polynomials $a(s, t)$, $b(s, t)$, $c(s, t)$ and $d(s, t)$ by n and call it the *parametric degree*. The implicit equation of an algebraic surface is given by $f(x, y, z) = 0$ where $f(x, y, z)$ is a polynomial in x, y, z , and its maximum degree is denoted by m , called the *implicit degree*. Like curves, it is *always* possible to find an implicit equation of a parametric surface, but parametric equations can generally be found only for a very select class of implicit surfaces. Algorithms for implicitization and inversion also exist for surfaces [15,30], but the process for surfaces is much more complicated than the curve case. For example, a degree n plane parametric curve has an implicit equation that is also degree n . For surfaces, however, the implicit degree m can be as high as n^2 if the parametric degree is n .

15.7.1. Implicit degree of a rational parametric surface

The implicit degree can be thought of as the number of times that the surface is intersected by a generic straight line [15,48]. Define a generic straight line as the intersection of two distinct planes in general position $a_1x + a_2y + a_3z + a_4 = 0$ and $b_1x + b_2y + b_3z + b_4 = 0$. The planes intersect the parametric surface (15.2) in curves

$$a_1a(s, t) + a_2b(s, t) + a_3c(s, t) + a_4d(s, t) = 0 \quad (15.13)$$

and

$$b_1a(s, t) + b_2b(s, t) + b_3c(s, t) + b_4d(s, t) = 0 \quad (15.14)$$

These curves are each degree n in s, t . By Bezout's theorem, these two curves intersect in n^2 points, which must also be the number of times that the straight line common to the two planes intersects the surface. Thus, the degree of the surface, and of its implicit equation, is n^2 .

It seems curious that there are gaps in the sequence of the implicit degrees of parametric surfaces: 1, 4, 9, Are there no parametric surfaces whose implicit degree is 3 or 5 for example, or under what conditions will the degree decrease?

It may happen that there are values s_b and t_b satisfying $a(s_b, t_b) = b(s_b, t_b) = c(s_b, t_b) = d(s_b, t_b) = 0$. These parameter pairs (s_b, t_b) are referred to as *base points*. If a base point exists, the intersection curve of any plane with the surface will contain the base point. Thus, the above two curves will intersect at the base point and at $n^2 - 1$ other points. However, since the base point does not map to a unique point on the surface ($x = y = z = 0/0$ is undefined), this does not represent a point at which the straight line intersects the surface, and the degree of the surface is therefore $n^2 - 1$. Each additional simple base point diminishes the degree of the surface by one. Base points at infinity occur when all plane sections have a common asymptotic direction.

To understand the influence of more complicated base points on the implicit degree, consider the linear system of all curves given by (15.13), where each curve in the linear system is the intersection of the surface with the plane $a_1x + a_2y + a_3z + a_4 = 0$. A base point is any point in common with all members of the linear system. If two general curves in the linear system are tangent at a base point, they intersect twice at the base point and the degree of the surface becomes $n^2 - 2$. If two general curves in the linear system have a double point in common, they intersect four times at that base point and the degree becomes $n^2 - 4$. Thus, a general degree formula is $n^2 - \rho$ where ρ is the total number of times that two general curves in the linear system intersect at base points. This also assumes that the surface has a one-to-one parametrization.

If the surface (15.2) is a tensor product surface—one of the most popular representations for surfaces in CAGD—the parametrization is a bi-degree (n_s, n_t) parametrization in s and t . That means, n_s and n_t are the highest degrees of the parametric equations with respect to s and t . In this case, the total parametric degree is $n = n_s + n_t$. But there exist two base points at infinity, corresponding to $s = \infty, t = \infty$, counted at least n_t^2 and n_s^2 times respectively [14]. Thus the implicit degree of a bi-degree (n_s, n_t) parametrized rational surface is at most $(n_s + n_t)^2 - n_s^2 - n_t^2 = 2n_s n_t$. For example, a bicubic surface is usually of implicit degree 18.

15.7.2. Surface intersection curves

Surface/surface intersection (i.e., finding the intersection curve of two surfaces) is an important geometric operation in CAGD. The usual approach is to compute an approximation for the intersection curve. Algebraic geometry provides important information on the nature of intersections of parametric surfaces. For example, the degree of the intersection curve is easy to determine using Bezout's theorem which states that two surfaces of degree m and n respectively intersect in a curve of degree mn . Therefore if two surfaces have implicit degree n_1 and n_2 , the intersection curve has a degree n_1n_2 (unless the surfaces have common components). Thus, two bicubic patches generally intersect in a curve of degree 324.

15.7.3. Implicitization

Resultants can be used to implicitize a rational parametric surface. Dixon's resultant is a good choice, because it works on three polynomials in two variables. Given a rational parametric surface (15.2), construct three auxiliary polynomials:

$$\begin{aligned} p(x, s, t) &= d(s, t)x - a(s, t), \\ q(y, s, t) &= d(s, t)y - b(s, t), \\ h(z, s, t) &= d(s, t)z - c(s, t) \end{aligned}$$

Note that $p(x, s, t) = q(y, s, t) = h(z, s, t) = 0$ only for values of x, y, z and s, t which satisfy (15.2). View $p(x, s, t)$, $q(y, s, t)$ and $h(z, s, t)$ as polynomials in s and t whose coefficients are linear in x , in y and in z , respectively. Then applying Dixon's resultant to these polynomials to eliminate s and t , we obtain a polynomial in x, y and z which we denote $f(x, y, z)$. Thus $f(x, y, z) = 0$ defines the implicit equation of the rational surface. In addition, by Cramer's rule, taking the ratio of the determinants of the submatrices from the Dixon's matrix corresponding to the terms $s^i t^j$ and $s^{i-1} t^j$, or the terms $s^i t^j$ and $s^i t^{j-1}$ yields the inversion equations of $s = s^i t^j / s^{i-1} t^j$ and $t = s^i t^j / s^i t^{j-1}$. Unfortunately, if the surface has finite basepoints, the resultant is identically zero and the algorithm fails.

The implicitization of rational parametric surfaces can also be accomplished by computing the elimination ideal. The Rational Implicitization Theorem [17] states that if $J = \langle dx - a, dy - b, dz - c, 1 - dw \rangle$, then $V(J \cap R[x, y, z])$ is the smallest variety in R^3 containing the parametric surface. The polynomial $1 - dw$ is introduced to assure that the method will work even if base points are present. Otherwise, base points would cause $J \cap R[x, y, z] = \{0\}$. In practical computation, we construct the Gröbner basis with the lexicographic ordering for the ideal J with $s > t > x > y > z$. The Gröbner basis will contain a polynomial in x, y, z . This is the implicit equation. If the parametrization of the surface is a one-to-one map, two polynomials linear in t and s are also contained in the Gröbner basis. They can produce the inversion maps.

Notwithstanding the robustness and elegance of the Gröbner basis solution to surface implicitization, it is not very computationally efficient. Recently, a promising new method, called the *moving surface method*, has been proposed for implicitizing rational surfaces [43]. Like resultants, the implicit equation is expressed as the determinant of a matrix.

We define a moving surface as

$$g(x, y, z, s, t) = \sum_{i=1}^{\sigma} f_i(x, y, z) \gamma_i(s, t) = 0$$

where the equations $f_i(x, y, z) = 0, i = 1, \dots, \sigma$ define a collection of implicit surfaces and where the $\gamma_i(s, t), i = 1, \dots, \sigma$ are a collection of polynomials in s and t . We require the $\gamma_i(s, t)$ to be linearly independent and to be relatively prime. A moving surface is said to “follow” a rational parametric surface (15.2) if

$$g\left(\frac{a(s, t)}{d(s, t)}, \frac{b(s, t)}{d(s, t)}, \frac{c(s, t)}{d(s, t)}, s, t\right) \equiv 0.$$

If we can find a set of σ moving surfaces

$$g_j(x, y, z, s, t) = \sum_{i=1}^{\sigma} f_{ji}(x, y, z) \gamma_i(s, t) = 0, \quad j = 1, \dots, \sigma$$

each of which follows a given rational surface, then

$$f(x, y, z) = \begin{vmatrix} f_{11}(x, y, z) & \cdots & f_{1\sigma}(x, y, z) \\ \vdots & \vdots & \vdots \\ f_{\sigma 1}(x, y, z) & \cdots & f_{\sigma\sigma}(x, y, z) \end{vmatrix} = 0$$

gives the implicit equation — as long as the degree of $f(x, y, z)$ is equal to the degree of the implicit equation of the rational surfaces.

In comparison, Gröbner basis method theoretically provides an elegant solution to implicitization of parametric surfaces, but involves a huge computation which limits its use in practice. The method of resultants is efficient, but fails when base points occur. Multivariate resultants can also be used for implicitization [15]. The moving surface method actually simplifies in the presence of base points. Furthermore, the method of moving surfaces provides a very compact representation for the implicit equation of a surface. For example, a bicubic patch can, in general, be written as a 9×9 determinant whose elements are all degree two in x, y, z . By contrast, Dixon’s resultant produces an 18×18 determinant. However, further study is needed on the moving surface method.

In summary, the operation of surface implicitization has not yet gained widespread use in practice, partly because the degree explosion that one encounters when moving from the parametric to the implicit form counteracts most algorithmic advantages that the implicit form might have over the parametric, and also because the computational complexity is very large, especially in the event of base points.

15.7.4. Parametrizaion

For an algebraic surface of arbitrary degree, Castelnuovo gave a necessary and sufficient condition for the existence of the rational parametrization [51]. Unfortunately, this criterion does not provide a constructive approach to parametrization. A systematic method for parametrizing a general rational algebraic surface is under investigation. Recently, various computational algorithms for parametrizing certain lower degree algebraic surfaces have been developed.

All degree two surfaces are rational. We can parametrize a quadric surface much like we did for conics. For example, we can use a pencil-of-lines approach by first translating the surface so that it touches the origin. Then define a line through the origin as the intersection of two pencils of planes: $y = sx$ and $z = tx$, say, and intersect the surface with that line. We demonstrate the procedure with a sphere $x^2 + y^2 + z^2 - 2z = 0$. Substituting $y = sx$ and $z = tx$, we obtain $x^2(1 + s^2 + t^2) - 2tx = 0$. This gives the x coordinates of the two points where the line intersects the sphere. Discarding the intersection at the origin gives the parametrization

$$x = \frac{2t}{1 + s^2 + t^2}, \quad y = \frac{2st}{1 + s^2 + t^2}, \quad z = \frac{2t^2}{1 + s^2 + t^2}$$

Most cubic surfaces are also rational. The only exception is the ruled cubic generated by a non-rational cubic curve. The cubic surface has a fascinating geometry. For example, the general cubic surface contains 27 straight lines, and those lines can be used in determining a parametrization for the surface. A detailed discussion can be found in [9,39,41].

15.8. OTHER ISSUES

The algebraic approaches to implicitization, parametrization and inversion illustrate how algebraic concepts and methods, such as resultants and Gröbner bases, help us analyze and solve some common problems in CAGD. Space limitations have prohibited the inclusion of several additional related topics, such as the following:

- A curve or a surface is said to be properly parametrized if to each point on the curve, except for possibly a finite number of points, there corresponds only one parameter value. It is natural to ask whether any improperly parametrized curve or surface can be reparametrized to become properly parametrized. For a rational curve, a classical theorem due to Lüroth guarantees the existence of a reparametrization [49]. However, for a rational surface, it depends on the base field where the surface is defined. Gröbner basis methods can be used for detecting and correcting improper parametrization [24].
- Geometric continuity was originally introduced as a smoothness measure for parametric curves and surfaces. This concept is also meaningful for implicit curves and surfaces. The geometric continuity conditions for implicit surfaces is studied in [25]. This consideration is important when using algebraic surfaces in geometric modeling. One application is to construct blending algebraic surfaces with a specified continuity [50]. Algebraic surfaces have been shown to have advantages over parametric surfaces when performing the blending operation.
- The fact that the most popular free-form surface patches – bicubic patches – are actually algebraic degree 18 has tempted some researchers to investigate the possibility of using lower degree algebraic surfaces for modeling purposes. Surfaces in algebraic geometry are usually global in nature while surfaces in CAGD are usually finitely defined (i.e., patches). The main reasons that parametric surface patches have been so popular in CAGD are that they can be pieced together with any desired degree of

continuity, and that there exist many elegant, intuitively meaningful techniques for controlling their shape. Therefore, to make algebraic surfaces useful in CAGD, the Bernstein-Bézier techniques have been adapted in defining algebraic surfaces [36]. Meaningful and efficient methods, such as interpolation, least-squares approximation and interactive modification, have been developed to model complicated shapes using *piecewise* implicit algebraic surfaces [7,8,20,21].

- Algebraic tools such as resultants and discriminants can help compute the intersection points between a ray and a surface (useful for performing ray tracing), and can help compute the silhouette points or curves in a scene. Some techniques based on algebraic methods have been developed to accurately render surfaces using computer graphics, such as ray-tracing [29,35] or in scan-line algorithms [40].
- The methods in algebraic geometry assume the procedure is carried out using exact (integer or rational number) arithmetic. Nevertheless, commercial computer systems dealing with CAGD use floating point arithmetic. This fact heavily hinders applying algebraic geometry methods to the practical problems of CAGD. Therefore computational theories and techniques of algebraic geometry in floating point arithmetic are of high interest. Some strategies for using algebraic methods in a floating point environment are discussed in [45].

Interest in algebraic techniques for the CAGD is growing, and it is evident that algebraic geometry is a valuable resource for computer aided geometric design.

REFERENCES

1. S. Abhyankar and C. Bajaj. Automatic parametrization of rational curves and surfaces I: Conics and conicoids. *Computer-Aided Design*, 19(1):11-14, 1987.
2. S. Abhyankar and C. Bajaj. Automatic parametrization of rational curves and surfaces II: Cubics and cubicoids. *Computer-Aided Design*, 19(9):499-502, 1987.
3. S. Abhyankar and C. Bajaj. Automatic parametrization of rational curves and surfaces III: Algebraic plane curves. *Computer Aided Geometric Design*, 5:309-321, 1988.
4. S. Abhyankar and C. Bajaj. Automatic parametrization of rational curves and surfaces IV: Algebraic space curves. *ACM Transactions on Graphics*, 8(4):324-333, 1989.
5. S.S. Abhyankar. *Algebraic Geometry for Scientists and Engineers*. American Mathematical Society, Providence, R.I., 1990.
6. W. Adams and P. Loustau. *An introduction to Gröbner bases*. American Mathematical Society, Providence, R.I., 1994.
7. C. Bajaj and I. Ihm. Smoothing of polyhedra with implicit algebraic splines. *Computer Graphics*, 26(2):79-88, SIGGRAPH 92, Chicago, Illinois, 1992.
8. C. Bajaj, J. Chenm, and G. Xu. Modeling with cubic A-patches. *ACM Transactions on Graphics*, 14(2):103-133, 1995.
9. C. Bajaj, R. Holt, and A. Netravali. Rational parametrizations of nonsingular cubic surfaces. *ACM Transactions on Graphics*, 17(1):1-31, 1998.
10. T. Becker and V. Weispfenning. *Gröbner Bases: A Computational Approach to Commutative Algebra*. Springer-Verlag, 1993.

11. W. Boehm and H. Prautzsch. *Geometric Concepts for Geometric Design*. A K Peters, Welleley, Massachusetts, 1993.
12. B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes Nach Einem Nulldimensionalen Polynomideal*. PhD thesis (in German), Universität Innsbruck, Austria, 1965.
13. B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N.K. Bose, editor, *Multidimensional Systems Theory*, pages 184-232. D.Reidel Publishing Co., Netherlands, 1985.
14. E. Chionh. *Base Points, Resultants, and the Implicit Representation of Rational Surfaces*. PhD thesis, Dept. of Computer Science, University of Waterloo, Canada, 1990.
15. E. Chionh and R. Goldman. Degree, multiplicity, and inversion formulas for rational surfaces using u-resultants. *Computer Aided Geometric Design*, 9:93-108, 1992.
16. E. Chionh and R. Goldman. Elimination and resultants, Part 1: Elimination & bivariate resultants and Part 2: Multivariate resultants. *IEEE Computer Graphics and Applications*, 15(1):69-77 and 15(2):60-69, 1995.
17. D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer-Verlag, 1992.
18. D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer-Verlag, 1998.
19. D. Cox, T. Sederberg, and F. Chen. The moving line ideal basis of planar rational curves. *Computer Aided Geometric Design*, 15:803-827, 1998.
20. W. Dahmen. Smooth piecewise quadratic surfaces. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*. Boston, Academic Press, 1989.
21. W. Dahmen and T. Thamm-Schaar. Cubicoids: modeling and visualization. *Computer Aided Geometric Design*, 10(2):89-108, 1993.
22. Y. De Montaudouin and W. Tiller. The Cayley method in computer aided geometric design. *Computer Aided Geometric Design*, 1:309-326, 1984.
23. A.L. Dixon. The eliminant of three quantics in two independent variables. *Proc. London Math. Soc.*, 6(Ser.2):468-478, 1908.
24. X. Gao and S. Chou. Implicitization of rational parametric equations. *Journal of Symbolic Computation*, 14:459-470, 1992.
25. T. Garrity and J. Warren. Geometric continuity. *Computer Aided Geometric Design*, 8(1):51-66, 1991.
26. R. Goldman, T. Sederberg, and D. Anderson. Vector elimination: A technique for the implicitization, inversion, and intersection of planar parametric rational polynomial curves. *Computer Aided Geometric Design*, 1:327-356, 1984.
27. C. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann, San Mateo, 1989.
28. C. Hoffmann. Implicit curves and surfaces in computer-aided geometric design. *IEEE Computer Graphics and Applications*, 13(1):79-88, 1993.
29. J. Kajiya. Ray tracing parametric patches. *Computer Graphics*, 16(3):245-254, 1982.
30. D. Manocha and J. Canny. Algorithm for implicitizing rational parametric surfaces. *Computer Aided Geometric Design*, 9(1):25-51, 1992.
31. G. Salmon. *Modern Higher Algebra*. G.E. Steckert & Co., New York, 1985.
32. G. Salmon. *A Treatise on the Analytic Geometry of Three Dimensions*, Vol I, II,

- R. Rogers, editor, Reprinted. Chelsea Publishing, 1914.
33. T. Sederberg. *Implicit and Parametric Curves and Surfaces*. PhD thesis, Purdue University, West Lafayette, IN, 1983.
 34. T. Sederberg, D. Anderson, and R. Goldman. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics and Image Processing*, 28:72-84, 1984.
 35. T. Sederberg and D. Anderson. Ray tracing of Steiner patches. *Computer Graphics*, 18(3):159-164, 1984.
 36. T. Sederberg. Piecewise algebraic surface patches. *Computer Aided Geometric Design*, 2:53-59, 1985.
 37. T. Sederberg and R. Goldman. Algebraic geometry for computer-aided geometric design. *IEEE Computer Graphics and Applications*, 6(6):5-59, 1986.
 38. T. Sederberg and S. Parry. A comparison of curve-curve intersection algorithms. *Computer-Aided Design*, 18:58-63, 1986.
 39. T. Sederberg and J. Snively. Parametrizing cubic algebraic surfaces. In R.R. Martin, editor, *The Mathematics of Surfaces II*, pages 299-320. Oxford University Press, Oxford UK, 1987.
 40. T. Sederberg and A. Zundel. Scan line display of algebraic surfaces. *Computer Graphics*, 23(3):147-156, 1989.
 41. T. Sederberg. Techniques for cubic algebraic surfaces, parts 1 and 2. *IEEE Computer Graphics and Applications*, July 1990, 14-25 and September 1990, 12-21.
 42. T. Sederberg, T. Saito, D. Qi, and K. Klimaszewski. Curve implicitization using moving lines. *Computer Aided Geometric Design*, 11:687-706, 1994.
 43. T. Sederberg and F. Chen. Implicitization using moving curves and surfaces. *SIGGRAPH 95, Computer Graphics Proceedings*, Annual Conference Series, pages 301-308, 1995.
 44. T. Sederberg, R. Goldman, and H. Du. Implicitizing rational curves by the method of moving algebraic curves. *Journal of Symbolic Computation*, 23:153-175, 1997.
 45. T. Sederberg. Applications to computer aided geometric design. *Proceedings of Symposia in Applied Mathematics*, 53:67-89, 1998.
 46. J. Semple and L. Roth. *Introduction to Algebraic Geometry*. Oxford University Press, Oxford, U.K, 1949.
 47. J. Sendra and F. Winkler. Symbolic parametrization of curves. *Journal of Symbolic Computation*, 6:607-632, 1991.
 48. B.L. Van der Waerden. *Modern Algebra*. Frederick Ungar, New York, 2nd edition, 1950.
 49. R. Walker. *Algebraic Curves*. Dover Publications, New York, 1949.
 50. J. Warren. Blending algebraic surfaces. *ACM Transactions on Graphics*, 8:263-278, 1989.
 51. O. Zariski. Castelnuovo's criterion of rationality $p_a = P_2 = 0$ of an algebraic surface, III. *J. Math.*, 2:303-315, 1958.
 52. J. Zheng and T. Sederberg. A direct approach to computing the mu-basis of planar rational curves. *Journal of Symbolic Computation*, 31(5):619-629, 2001.

Chapter 16

Scattered Data Interpolation: Radial Basis and Other Methods

Suresh K. Lodha and Richard Franke

This chapter presents some techniques for solving scattered data interpolation for functional data. The focus here is to present some practical techniques for solving these problems using radial basis functions and some other local methods. Many of these techniques using radial basis functions have been developed only in last few years. Furthermore, although many of these techniques can be extended to higher dimensions, here we concern ourselves mostly with two and three dimensions. For a more comprehensive survey and literature on scattered data interpolation, we refer the reader to our earlier work [28].

16.1. INTRODUCTION

Scattered data interpolation and approximation problems arise in a variety of applications including meteorology, hydrology, oceanography, computer graphics, computer-aided geometric design, and scientific visualization. There exist several variants of the basic problem. The basic problem, referred to as the *functional scattered data problem* is to find a surface that interpolates or approximates a finite set of points in a k -dimensional space R^k . Sometimes the scattered data obtained is noisy (for example when collected using a 3D depth range finder) and approximation is desirable. Sometimes the data obtained is fairly accurate (sampled from a given model or an object) and interpolation is desired. In other variations, the data is specified on a sphere or on a surface (such as an aeroplane wing). While all these problems are undoubtedly very interesting, in this work, we focus on the functional scattered data interpolation problem in two or three dimensions.

Solutions to the scattered data interpolation or approximation problem are equally varied. Typically, the researcher makes a-priori choice regarding the type of solutions. Popular choices include polynomial or rational parametric representations, algebraic or implicit representations, subdivision methods, radial basis methods, Shepard's techniques and a combination of some of these approaches. Although the choice of the type of solution

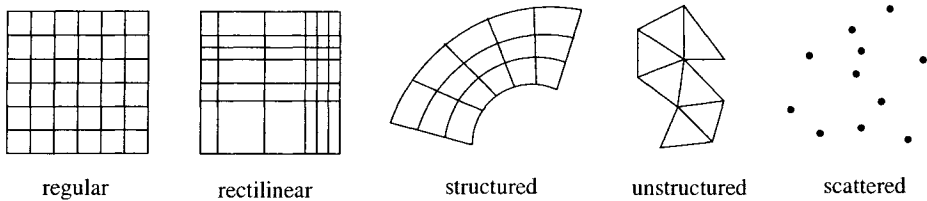


Figure 16.1. Different types of grids: regular, rectilinear, structured (each interior data point is connected to exactly four neighbors), unstructured (no restriction on the number of neighbors), scattered (no underlying grid)

sometimes may be guided by the application domain and the previous methodologies prevalent in the discipline, the distribution of the data points often play an important role in dictating the type of solutions pursued. One of the most important criteria is whether the scattered data is prescribed with an underlying grid or not. When data is uniformly distributed such as in the case of rectilinear or the structured grid (see Figure 16.1), one can employ special types of methods (such as bilinear or trilinear interpolation or more generally non-uniform rational B-splines (NURBS)) that may not be available for truly scattered data. Of course, one can always triangulate scattered data using Delaunay triangulation, for example, and employ methods that take advantages of the underlying triangular grid in two dimensions and tetrahedral grid in three dimensions. This approach has often been used in constructing polynomial or rational parametric and algebraic solutions. In cases where the underlying mesh consists of a mixture of triangles, quadrilaterals, and higher order polygons (or polytopes in higher dimensions), subdivision methods have been applied. Radial basis methods, in contrast, do not assume or need any underlying grid.

Although all of the techniques mentioned above are useful in diverse applications, we have chosen to concentrate on radial basis methods and some local methods in this work. Our choice is guided by the fact that this handbook covers rational parametric solutions and subdivision methods in other chapters. Besides, some exciting recent developments in radial basis methods have made these techniques much more practical for very large data sets that was not possible only a few years ago. Also, we chose to focus on techniques that are likely to be useful in practical applications that often arise without any underlying grids.

16.2. RADIAL INTERPOLATION

We begin by presenting a brief introduction to the radial basis functions. A function $\phi(r_k)$, where $r_k = \sqrt{(x - x_k)^2 + (y - y_k)^2}$ is referred to as a *radial function*, because it depends only upon the Euclidean distance between the points (x, y) and (x_k, y_k) . The points (x_k, y_k) are referred to as *centers* or *knots*. In particular, the function $\phi(r_k)$ is radially symmetric around the center (x_k, y_k) . The solution to the scattered data interpolation problem is obtained by considering a linear combination of the translates of a suitably

chosen radial basis function. Sometimes a polynomial term is added to the solution, when ϕ_k is conditionally positive definite (defined later in this section), or in order to achieve polynomial precision. More formally, the solution to the interpolation problem is sought in the following form:

$$F(x, y) = \sum_{k=1}^N A_k \phi(r_k) + \sum_{l=1}^M B_l q_l(x, y), \tag{16.1}$$

where $q_l(x, y), l = 1, \dots, M$ is any basis for the space P_m of bivariate polynomials of degree less than m , and therefore $M = \frac{m(m+1)}{2}$. Notice that $m = 0, 1, 2$ corresponds to the case when no polynomial, constant function or a linear polynomial is added to the interpolant respectively. In order to satisfy the interpolation conditions, one poses the following system of N linear equations in N unknowns $A_k, k = 1, \dots, N$, when $m = 0$:

$$\sum_{k=1}^N A_k \phi(r_{ik}) = f_i, i = 1 \dots, N, \tag{16.2}$$

where $r_{ik} = \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2}$ is the Euclidean distance between the points (x_i, y_i) and (x_k, y_k) . When $m \neq 0$, a slightly modified system of $N + M$ linear equations in $N + M$ unknowns $A_k, k = 1, \dots, N$ and $B_l, l = 1, \dots, M$ is formulated as follows:

$$\begin{aligned} \sum_{k=1}^N A_k \phi(r_{ik}) + \sum_{l=1}^M B_l q_l(x_i, y_i) &= f_i, i = 1, \dots, N, \\ \sum_{k=1}^N A_k q_l(x_k, y_k) &= 0, l = 1, \dots, M. \end{aligned} \tag{16.3}$$

In addition, throughout this section we shall assume the following mild geometric condition on the location of scattered data points:

$$p(x, y) \in P_m, p(x_i, y_i) = 0, i = 1, \dots, N \Rightarrow p \equiv 0. \tag{16.4}$$

Notice that this geometric condition is vacuous for $m = 0, 1$. For $m = 2$, this condition states that all the scattered data points do not lie on a straight line. Assuming that a solution to the system of equations (16.2) or (16.3) exist, the radial basis interpolant is then given by Equation (16.1).

Figure 16.2 presents some examples of radial basis functions with global support, that is, these functions never vanish on any interval. In these examples, h is a parameter. For good choices of this parameter, we refer the reader to [33]. The property of global support poses one of the major practical difficulties in computing and evaluating these radial basis functions. One of the main focuses of this chapter is to present recent developments that address these difficulties and suggest methods of overcoming them. In another approach, radial basis functions with compact support were introduced by several researchers including Schaback, Wu and Wendland [38,35,36,41,37], and applied to scattered data problems by Floater, Iske and others [17,24,18,26]. However, radial basis functions with compact support seem to exhibit inferior convergence properties in comparison to radial basis

Hardy's multiquadrics	$\phi(r_k) = \sqrt{r_k^2 + h^2}$
Thin plate splines	$\phi(r_k) = r_k^2 \log r_k$
Inverse multiquadrics	$\phi(r_k) = \frac{1}{\sqrt{r_k^2 + h^2}}$
linear distance function	$\phi(r_k) = r_k$
cubic distance function	$\phi(r_k) = r_k^3$
Gaussian radial basis functions	$\phi(r_k) = e^{-h^2 r_k^2}$
Shifted thin plate splines	$\phi(r_k) = (r_k^2 + h^2) \log(r_k^2 + h^2)^{\frac{1}{2}}$

Figure 16.2. Some radial basis functions with global support

functions with global support [13]. Even in the case of radial basis functions with global support, radial basis functions that grow with distances such as multiquadrics and thin plate splines seem to provide superior solutions in many practical applications in comparison to those that decay with distances such as inverse multiquadrics and Gaussian radial basis functions. Therefore, in this work, we focus mostly on multiquadrics and thin plate splines. Surprising though it may seem, these radial basis functions, although not absolute or square integrable, yield very good convergence. The main question, of course, is how to compute and evaluate interpolants using these radial basis functions. Before we turn to this question, we briefly describe some existence and uniqueness properties of these interpolants.

16.2.1. Existence and uniqueness

To describe the existence of radial basis interpolants, we begin with a few definitions. Given a radial function $\phi(r_k)$ and N scattered data points, consider the $N \times N$ square symmetric matrix $A = (a_{ij})$, where $a_{ij} = \phi(r_{ij})$. The radial function $\phi(r_k)$ is said to be positive definite iff $v^t A v \geq 0$ for all $v \in R^n$. The radial function $\phi(r_k)$ is said to be strictly positive definite if in addition, $v^t A v > 0$ whenever $v \neq 0$. If a radial basis function is strictly positive definite, then the matrix A is invertible. This is exactly what is needed in order to solve the system of linear equations (16.2) and guarantee the existence of an interpolant.

However, as mentioned before, for some radial basis functions such as thin plate splines, the matrix A is not always invertible and addition of at least a linear polynomial term to the interpolant is required. To describe these existence results in somewhat greater generality, the notion of conditionally positive definiteness is introduced. Let P_m denote the space of polynomials of degree less than m . Consider the collection V of vectors $v = (v_1, \dots, v_N)$ in R^N that satisfy $\sum_{i=1}^N v_i q(x_i) = 0$ for any $q \in P_m$. The radial function $\phi(r)$ is said to be conditionally positive definite (cpd) of order m iff $v^t A v \geq 0$ for all $v \in V$. The radial function $\phi(r)$ is said to be conditionally strictly positive definite (cpspd) of order m if in addition, $v^t A v > 0$ whenever $v \neq 0$. It can be proved with a little effort that the the system of equations 16.3 is uniquely solvable if the radial basis function is conditionally strictly positive definite of order m and the scattered data points satisfy the geometric condition (16.4).

Micchelli provided the following characterization for conditionally positive definite functions and derived the following important result:

Theorem: A function $f(t)$ is conditionally positive definite of order m in R^d for $d \geq 1$, if and only if $(-1)^j \frac{d^j}{dt^j} f(\sqrt{t}) \geq 0$, $t > 0$, $j \geq m$. If in addition $\frac{d^m}{dt^m} f(\sqrt{t}) \neq \text{constant}$, then $f(t)$ is conditionally strictly positive definite of order m .

It is now easy to verify that multiquadrics ($m \geq 1$), inverse multiquadrics ($m \geq 0$), thin plate splines ($m \geq 2$), linear distance function ($m \geq 1$), cubic power of the distance function ($m \geq 2$), shifted thin plate splines ($m \geq 2$), and Gaussians ($m \geq 0$) are cspd of order m up to a constant multiple, that is, either these functions or their negatives are cspd of order m . Some of these results derived from the theorem above can be strengthened further. In particular, the scattered data interpolation problem is solvable with $m = 0$ and $f(t)$ cspd of order 1, whenever $f(t) < 0$ for $t > 0$. This result guarantees the solvability of interpolation problem for multiquadrics without any addition of a constant or a polynomial term. Since so many choices of m are available, what is an appropriate m to choose while using these interpolants? We refer the reader to our previous survey [28] where we have addressed this question along with a number of other issues such as the choice of parameters. We also refer the reader to several articles by Buhmann [13,12,11] on radial basis functions. Here, it will suffice to say that in practice, multiquadrics and inverse multiquadrics are implemented without any polynomial term at all or at most with addition of a constant, while the thin plate splines require and are usually implemented with addition of a linear polynomial term.

16.2.2. Computation of the interpolant

The computational cost of direct methods for solving the scattered data interpolation problem using radial basis functions with global support is $O(N^3)$, and the storage requirements are $O(N^2)$. This is prohibitively expensive even with the fastest workstations currently available. This difficulty has restricted the use of radial basis functions to at most a few thousand centers till recently. However, recent developments may allow fitting data sets up to 5 million data points in 2 dimensions and up to 250,000 points in three dimensions in less than 10 seconds.

There are several approaches that are being actively investigated to compute these interpolants. Here, we present two methods – domain decomposition methods and GMRES (Generalized Minimum Residual) iteration methods [27]. Of these two methods, currently domain decomposition methods are most promising in terms of computing with very large data sets. GMRES iteration methods are less complex to implement and can deal with moderately sized data. Finally, multipole approaches are discussed for evaluating the radial interpolants. We now discuss each of these methods in somewhat greater detail.

Lagrangian and domain decomposition approach

Domain decomposition technique was proposed by Beatson and Powell and further developed by Beatson, Goodsell and Powell for thin plate splines in two dimensions [10,5]. A proof of convergence for thin plate splines and in fact, for conditionally positive definite radially symmetric functions including multiquadrics, inverse multiquadrics, and Gaussians, for two and higher dimensions was presented by Faul and Powell [16]. Further improvements in the domain decomposition method and numerical results are presented

by Beatson, Light and Billings [7].

The domain decomposition method divides the main problem into several small to medium size problems on smaller domains. Lagrange functions are employed on the small and medium size problems. These solutions are then combined to generate an initial approximation to the main problem. An iterative refinement of the initial approximation is performed by computing and improving the residuals.

We now explain each of these steps in somewhat greater detail. A decision is made to divide the main problem into problems of smaller size q , say 30 or 50. A uniformly distributed subset S of the scattered data point of roughly the same size is extracted on which the interpolation problem is uniquely solvable. The remaining points are ordered based on proximity. An initial seed point is chosen to start the ordering. q closest points are then added to this cluster. If there is a tie, it can be broken randomly. One can use alternative methods of structuring decomposing the domain. For example, Beatson, Light, and Billings [7] have used balanced nD-tree to subdivide space into rectangular boxes to guide the decomposition of the space. As we will soon see, Lagrangian interpolations will be performed on clusters of size q .

Now, there are roughly $\frac{N}{q}$ problems of size q . Let C denote the set of points belonging to one of these smaller subproblems or clusters. On these smaller subproblems, local Lagrange interpolation functions of the following form are used:

$$L_i(x) = \sum_{j \in C} \lambda \phi(r) + p_i(x).$$

Lagrange functions satisfy the following property: they are exactly 1 at one of the data points and 0 at the remaining $q - 1$ points, that is, $L_i(x_j) = \delta_{ij}$ for $i, j \in C$. This yields a system of linear equations. This computation is performed for each of the subproblems once and the results λ_{ij} are stored for each subproblem. This step is an $O(N)$ process.

The next step is to combine the solutions to these Lagrangian subproblems to generate an initial approximation to the main problem. The initial guess $s_0(x)$ starts with zero. For each point in the ordered set (that is for all points except for the points in the subset S), the successive approximant is built as follows:

$$s(x) \rightarrow s(x) + c_i(x)L_i(x),$$

where,

$$c_i(x) = \frac{1}{\lambda_{ii}} \sum_{j \in C} (f(x_j) - s(x_j)).$$

Please note that the $c_i(x)$ is the residual at each point. It can be established that the λ_{ii} is positive so that the division above does not pose any problems. The main work, therefore, in this algorithm is the computation of the residuals. In the final step of this sweep of the algorithm, the solution $\sigma(x)$ to the interpolation problem with centers at S with $\sigma(i) = f(i) - s(i)$ for $i \in S$, is added to complete the first initial approximation to the main problem. Let $s_1(x)$ denote the end result after the the first sweep of the algorithm.

Now, an iterative refinement is performed where the entire sweep of the algorithm takes place starting with the initial approximation $s_1(x)$. It is remarkable that not only that

this algorithm converges, it turns out that there is an increase in accuracy of one digit per sweep of the algorithm, that is, each sweep reduces the maximum error at each data point by a factor of ten, which indicates very fast convergence.

There are several algorithmic and implementation level details that can be incorporated to make this algorithm efficient. For details, we refer the reader to [7]. Numerical results with thin plate splines in two and three dimensions indicate that one can obtain approximately $O(N \log N)$ complexity with this algorithm. This allows computation of the interpolant with up to 5 million data points in two dimensions, and up to 250,000 points in three dimensions.

GMRES iteration

GMRES (Generalized Minimal Residual) algorithm for solving nonsymmetric linear systems was introduced by Saad and Schultz [34]. Implementation of this method using Householder transformations has been discussed by Walker [40]. Beatson, Cherrie and Mouat [4] applied GMRES iteration technique to solve the scattered data interpolation problem using radial basis functions. GMRES iterative methods reduce the computational cost of constructing the interpolant to $O(N)$ storage and $O(N \log N)$ operations. The implementation of this method is simpler than the domain decomposition method. Numerical results have been reported using the GMRES method with 10,000 points in two dimensions for thin plate splines and multiquadrics with satisfactory results [4].

As in the domain decomposition method, the GMRES method begins by solving N smaller subsystems of linear equations. Three different strategies have been proposed for constructing these smaller subproblems. These strategies are based on (i) purely local centers, (ii) local centers and special points, and (iii) decay elements. In the purely local centers approach, Lagrange interpolation functions are constructed for the closest points as in the case of domain decomposition method described before. In the local centers and special points approach, some special points uniformly distributed and far away from the local points are added to construct the Lagrange functions in the hope that this will force the deviation of the Lagrange functions to be small near these far away regions. In the decay element approach, the objective is to construct approximate Lagrange functions in the neighborhood of the point and decay rapidly as the distance increases. This is achieved by solving the constrained least squares problem $L_i(x_j) = \delta_{ij}$ subject to $L_j(x) = O(|x|^{-3})$ as $|x| \rightarrow \infty$. This decay condition is actually equivalent to a homogeneous system of linear constraints. However, decay elements cannot be used exclusively as a basis since they do not span the whole space. Therefore, some non-decay elements are used as well. For some suitable tolerance μ (say, 0.5), the decay element L_{ij} is used if and only if $\sum_{i \in C} |L_i(x_j) - \delta_{ij}| < \mu$. Most points in the interior satisfy this criterion. If this condition is not satisfied, then the second method of local centers and special points is used to build the interpolant.

The GMRES method differs from the domain decomposition method in how the results of the smaller problems are combined to create an approximate solution to the main problem. Let L_j be the local interpolant associated with the point x_j as described in the previous paragraph. Now, we consider the system of linear equations:

$$\psi\mu = f,$$

where $\psi_{ij} = \psi_j(x_i)$. This system of linear equations is then solved by the standard GMRES technique.

16.2.3. Evaluation

Multipole methods use analytic expansions of the underlying radial functions for large argument, referred to as far-field expansions or Laurent series expansions. The salient ideas for this approach were used by Greengard and Rokhlin [23] to solve numerical integral equations and were applied to the scattered data interpolation problem using radial basis functions by Beatson and Newsam [8].

The overall idea for the evaluation of the interpolant is to break down the evaluation into two parts – contributions from the near terms and contributions from the far term. In order to define what is near and what is far, the data is structured hierarchically, for example, using quadtrees. Contributions from the near terms are computed exactly and explicitly. The main challenge is to compute the contributions from the far terms. Contribution from a far term can be approximated well away from the origin by a truncated Laurent series expansion because radial basis functions are analytic at the origin, even when made multivariate through composition with Euclidean norms. However, summing up contributions from each far term using Laurent series expansion is still very costly. The key idea is to form clusters of far terms and combine the Laurent series expansions of each of these clusters into a single Taylor series expansion at the desired point of computation. This method reduces the computation cost to $O(1)$ for each approximate evaluation of the radial interpolant. The accuracy of the computation can be prescribed by the user and can be matched by using appropriate truncation of the series expansions.

Several improvements and extensions of the above algorithm have been proposed in the last decade [6,9,3]. In particular, the algorithm described above requires much mathematical analysis of appropriate series expansions and corresponding translator operators for every new radial basis function. Recently, Beatson, Newsam, and Chacko have introduced moment-based methods for evaluating radial basis functions that still take $O(1)$ operations for each single evaluation of the interpolant but in addition, require much simpler implementation to accommodate additional radial basis functions. This is achieved by replacing the far field expansions using Laurent series expansions with computations involving moments of the data.

We now describe each of the steps mentioned above in somewhat greater detail. We first describe the four steps needed to set up the computation. Then we describe the two steps required for the evaluation. In the first step of the set up, the space is subdivided in a hierarchical manner. The space can be enclosed within a given square or a volume. One can then use uniform quadtree subdivisions or adaptive subdivisions depending upon the distribution of the data set. Each subdivided region is referred to as a cell or a panel. Typically, the subdivision is carried down to $\log N$ levels. The centers are associated with the cells that they lie in. Two cells are considered near iff they are adjacent to each other. In the next step, far points are to be grouped together. To this purpose, one can define the notion of a distance between two cells based on the hierarchical subdivision of the space using the number of edges in the tree to be traversed to get from one cell to another. Then all the cells that are equidistant from the given cell are grouped together. In the third step, in the original algorithm, Laurent series expansion is computed for each

point and translation is used so that these expansions can be reused for other centers. Then these computations are done for each cell starting from the most refined levels or the bottom of the tree and traverse upwards towards coarser levels forming analogous expressions by combining the computations at the refined levels. In the moment-based algorithms, the moments of the coefficients (used in the solution to the scattered data interpolation problem) around the cell centers are computed from the most-refined levels to the coarser levels of the tree by traversing the tree as before. In the fourth step of the original algorithm, the hierarchical tree is traversed downwards starting from the root and for each cell, Taylor expansion is computed by combining the Laurent series expansions of the whole far field. In the moment-based algorithms, the tree is also traversed downwards as above, however, polynomial approximations are now formed by combining moments and certain approximations to the radial basis functions.

In the evaluation phase, first we identify the leaf node cell (at the bottom of the tree) containing the point where the evaluation is needed. In the second step, the contribution by the near points is computed exactly and precisely. To this we add the contribution by the far points, which is computed by error-driven truncation of the Taylor series expansion of the far fields associated with this cell in the original algorithm or by appropriate evaluation of the polynomial approximations in the moment-based algorithm.

In the moment-based algorithms, real Fast Fourier Transforms (FFTs) are used to compute the required moments. Then the shifted moments corresponding to shifted centers can be computed as convolutions of moments. Computations needed in the approximations of the radial basis functions in the moment-based methods can be greatly reduced by symmetry considerations. Details and some numerical results involving 32000 points in 2D can be found in [3].

16.2.4. Applications

Radial basis functions have been applied in a wide variety of disciplines including bathymetry (ocean depth measurement), topography (altitude measurements), hydrology (rainfall interpolation), surveying, mapping, geophysics, and geology [25]. More recent applications include image warping [42,1], medical imaging [14], and 3D object representations and reconstructions [15].

Here we discuss some of these applications briefly. In the image warping application by Arad et al. [1], radial basis functions are used to approximate warping of 2D facial expressions. Some key features of the face are identified by a user as pixels or points, referred to as anchor points, on an image. The authors have developed a system for identifying important facial features on eyes and mouth using a technique called generalized symmetry. In the image warping application, an added advantage is that the anchor points need not be specified very accurately. When facial expressions undergo change, new locations of these anchor points are also determined using the feature finding algorithm. This sets up a mapping from R^2 to R^2 for anchor points. The objective is to find realistic mapping for the whole image. This problem can be decomposed into two independent scattered data interpolation problem from R^2 to R . It is well known that if there are only 3 anchor points, one can find an affine mapping from one image to another. In practice, the authors claim that a small number of anchor points yield fairly good results. In examples using the warping of images of Mona Lisa and Ronald Reagan, the authors have used from 6

to 14 anchor points. Thin plate splines and Gaussian radial basis functions are used to solve the image warping problem. Thin plate splines have the property that they minimize the bending energy and therefore seem to be appropriate in warping applications. Gaussian radial basis functions can be used to provide a locality condition by judiciously choosing the scalar σ that affects the region of influence around the anchor points. In a variation to the interpolation problem, least squares minimization of the sum of the Euclidean distances between the anchor points is also considered. This variation introduces a scalar factor λ that determines whether the solutions will be a radial solution (when $\lambda = 0$) or will be an affine mapping minimizing the least squares distance (when $\lambda = \infty$) or somewhere in between. Several results using different values of σ and λ are shown to establish the effectiveness of this approach.

In [14], Carr et al. have used radial basis functions to design cranial implants for the repair of defects, usually holes, in the skull. When a defect is large (≥ 25 sq. cm.), implants are fabricated presurgically. Prefabrication requires an accurate model of the defect area to ensure that a good fit is achieved. Depth maps of the skull's surface obtained from X-ray CT data using ray-tracing techniques have been used to construct models of cranial defects. The depth map is a mapping from a subset of R^2 to R . Due to the presence of defects, the data sets will have holes in the domain. These holes need to be filled. Use of radial basis functions is appealing because no regular underlying grid is available. In this application, a combination of linear radial basis functions and thin plate splines are used. Several examples and results using 300 to 700 points are discussed. Solutions obtained by radial basis functions are compared with the original skull (by artificially introducing holes or defects in the data). Examples for repairing large defects (150 sq. cm.) in convex regions as well as repairing holes close to the orbital margin and other regions of high curvature are also presented.

In the third application, radial basis functions have been used to warp aerial photographs to orthomaps using thin plate splines [42]. Orthomaps, which typically have a pixel resolution of 1 meter on the ground, are produced by a complex and costly process involving acquisition of aerial photographs and ground control data, aerial triangulation, and sophisticated processing of raw images in association with a digital elevation model. However, orthomaps undergo change due to changing characteristics of the region both due to natural (seasonal, wildfires) and man-made changes (new roads, buildings etc.) . To update an orthomap, new aerial photographs are taken that need to be registered or warped onto the orthomap because geometric distortions arise in the photographs as a result of the finite height of the aerial camera and the relief of the terrain being imaged. In this application, six image pairs were used for the British Columbia region. This application is very similar to the first one discussed above except that in this case approximately 200 to 300 feature points were selected either manually or using an automated or semi-automated procedure. In this application, aerial photographs were accurate to within 10m on the ground and to within 5m in altitude. Using cross-correlational analysis, the authors concluded that the warped features were corrected with great effectiveness being accurate 50% of the time to within 1.5m, and 90% of the time to within 5m.

16.3. OTHER LOCAL METHODS

Local methods are attractive for very large data sets because the interpolation or approximation at any point can be achieved by considering only a local subset of the data. For data sets of up to at least a few hundred points, global methods such as thin plate splines and multiquadrics are easily applied. Computational experience seems to indicate that for many problems, using very few local points (perhaps fewer than 50 to 100 points) yields surfaces for which the local variations are over-emphasized. This results in what might be described as a somewhat "lumpy" surface, so current recommendation is to take care not to let the surface definition be too locally defined.

Many local methods can be characterized as weighted sums of local approximations, $L_k(x, y)$, where the weights, $W_k(x, y)$ form a partition of unity. Interpolation properties of the local approximations are preserved in the function $F(x, y) = \sum_{k=1}^N W_k(x, y)L_k(x, y)$, provided that each $L_k(x, y)$ takes on the value f_j at each (x_j, y_j) for which $W_k(x_j, y_j) \neq 0$. In order for the overall method to be local, it is necessary that the weight functions be local, that is, nonzero over a limited region, or at a limited number of the data points.

We briefly review Shepard methods. These arise from the simple interpolation formula due to Shepard [39] of the form $F(x, y) = \sum_{k=1}^N \frac{f_k}{d_k^2(x, y)} / \sum_{k=1}^N \frac{1}{d_k^2(x, y)}$ where $d_k^2(x, y) = (x - x_k)^2 + (y - y_k)^2$ is the square of the distance from (x, y) to (x_k, y_k) and μ is a parameter, often taken to be 2. This is a global method due to the global weighting functions for the data values. It has well-known shortcomings, such as flat spots at all data points. Many of the shortcomings are overcome by what is called the local quadratic Shepard method, a version of which is available as a Fortran program in the TOMS series, Algorithm 660 [31]. A trivariate version is available as Algorithm 661 [32]. We view the method as a weighted sum of local approximations, where for Shepard's method, the local approximation $L_k(x, y)$ is f_k , and the weight function $W_k(x, y)$ is then clear. We now replace the weight function with another that has compact support and the local approximation with a quadratic function that interpolates the value f_k at (x_k, y_k) . The choice for the weight function used by Renka was suggested by Franke and Little [2] and is of the form $W_k(x, y) = \frac{(R_k - d_k(x, y))_+^2}{(R_k d_k(x, y))^2}$, where $u_+ = 0$ if $u < 0$ and $u_+ = u$ if $u \geq 0$. This weight function behaves essentially like $d_k(x, y)$ for (x, y) points near (x_k, y_k) , while becoming zero at distance R_k . The local approximation $L_k(x, y)$ is a quadratic function taking on the value f_k at the point (x_k, y_k) with the other coefficients being determined by a weighted least squares approximation using the data values at a given number of data points near to (x_k, y_k) . The weights for the approximation have the same form as the weight function $W_k(x, y)$ above, but using a different value for the radius at which the weight function becomes zero. Methods such as this can be used easily, with reasonable results. The primary disadvantage for large data sets is that a considerable amount of preprocessing is needed to determine closest points and calculate the local approximations. On the other hand, if approximations are only required in some localized area, the preprocessing may not be burdensome.

Another version of the weighted local approximation was given by Franke [19]. In this algorithm the search for nearby points and computation of local approximations is

enhanced in several ways. The plane is subdivided into a finite rectangular grid, with the goal that each subrectangle contains approximately a specified number of points. The success of this depends on the locations of the data points, and the points may necessarily be poorly apportioned in particular cases. The weight functions are taken to be Hermite bicubic functions with value one and slope zero at the intersection of a vertical and horizontal grid line, and value zero and slope zero at the exterior boundary of the four adjacent subrectangles. This set of functions over the grid points forms a partition of unity. To ensure interpolation it is necessary that the local interpolation function, $W_k(x, y)$ take on the specified value at each of the data points within the four subrectangles adjacent to the corresponding grid point. This approximation is taken to be a thin plate spline, with some accommodation necessary for fewer than the required number of points (that is, three not on a line) within the four subrectangles. While the original paper suggested a fairly small number of points for each set of four subrectangles, later experience indicates the number should be as large as several hundred. The use of rectangular regions greatly simplifies the problem of finding the nonzero terms in the summation since this reduces to a pair of one-dimensional searches to find the subrectangle in which the evaluation point lies. Again, the amount of preprocessing required is substantial. As with the quadratic Shepard method, this can be reduced a great amount if the approximation is to be carried out only in a localized area.

Approximations of the above type can be implemented where the local approximation is determined by least squares, for example. Interpolation will not be maintained, but that may not be an important consideration. One candidate for local approximation in this case is a local least squares multiquadric function. Such functions have been studied by Franke and coauthors [21,22,20] and the method appears to be useful using only a small number of knots (centers for the multiquadric basis functions). However, the preprocessing required is greater per local approximation, but this would be offset by a smaller number of local approximations being required. Further, when many evaluations require that it be done rapidly, substantial preprocessing may be acceptable.

Finally, we discuss triangle based methods for solution of the scattered data interpolation problem. The general method we discuss is sometimes described as a finite element method since the approximation is basically a piecewise function defined over a triangulation. We assume that a smooth (continuous first derivatives) interpolation function is to be constructed. For large data sets, the overhead involved in the construction of the triangulation and storage of sufficient information about it may be excessive. This may be overcome in at least two ways. First, we consider methods that use the data points as the vertices of the Delaunay triangulation, and which estimate gradients at the data points from a local subset of the data. Because the Delaunay triangulation is local (made clear by the circle property of the triangulation), it is necessary to process only a subset of the data in the same locality as the evaluation points. Thus the problem can be handled by processing overlapping subsets of the data (some care must be taken to ensure the overlap is sufficient to obtain the same results as for the global problem). Algorithms for performing the triangulation are quite efficient both in terms of time and required computational resources [30]. Once the triangulation is completed (whether only a local part of it, or the global one), the gradients at the vertices need to be estimated. This can be done in a variety of ways, the most successful seeming to be those that fit

a polynomial approximation to local data points in ways very similar to that described above for the quadratic Shepard method, except here the approximations are only used to obtain gradient estimates. Alternatively, the subset selection method for gradient approximation may be based on the neighbors in the triangulation, and perhaps neighbors of neighbors. This has a certain cohesiveness and also has the advantage of ease of finding the neighbors through the triangulation. On the debit side, points relatively far away could then be involved, although generally only near the boundary of the convex hull of the data points.

When the number of data points is excessive, it may be desirable to choose the vertices of the triangulation as a subset of the data points, or to use as vertices other points not directly related to the data points. A procedure for elimination of vertices from a piecewise polynomial approximation of a function was given by Le Méhauté and LaChance [29]. The algorithm eliminates unneeded vertices by calculating a measure of significance for each vertex point, and eliminating those with low significance, subject to maintaining accuracy to within a specified tolerance. The amount of computation for a significant reduction in the number of vertices may be large, but again, may be worthwhile in certain circumstances.

16.4. CONCLUSIONS

Although scattered data interpolation and approximation is a very old problem, new research and recent developments particularly in the area of radial basis functions and subdivision techniques have infused fresh energy into this area. In this work, we have presented a high level view of some of the recent developments that carry great promise for solving practical problems using radial basis functions. This research is still in a state of flux, and many algorithmic and implementation level details need to be stabilized to carve out interpolation and approximation algorithms that can be used by a wider audience. Furthermore, great theoretical advances in radial basis functions have remained inaccessible to the majority of practitioners due to highly sophisticated mathematical presentation of these recent developments. We believe that radial basis function methods are richly deserving of practical implementations for large data sets. We hope that this work provides some pointers to practitioners that will encourage such endeavors.

REFERENCES

1. N. Arad, N. Dyn, D. Reisfeld, and Y. Yeshurun. Image warping by radial basis functions: application to facial expressions. *CVGIP: Graphical Models and Image Processing*, 56(2):161–172, March 1994.
2. R.E. Barnhill. Representation and approximation of surfaces. In J. R. Rice, editor, *Mathematical Software III*, pages 69–120. Academic Press, New York, 1977.
3. R.K. Beatson and E. Chacko. Fast evaluation of radial basis functions: a multivariate momentary evaluation scheme. In C. Rabut, A. Cohen and L.L. Schumaker, editors, *Curve and Surface Fitting*, pages 37–46. Vanderbilt University Press, Nashville, 2000.
4. R.K. Beatson, J.B. Cherrie, and C.T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Advances in Computational Mathematics*, 11(2):253–270, 1999.

5. R.K. Beatson, G. Goodsell, and M.J.D. Powell. On multigrid techniques for thin plate spline interpolation in two dimensions. *Lectures in Applied Mathematics*, 32:77–97, 1996.
6. R.K. Beatson and W.A. Light. Fast evaluation of radial basis functions: methods for two-dimensional polyharmonic splines. *IMA Journal of Numerical Analysis*, 17(3):343–372, 1997.
7. R.K. Beatson, W.A. Light, and S. Billings. Fast solution of the radial basis function interpolation equations: domain decomposition methods. *SIAM Journal on Scientific Computing*, 22(5):1717–1740, 2000.
8. R.K. Beatson and G.N. Newsam. Fast evaluation of radial basis functions. *Computers and Mathematics with Applications*, 24(12):7–19, 1992.
9. R.K. Beatson and G.N. Newsam. Fast evaluation of radial basis functions: moment-based methods. *SIAM Journal on Scientific Computing*, 19(5):1428–1449, 1998.
10. R.K. Beatson and M.J.D. Powell. An iterative method for thin plate spline interpolation that employs approximations to lagrange functions. In D.F. Griffiths and G.A. Watson, editors, *Numerical Analysis*, pages 17–39. Longman Scientific & Technical (Burnt Mill), 1993.
11. M.D. Buhmann. A new class of radial basis functions with compact support. Tech Report Number 164, Universitat Dortmund, Germany, 1998.
12. M.D. Buhmann. Approximation and interpolation with radial functions. Tech Report Number 169, Universitat Dortmund, Germany, 1999.
13. M.D. Buhmann. Radial basis functions. *Acta Numerica*, pages 1–38, 2000.
14. J.C. Carr, W. Fright, and R.K. Beatson. Surface interpolation with radial basis functions for medical imaging. *IEEE Transactions on Medical Imaging*, 16(1):96–107, February 1997.
15. J.C. Carr, T. Mitchell, R. Beatson, J.B. Cherrie, W.R. Fright, B.C. McCallum, and T.R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In Eugene Fiume, editor, *SIGGRAPH Proceedings*, pages 67–76. ACM Press, 2001.
16. A.C. Faul and M.J.D. Powell. Proof of convergence of an iterative technique for thin plate spline interpolation in two dimensions. *Advances in Computational Mathematics*, 11(2):183–192, 1999.
17. M.S. Floater and A. Iske. Thinning, inserting, and swapping scattered data. In A. Le Mehaute, C. Rabut, and L.L. Schumaker, editors, *Surface Fitting and Multiresolution Methods*, pages 139–144. Vanderbilt University Press. Nashville, TN, 1996.
18. M.S. Floater and A. Iske. Thinning algorithms for scattered data interpolation. *BIT*, 38(4):705–720, 1998.
19. R. Franke. Smooth interpolation of scattered data by local thin plate splines. *Computers and Mathematics with Applications*, 8:273–281, 1982.
20. R. Franke and H. Hagen. Least squares surface fitting using multiquadrics and parametric domain distortion. *Computer Aided Geometric Design*, 16:177–196, 1999.
21. R. Franke, H. Hagen, and G.M. Nielson. Least squares surface approximation to scattered data using multiquadric functions. *Advances in Computational Mathematics*, 2:81–99, 1994.
22. R. Franke, H. Hagen, and G.M. Nielson. Repeated knots in least squares multiquadric

- functions. *Computing Supplem.*, 10:177–185, 1995.
23. L.L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.
 24. T. Gutzmer and A. Iske. Detection of discontinuities in scattered data approximation. *Numerical Algorithms*, 16(2):155–170, 1997.
 25. R.L. Hardy. Theory and applications of the multiquadric-biharmonic method. *Computers and Mathematics with Applications*, 19:163–208, 1990.
 26. A. Iske. Reconstruction of smooth signals from irregular samples by using radial basis function approximation. In *Proceedings of the 1999 International Workshop on Sampling Theory and Applications*, pages 82–87. The Norwegian University of Science and Technology, Trondheim, 1999.
 27. C.T. Kelley. *Iterative methods for linear and nonlinear equations*. SIAM, 1995.
 28. S.K. Lodha and R. Franke. Scattered data techniques for surfaces. In G. Nielson, H. Hagen, and F. Post, editors, *Scientific Visualization, Dagstuhl '97*, pages 181–222. IEEE Computer Society Press, 1999.
 29. A.J.Y. Le Méhauté and Y. LaChance. A knot removal strategy for scattered data. In T. Lyche and L.L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 419–426. Academic Press, 1989.
 30. R.J. Renka. Algorithm 624: Triangulation and interpolation of arbitrarily distributed points in the plane. *ACM Transactions on Mathematical Software*, 10:440–442, 1984.
 31. R.J. Renka. Qshep2d: Fortran routines implementing the quadratic shepard method for bivariate interpolation of scattered data. *ACM Transactions on Mathematical Software*, 14:149–150, 1988.
 32. R.J. Renka. Qshep3d: Fortran routines implementing the quadratic shepard method for trivariate interpolation of scattered data. *ACM Transactions on Mathematical Software*, 14:151–152, 1988.
 33. S. Rippa. An algorithm for selecting a good value for the parameter c in radial basis function interpolation. *Advances in Computational Mathematics*, 11(2):193–210, 1999.
 34. Y. Saad and M.H. Schultz. GMRES:a generalized minimal residual algorithm for solving nonsymmetric linear equations. *SIAM Journal of Scientific and Statistical Computing*, 7:856–869, 1986.
 35. R. Schaback. Remarks on meshless local construction of surfaces. In *Mathematics of Surfaces X*. Oxford University Press. To appear.
 36. R. Schaback and H. Wendland. Adaptive greedy techniques for approximate solutions of large RBF systems. *Numerical Algorithms*. To appear.
 37. R. Schaback and H. Wendland. Characterization and construction of radial basis functions. In *Eilat Proceedings*. Cambridge University Press. To appear.
 38. R. Schaback and H. Wendland. Numerical techniques based on radial basis functions. In A. Cohen, C. Rabut, and L. L. Schumaker, editors, *Curve and Surface Fitting*, pages 1–16. Vanderbilt University Press, Nashville, TN, 1999.
 39. D. Shepard. A two dimensional interpolation function for irregular spaced data. *Proceedings of 23rd ACM National Conference*, pages 517–524, 1968.
 40. H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM Journal of Scientific and Statistical Computing*, 9:152–163, 1988.

41. H. Wendland. Local polynomial reproduction and moving least squares approximation. *IMA Journal of Numerical Analysis*, 21:285–300, 2001.
42. C. A. Zala and I. Barrodale. Warping aerial photographs to orthomaps using thin plate splines. *Advances in Computational Mathematics*, 11(2):211–227, 1999.

Chapter 17

Pythagorean-Hodograph Curves

Rida T. Farouki

17.1. PREAMBLE

With the advent of a new millenium, it seems appropriate to begin with a brief historical perspective on a topic that entails a remarkable confluence of ideas spanning nearly 4000 years of geometry and algebra. Figure 17.1 shows cuneiform tablet No. 322 in the Plimpton Collection of the Rare Book & Manuscript Library, Columbia University. This compilation of sexagesimal numbers, composed in the Old Babylonian Period (~ 1900 to 1600 BC), was discovered in the 1920s and subsequently deciphered by Neugebauer and Sachs [54] in 1945. Far from being a mere financial or commercial record, the tablet reveals profound knowledge [8] of the fundamental characterization

$$a = u^2 - v^2, \quad b = 2uv, \quad c = u^2 + v^2 \quad (17.1)$$

for integer solutions to the “Pythagorean” equation, $a^2 + b^2 = c^2$. After a thousand years, Mesopotamian supremacy in algebra was superceded by the ascent of Greek geometry — Pythagoras of Samos (~ 560 – 480 BC) is credited with the first proof that this equation governs the sides of all right triangles, and is thus fundamental in distance measurement. Unfortunately, geometry fell into a prolonged stagnation after the Greeks, until Descartes’ *La géométrie* of 1637 proposed a propitious marriage of geometry and algebra through the use of coordinates. Although this opened the fascinating realm of higher-order curves to mathematical scrutiny, the first steps were hesitant: Descartes blundered by categorically rejecting the idea of “rectification” (i.e., arc-length measurement) of curves.

The calculus of Leibniz and Newton resolved the existential, but not the computational, aspects of arc length measurement. Applying the Pythagorean theorem to an infinitesimal curve segment allows us to express the length of a (sufficiently smooth) parametric curve $\mathbf{r}(t) = (x(t), y(t))$ as the integral

$$s(t) = \int_0^t \sqrt{x'^2(u) + y'^2(u)} \, du, \quad (17.2)$$



Figure 17.1. Plimpton 322, the cuneiform Pythagorean triples tablet from Mesopotamia.

but this does not, in general, admit closed-form evaluation for “simple” (rational) curves. Ideally the curve parameter *is* the arc length, $s(t) \equiv t$, an assumption that helps elucidate the intrinsic geometry of curves. It is a matter of some subtlety, however, that this ideal is unattainable by any rational curve other than a straight line [31].

Although we must relinquish the hope of rational arc-length parameterization, we can nevertheless ensure exact mensurability of the arc-length function (17.2) by incorporating a special algebraic structure in the curves $\mathbf{r}(t) = (x(t), y(t))$ under consideration, based on the recognition that Pythagorean triples of *polynomials* admit the same characterization (17.1) as triples of *integers* [46]. Thus, by constructing curves with hodograph (derivative) components $x'(t)$, $y'(t)$ that are elements of Pythagorean triples, we ensure reduction of the integral (17.2) to a polynomial in t . This is the key concept motivating the introduction [30] of the Pythagorean-hodograph (PH) curves, which offer many additional advantages (rational offsets, superior shape properties, real-time interpolators, etc.) described below.

Apart from these practical advantages, the investigation of PH curves is of considerable intellectual appeal for the wealth of mathematical ideas it entails, including the geometry of complex numbers, inaugurated by Wessel, Argand, and Gauss; Hamilton’s quaternions and the geometrical algebras of Clifford; medial axis transforms and the Minkowski metric of special relativity; projective geometry and dual representations; the cyclographic map and Laguerre geometry; and connections with classical geometrical optics.

17.2. POLYNOMIAL PH CURVES

The distinguishing feature of a polynomial PH curve $\mathbf{r}(t)$ is that the components of its hodograph $\mathbf{r}'(t)$ satisfy a Pythagorean condition, i.e., the sum of their squares is equal to

the square of a polynomial $\sigma(t)$. The satisfaction of this condition entails rather different approaches in the context of planar and spatial curves, as described below.

17.2.1. Planar PH curves

The hodograph $\mathbf{r}'(t) = (x'(t), y'(t))$ of a planar PH curve must satisfy

$$x'^2(t) + y'^2(t) = \sigma^2(t), \tag{17.3}$$

where $\sigma(t)$ is a polynomial. Satisfying this condition is equivalent [30] to the requirement that, in terms of polynomials $u(t), v(t), h(t)$, the hodograph has the form

$$x'(t) = [u^2(t) - v^2(t)]h(t), \quad y'(t) = 2u(t)v(t)h(t), \tag{17.4}$$

and hence $\sigma(t) = [u^2(t) + v^2(t)]h(t)$. Taking $h(t) = 1$ and $\gcd(u, v) = 1$ gives a *primitive* Pythagorean hodograph, defining a regular PH curve (i.e., $\gcd(x', y') = 1$) of odd degree.

A direct consequence of (17.3) is that, for PH curves, the cumulative arc-length function (17.2) is just a *polynomial* (rather than an irreducible integral). Moreover, the offset curves at each distance d from $\mathbf{r}(t)$ — defined [23] by

$$\mathbf{r}_d(t) = \mathbf{r}(t) + d\mathbf{n}(t), \tag{17.5}$$

where $\mathbf{n}(t)$ is the unit normal to $\mathbf{r}(t)$ — admit exact *rational* representations, eliminating the need for approximation schemes that can be inaccurate, data intensive, or lacking in robustness — see [10,59] and references therein. The exact arc length and offset properties of PH curves are extremely useful in the context of CNC machining (see §17.4 below).

Taking constants for $u(t), v(t), h(t)$ reveals the trivial fact that straight lines are PH curves. The first non-trivial examples are cubics, defined by choosing $h(t) = 1$ and linear polynomials $u(t), v(t)$ in (17.4). PH cubics can be characterized geometrically in terms of their Bézier control points $\mathbf{p}_0, \dots, \mathbf{p}_3$. Namely, if $L_k = |\mathbf{p}_k - \mathbf{p}_{k-1}|$ are the lengths of the control-polygon legs, and θ_1, θ_2 are the angles at the interior points $\mathbf{p}_1, \mathbf{p}_2$, the conditions

$$L_2 = \sqrt{L_3L_1} \quad \text{and} \quad \theta_1 = \theta_2 \tag{17.6}$$

are sufficient and necessary for a PH cubic [30]. On closer scrutiny, the elegant simplicity of this characterization reveals a deeper truth: modulo rigid motions, scalings, and linear reparameterizations, PH cubics are all segments of a unique curve, *Tschirnhausen's cubic*. Since it cannot inflect, this curve is of limited value in design applications [25].

For shape flexibility similar to that of “ordinary” cubics, we must appeal to PH quintics, defined by taking $h(t) = 1$ and quadratic polynomials $u(t), v(t)$ in (17.4). The PH quintics can inflect and can interpolate arbitrary first-order Hermite data; they can also be used to construct C^2 splines interpolating a sequence of points (see §17.3). However, their control polygons do not admit simple geometrical characterizations [12], analogous to (17.6).

To ensure numerical stability [27,28] we always specify polynomials in Bernstein-Bézier form on $[0, 1]$. Choosing $h(t) = 1$ and degree- m polynomials $u(t)$ and $v(t)$ with Bernstein coefficients u_0, \dots, u_m and v_0, \dots, v_m in (17.4) defines a PH curve $\mathbf{r}(t)$ of degree $n = 2m+1$, whose parametric speed $\sigma(t) = |\mathbf{r}'(t)|$ is a polynomial of degree $n - 1$, specified [11] by the Bernstein coefficients

$$\sigma_k = \sum_{j=\max(0,k-m)}^{\min(m,k)} \frac{\binom{m}{j} \binom{m}{k-j}}{\binom{n-1}{k}} (u_j u_{k-j} + v_j v_{k-j}), \quad k = 0, \dots, n - 1.$$

Integrating $\sigma(t)$, we obtain the arc-length function as a polynomial $s(t)$ of degree n with Bernstein coefficients

$$s_0 = 0 \quad \text{and} \quad s_k = \frac{1}{n} \sum_{j=0}^{k-1} \sigma_j, \quad k = 1, \dots, n.$$

Note that $s(t)$ is monotone-increasing, since it is the integral of a non-negative polynomial $\sigma(t)$. Thus, although $s(t)$ does not possess a closed-form inverse [15], the parameter value t_* corresponding to any given arc length s_* can be computed to machine precision, as the unique real root of the equation $s(t_*) = s_*$, by means of a few Newton-Raphson iterations. This property is especially useful in the formulation of real-time CNC interpolators; see §17.4 below. The total arc length S for $t \in [0, 1]$ is simply $s_n = (\sigma_0 + \dots + \sigma_{n-1})/n$.

To represent the offsets (17.5) as rational curves, we introduce homogeneous coordinates (W, X, Y) and write the control points of the PH curve $\mathbf{r}(t)$ as $\mathbf{P}_k = (1, x_k, y_k)$ for $k = 0, \dots, n$. With $\Delta\mathbf{P}_k = \mathbf{P}_{k+1} - \mathbf{P}_k = (0, \Delta x_k, \Delta y_k)$ and

$$\Delta\mathbf{P}_k \times \mathbf{z} = (0, \Delta y_k, -\Delta x_k),$$

where $\Delta x_k = x_{k+1} - x_k$, $\Delta y_k = y_{k+1} - y_k$, and \mathbf{z} is a unit vector orthogonal to the plane, the offset $\mathbf{r}_d(t)$ is described by polynomials $W(t), X(t), Y(t)$ of degree $2n - 1$, with Bernstein coefficients $\mathbf{O}_k = (W_k, X_k, Y_k)$ given [11] by

$$\mathbf{O}_k = \sum_{j=\max(0, k-n)}^{\min(n-1, k)} \frac{\binom{n-1}{j} \binom{n}{k-j}}{\binom{2n-1}{k}} (\sigma_j \mathbf{P}_{k-j} + dn \Delta\mathbf{P}_j \times \mathbf{z}), \quad k = 0, \dots, 2n - 1.$$

As the offset distance d is increased, the control points of $\mathbf{r}_d(t)$ move uniformly along fixed lines, but their “weights” W_k remain constant; see Figure 17.2. Although the offsets to PH curves are of higher degree, this is not problematic if we adhere to the numerically stable Bernstein form in their construction (note also that we may regard the offset as the sum of a polynomial curve of degree n and a rational curve of degree $n - 1$). In §17.5 we shall see that the *rational* PH curves entail no increase of degree in their offsets.

17.2.2. Complex representation

By identifying points (x, y) in the plane with complex numbers $x + iy$, any plane curve $\mathbf{r}(t) = (x(t), y(t))$ can be regarded [80] as a complex-valued function $x(t) + iy(t)$ of a real parameter t . For planar PH curves, this perspective proves to be extremely useful [12] — since, if $\mathbf{w}(t) = u(t) + iv(t)$ is any complex polynomial with $\gcd(u, v) = 1$, its image

$$\mathbf{w}^2(t) = u^2(t) - v^2(t) + i2u(t)v(t)$$

under the conformal map $\mathbf{z} \rightarrow \mathbf{z}^2$ is a polynomial whose real and imaginary parts are elements of a primitive Pythagorean triple of the form (17.4), with $h(t) = 1$. Hence, in the complex representation, the (regular) PH curves are those curves whose hodographs are simply the *squares of complex polynomials*: $\mathbf{r}'(t) = \mathbf{w}^2(t)$.

The complex representation shows that the form $x'(t) = u^2(t) - v^2(t)$, $y'(t) = 2u(t)v(t)$ is invariant with respect to rotations about the origin. The rotated hodograph can be written

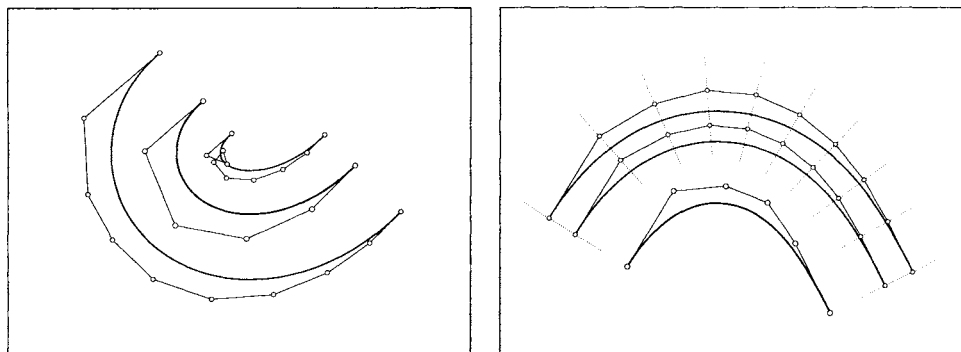


Figure 17.2. Left: interior and exterior offsets to a PH quintic, as rational Bézier curves of degree nine. Right: as the offset distance d is increased, the control points for successive offsets move uniformly along straight lines, and their weights remain constant.

as $\bar{\mathbf{r}}'(t) = \tilde{x}'(t) + i\tilde{y}'(t) = \exp(i\theta) \mathbf{r}'(t) = \bar{\mathbf{w}}^2(t)$, with $\bar{\mathbf{w}}(t) = \tilde{u}(t) + i\tilde{v}(t) = \exp(i\frac{1}{2}\theta) \mathbf{w}(t)$, and hence we deduce that $\tilde{x}'(t) = \tilde{u}^2(t) - \tilde{v}^2(t)$, $\tilde{y}'(t) = 2\tilde{u}(t)\tilde{v}(t)$ where

$$\tilde{u}(t) = \cos \frac{1}{2}\theta u(t) - \sin \frac{1}{2}\theta v(t), \quad \tilde{v}(t) = \sin \frac{1}{2}\theta u(t) + \cos \frac{1}{2}\theta v(t).$$

The complex form plays a key role in simplifying the construction and shape analysis of planar PH curves [3,12,13,19,20,24]. Suppose, for example, that

$$\mathbf{r}(t) = \sum_{k=0}^5 \mathbf{P}_k \binom{5}{k} (1-t)^{5-k} t^k$$

is a PH quintic in Bézier form, obtained by integrating the hodograph (17.4) with $h(t) = 1$ and quadratics with Bernstein coefficients u_0, u_1, u_2 and v_0, v_1, v_2 for $u(t)$ and $v(t)$. In real arithmetic, we obtain the rather cumbersome expressions

$$\begin{aligned} (x_1, y_1) &= (x_0, y_0) + \frac{1}{5}(u_0^2 - v_0^2, 2u_0v_0), \\ (x_2, y_2) &= (x_1, y_1) + \frac{1}{5}(u_0u_1 - v_0v_1, u_0v_1 + u_1v_0), \\ (x_3, y_3) &= (x_2, y_2) + \frac{2}{15}(u_1^2 - v_1^2, 2u_1v_1) + \frac{1}{15}(u_0u_2 - v_0v_2, u_0v_2 + u_2v_0), \\ (x_4, y_4) &= (x_3, y_3) + \frac{1}{5}(u_1u_2 - v_1v_2, u_1v_2 + u_2v_1), \\ (x_5, y_5) &= (x_4, y_4) + \frac{1}{5}(u_2^2 - v_2^2, 2u_2v_2), \end{aligned}$$

for the control points¹ $\mathbf{p}_k = (x_k, y_k)$. Writing $\mathbf{p}_k = x_k + i y_k$ and $\mathbf{w}_i = u_i + i v_i$, on the other hand, yields the compact characterization

$$\begin{aligned} \mathbf{p}_1 &= \mathbf{p}_0 + \frac{1}{5} \mathbf{w}_0^2, \\ \mathbf{p}_2 &= \mathbf{p}_1 + \frac{1}{5} \mathbf{w}_0 \mathbf{w}_1, \\ \mathbf{p}_3 &= \mathbf{p}_2 + \frac{2\mathbf{w}_1^2 + \mathbf{w}_0 \mathbf{w}_2}{15}, \\ \mathbf{p}_4 &= \mathbf{p}_3 + \frac{1}{5} \mathbf{w}_1 \mathbf{w}_2, \\ \mathbf{p}_5 &= \mathbf{p}_4 + \frac{1}{5} \mathbf{w}_2^2. \end{aligned} \tag{17.7}$$

By means of the complex form, one can easily see [12] that the set of (regular) PH curves and the set of “ordinary” (regular) polynomial curves are of the same cardinality. Familiar algorithms for the construction or modification of polynomial curves always admit analogs in terms of PH curves — although the latter are inherently non-linear, use of the complex form can greatly simplify their formulation and implementation.

17.2.3. PH space curves

By analogy with (17.3), a PH space curve $\mathbf{r}(t) = (x(t), y(t), z(t))$ satisfies

$$x'^2(t) + y'^2(t) + z'^2(t) = \sigma^2(t), \tag{17.8}$$

for some polynomial $\sigma(t)$. Such curves were introduced in [32], using the form

$$\begin{aligned} x'(t) &= [u^2(t) - v^2(t) - w^2(t)] h(t), \\ y'(t) &= 2u(t)v(t)h(t), \\ z'(t) &= 2u(t)w(t)h(t), \end{aligned} \tag{17.9}$$

and hence $\sigma(t) = h(t) [u^2(t) + v^2(t) + w^2(t)]$. This is not, however, an entirely satisfactory spatial extension of the hodograph (17.4). Whereas the latter form is both sufficient and necessary for a plane PH curve, the form (17.9) is *only sufficient* for a PH space curve. The failure of (17.9) to describe *all* PH space curves is apparent in the fact that this form is not invariant with respect to a re-labelling of the axes (the invariance of (17.4) can be seen by replacing (u, v) by (\tilde{u}, \tilde{v}) , where we define $\sqrt{2}\tilde{u} = u + v$ and $\sqrt{2}\tilde{v} = u - v$).

Subsequently, a sufficient-and-necessary characterization of polynomial solutions to (17.8) was given in terms of polynomials $u(t), v(t), p(t), q(t)$ by Dietz et al. [9]:

$$\begin{aligned} x'(t) &= u^2(t) + v^2(t) - p^2(t) - q^2(t), \\ y'(t) &= 2u(t)p(t) + 2v(t)q(t), \\ z'(t) &= 2u(t)q(t) - 2v(t)p(t), \end{aligned} \tag{17.10}$$

and thus $\sigma(t) = u^2(t) + v^2(t) + p^2(t) + q^2(t)$. Moreover, this defines a regular PH space curve with $\gcd(x', y', z') = 1$ whenever $u(t), v(t), p(t), q(t)$ have no common factor, whereas for (17.9) with $h(t) = 1$, the condition $\gcd(u, v, w) = 1$ does *not* guarantee a regular curve.

¹Note here that \mathbf{p}_0 is an arbitrary integration constant.

As with planar PH cubics, the twisted PH cubics can be characterized by geometrical constraints on their Bézier control polygons. In fact, the spatial PH cubics are all segments of (non-circular) helices [32] — i.e., their tangents maintain a constant angle with a given axis, and they exhibit a constant ratio of curvature to torsion.

The arc length $s(t)$ for PH space curves is obtained by a trivial extension of the methods given above for plane PH curves. The spatial analog of an offset curve is the *canal surface* with a given space curve as its spine (i.e., the envelope of a one-parameter family of fixed-radius spheres, centered on the spine curve). Since PH space curves admit orthonormal frames² $(\mathbf{t}, \mathbf{e}_1, \mathbf{e}_2)$ dependent rationally on t , where \mathbf{t} is the tangent and $\mathbf{e}_1, \mathbf{e}_2$ span the normal plane, the canal surfaces with PH spine curves are rational [32]. Lü and Pottmann showed that the canal surfaces associated with *any* rational (not just PH) spine curves are, in fact, rational [49] — but their rational forms are more difficult to construct [57]. Jüttler [40,43] describes applications of PH space curves to the modelling of swept surfaces.

An algebraic model for the spatial PH curves, analogous to the complex representation (see §17.2.2) of planar PH curves, was formulated in terms of quaternions³ by Choi et al. [7]. Let $1, \mathbf{i}, \mathbf{j}, \mathbf{k}$ be elements of the quaternion basis, satisfying $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$ and

$$\mathbf{i}\mathbf{j} = -\mathbf{j}\mathbf{i} = \mathbf{k}, \quad \mathbf{j}\mathbf{k} = -\mathbf{k}\mathbf{j} = \mathbf{i}, \quad \mathbf{k}\mathbf{i} = -\mathbf{i}\mathbf{k} = \mathbf{j}.$$

Then, in terms of the quaternion-valued polynomial

$$\mathcal{A}(t) = u(t) + v(t)\mathbf{i} + p(t)\mathbf{j} + q(t)\mathbf{k}, \tag{17.11}$$

the hodograph (17.10) can be expressed in quaternion form as

$$\begin{aligned} \mathbf{r}'(t) = \mathcal{A}(t)\mathbf{i}\mathcal{A}^*(t) &= [u^2(t) + v^2(t) - p^2(t) - q^2(t)]\mathbf{i} \\ &+ [2u(t)q(t) + 2v(t)p(t)]\mathbf{j} + [2v(t)q(t) - 2u(t)p(t)]\mathbf{k}, \end{aligned} \tag{17.12}$$

$\mathcal{A}^*(t) = u(t) - v(t)\mathbf{i} - p(t)\mathbf{j} - q(t)\mathbf{k}$ being the conjugate of $\mathcal{A}(t)$. The choice of the basis element \mathbf{i} between $\mathcal{A}(t)$ and $\mathcal{A}^*(t)$ has no special significance; choosing \mathbf{j} or \mathbf{k} instead leads to permutations of x', y', z' and u, v, p, q . We also note that there is a one-parameter family of quaternion polynomials that yield exactly the same spatial Pythagorean hodograph [17]: for any real ξ , we can post-multiply (17.11) by $\cos \xi + \sin \xi \mathbf{i}$ without altering (17.12).

The quaternion representation (17.12) is useful in establishing a key property of the form (17.10), namely, its invariance — unlike (17.9) — with respect to spatial rotations [17]. Suppose $\mathbf{r}'(t) = (x'(t), y'(t), z'(t))$ is rotated by angle θ about the unit axis vector $\mathbf{n} = (n_x, n_y, n_z)$ to yield $\tilde{\mathbf{r}}'(t) = (\tilde{x}'(t), \tilde{y}'(t), \tilde{z}'(t))$. We wish to express the components of the latter in the form (17.10), in terms of four new polynomials $\tilde{u}(t), \tilde{v}(t), \tilde{p}(t), \tilde{q}(t)$. Now since the rotation of $\mathbf{r}'(t)$ into $\tilde{\mathbf{r}}'(t)$ has [65] the quaternion description

$$\tilde{\mathbf{r}}'(t) = \mathcal{U}\mathbf{r}'(t)\mathcal{U}^*,$$

²In general, \mathbf{e}_1 and \mathbf{e}_2 do not coincide with the principal normal \mathbf{n} and binormal \mathbf{z} — for a discussion of curves with rational Frenet frames, see Wagner and Ravani [77].

³Ueda [76] has also expressed PH space curves in terms of (a special class of) quaternions, but he employs the special hodograph form (17.9) rather than the sufficient-and-necessary form (17.10).

where $\mathcal{U} = \cos \frac{1}{2}\theta + \sin \frac{1}{2}\theta(n_x \mathbf{i} + n_y \mathbf{j} + n_z \mathbf{k})$ is a unit quaternion satisfying $\mathcal{U}\mathcal{U}^* = 1$, the new polynomials can be obtained [17] through the linear transformation

$$\begin{bmatrix} \tilde{u}(t) \\ \tilde{v}(t) \\ \tilde{p}(t) \\ \tilde{q}(t) \end{bmatrix} = \begin{bmatrix} \cos \frac{1}{2}\theta & -n_x \sin \frac{1}{2}\theta & -n_y \sin \frac{1}{2}\theta & -n_z \sin \frac{1}{2}\theta \\ n_x \sin \frac{1}{2}\theta & \cos \frac{1}{2}\theta & -n_z \sin \frac{1}{2}\theta & n_y \sin \frac{1}{2}\theta \\ n_y \sin \frac{1}{2}\theta & n_z \sin \frac{1}{2}\theta & \cos \frac{1}{2}\theta & -n_x \sin \frac{1}{2}\theta \\ n_z \sin \frac{1}{2}\theta & -n_y \sin \frac{1}{2}\theta & n_x \sin \frac{1}{2}\theta & \cos \frac{1}{2}\theta \end{bmatrix} \begin{bmatrix} u(t) \\ v(t) \\ p(t) \\ q(t) \end{bmatrix}. \quad (17.13)$$

17.3. CONSTRUCTION ALGORITHMS

Since PH curves are defined by hodographs that depend on the squares and products of polynomials $u(t)$, $v(t)$, etc., the determination of coefficients for these polynomials so as to match given discrete geometrical data (points, tangents, etc.) typically incurs non-linear problems with a multiplicity of solutions.

17.3.1. PH quintic Hermite interpolants

The first-order Hermite interpolation problem is concerned with constructing a smooth curve, $\mathbf{r}(t)$ for $t \in [0, 1]$, with given end points and derivatives: $\mathbf{r}(0) = \mathbf{p}_0$, $\mathbf{r}'(0) = \mathbf{d}_0$ and $\mathbf{r}(1) = \mathbf{p}_1$, $\mathbf{r}'(1) = \mathbf{d}_1$. As is well known, there is a unique solution among the ‘‘ordinary’’ cubics; to obtain sufficient degrees of freedom within the PH curves, we must appeal to the quintics [24]. It is convenient to use the complex representation, and assume⁴ that $\mathbf{p}_0 = 0$ and $\mathbf{p}_1 = 1$ (note that bold characters denote points, vectors, and complex variables).

To define a PH quintic, we choose a hodograph that is the square of a complex quadratic polynomial $\mathbf{w}(t)$ expressed in Bernstein form,

$$\mathbf{r}'(t) = [\mathbf{w}_0(1-t)^2 + \mathbf{w}_1 2(1-t)t + \mathbf{w}_2 t^2]^2. \quad (17.14)$$

With the integration constant $\mathbf{r}(0) = \mathbf{p}_0$, the coefficients $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2$ are determined by the Hermite interpolation conditions

$$\mathbf{r}'(0) = \mathbf{d}_0, \quad \mathbf{r}'(1) = \mathbf{d}_1, \quad \int_0^1 \mathbf{r}'(t) dt = \mathbf{p}_1 - \mathbf{p}_0 = 1,$$

which yield the system of quadratic equations

$$\mathbf{w}_0^2 = \mathbf{d}_0, \quad \mathbf{w}_2^2 = \mathbf{d}_1, \quad (17.15)$$

$$\mathbf{w}_0^2 + \mathbf{w}_0 \mathbf{w}_1 + \frac{2\mathbf{w}_1^2 + \mathbf{w}_2 \mathbf{w}_0}{3} + \mathbf{w}_1 \mathbf{w}_2 + \mathbf{w}_2^2 = 5. \quad (17.16)$$

This has a simple formal solution: equations (17.15) furnish two complex values for each of $\mathbf{w}_0, \mathbf{w}_2$ and substituting them into (17.16) allows the latter to be solved as a quadratic equation in \mathbf{w}_1 . Although there are 8 solutions, they define only 4 distinct PH quintics: if $\mathbf{w}_0, \mathbf{w}_2, \mathbf{w}_1$ is a solution, so is $-\mathbf{w}_0, -\mathbf{w}_2, -\mathbf{w}_1$, and it yields exactly the same curve.

⁴This ‘‘standard form’’ for the Hermite data helps simplify the analysis -- it is a trivial matter to map arbitrary Hermite data to and from it.

Empirically, one “good” interpolant is observed among the four distinct solutions — the others typically exhibit undesired loops or extreme curvature variations. The good interpolant may be identified as the one that minimizes a global measure of shape, such as the *absolute rotation index* or *elastic bending energy* (see Figure 17.3):

$$\mathcal{R}_{\text{abs}} = \frac{1}{2\pi} \int_0^1 |\kappa(t)| |\mathbf{r}'(t)| dt, \quad \mathcal{E} = \int_0^1 \kappa^2(t) |\mathbf{r}'(t)| dt. \tag{17.17}$$

The complex form facilitates exact evaluation of these quantities [13,24]. For this purpose, it is convenient to re-write the hodograph (17.14) as

$$\mathbf{r}'(t) = \mathbf{k} (t - \mathbf{a})^2 (t - \mathbf{b})^2,$$

in terms of which the curvature can be expressed as

$$\kappa(t) = \frac{\text{Im}(\bar{\mathbf{r}}'(t)\mathbf{r}''(t))}{|\mathbf{r}'(t)|^3} = \frac{2}{|\mathbf{k}|} \frac{\beta|t - \mathbf{a}|^2 + \alpha|t - \mathbf{b}|^2}{|(t - \mathbf{a})(t - \mathbf{b})|^4}, \tag{17.18}$$

where $\alpha = \text{Im}(\mathbf{a})$ and $\beta = \text{Im}(\mathbf{b})$. The locations of \mathbf{a} and \mathbf{b} in the complex plane relative to the interval $[0, 1]$ play a key role in determining the shape of PH quintics [24].

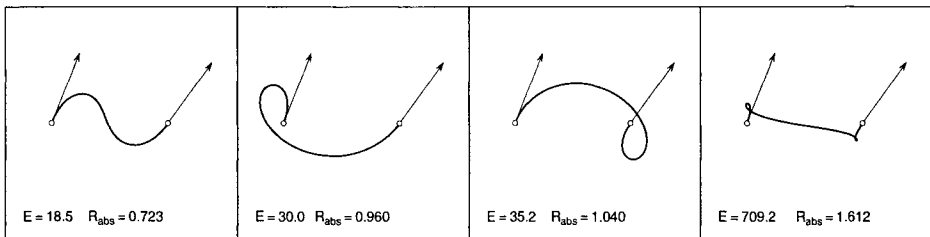


Figure 17.3. The four distinct PH quintic Hermite interpolants to the data $\mathbf{p}_0 = 0$, $\mathbf{p}_1 = 1$ and $\mathbf{d}_0 = 0.24 + i0.60$, $\mathbf{d}_1 = 0.38 + i0.52$, together with values for the bending energy and absolute rotation index (17.17). The derivatives are shortened by a factor of 5 for clarity.

An alternative approach to selecting the good solution employs a qualitative criterion — “absence of anti-parallel tangents” — based on comparing the PH quintic and ordinary cubic interpolants [52]: one can show that, for “reasonable” Hermite data $\mathbf{d}_0, \mathbf{d}_1$ satisfying

$$\text{Re}(\mathbf{d}_i) > 0 \quad \text{and} \quad |\mathbf{d}_i| < 3, \tag{17.19}$$

the “good” solution can be directly constructed by making a specific choice of signs in the solution. The conditions (17.19) ensure that the derivatives have positive components in the direction of the displacement $\mathbf{p}_1 - \mathbf{p}_0$, and their magnitudes are commensurate with the distance $|\mathbf{p}_1 - \mathbf{p}_0| = 1$ (as would be expected in most practical applications).

17.3.2. Shape properties of PH quintics

A remarkable (empirical) feature of the “good” PH quintic Hermite interpolants is that they are generally of *fairer shape* — i.e., they exhibit more even curvature profiles — than the corresponding “ordinary” cubics, as is evident from the examples shown in Figure 17.4. This is true not only for individual Hermite segments, but also C^2 splines that interpolate a sequence of N points (see Figure 17.5). The superior curvature behavior of PH curves is advantageous not only in free-form design applications, but also in path planning for mobile robots — where physical limitations of the steering mechanism incur constraints on the allowed path curvature [5]. The curvature (17.18) of a PH quintic is a *rational* function of the curve parameter, with positive denominator: expressing it in Bernstein form yields an algorithm to compute rapidly-convergent bounds on the curvature of PH curves using only rational arithmetic on their coefficients [52]. Note also that the availability of closed-form expressions for the total arc length S and the elastic bending energy \mathcal{E} in (17.17) opens up the possibility of quantitative “shape optimization” for PH curves [13].

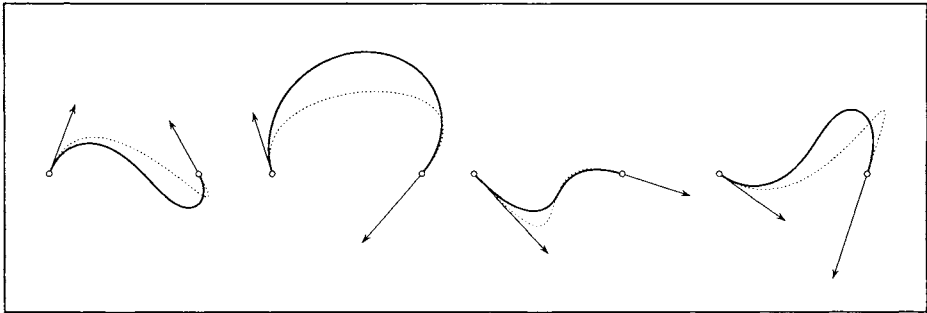


Figure 17.4. Comparison of the “good” PH quintic (solid) and ordinary cubic (dashed) Hermite interpolants to various end derivatives \mathbf{d}_0 and \mathbf{d}_1 (shortened by a factor of 5).

17.3.3. C^2 PH quintic splines

Apart from individual Hermite interpolants to end-point data, the ability to smoothly interpolate a sequence of points $\mathbf{p}_0, \dots, \mathbf{p}_N$ is a common design requirement. As is well-known, C^2 cubic splines satisfy this need and incur only the solution of a tridiagonal linear system. An analogous construction is also possible [3] for C^2 PH quintic splines — the defining equations still have “bandwidth” 3, but are *complex* and *quadratic*, and thus computationally more challenging. As compensation for the greater computational cost, however, we shall see that the PH quintic splines provide much “smoother” loci (with more even curvature distributions) than their ordinary cubic counterparts.

The construction of a C^2 PH quintic spline, interpolating a sequence of $N + 1$ points⁵

⁵It is understood here that the points are specified in complex form, $\mathbf{p}_k = x_k + iy_k$.

$\mathbf{p}_0, \dots, \mathbf{p}_N$ and satisfying specified end conditions, entails solving a system of N quadratic equations in N complex unknowns $\mathbf{z}_1, \dots, \mathbf{z}_N$. We begin by writing the hodograph of the k -th PH quintic span $\mathbf{r}_k(t)$ of the spline curve, between \mathbf{p}_{k-1} and \mathbf{p}_k , in the form

$$\mathbf{r}'_k(t) = \left[\frac{1}{2}(\mathbf{z}_{k-1} + \mathbf{z}_k)(1-t)^2 + \mathbf{z}_k 2(1-t)t + \frac{1}{2}(\mathbf{z}_k + \mathbf{z}_{k+1})t^2 \right]^2, \tag{17.20}$$

which ensures that successive spans satisfy the continuity conditions $\mathbf{r}'_k(1) = \mathbf{r}'_{k+1}(0)$ and $\mathbf{r}''_k(1) = \mathbf{r}''_{k+1}(0)$. Assigning the integration constant $\mathbf{r}_k(0) = \mathbf{p}_{k-1}$ to this hodograph and also requiring that $\mathbf{r}_k(1) = \mathbf{p}_k$ then gives the equation

$$\mathbf{f}_k(\mathbf{z}_1, \dots, \mathbf{z}_N) = 3\mathbf{z}_{k-1}^2 + 27\mathbf{z}_k^2 + 3\mathbf{z}_{k+1}^2 + \mathbf{z}_{k-1}\mathbf{z}_{k+1} + 13(\mathbf{z}_{k-1} + \mathbf{z}_{k+1})\mathbf{z}_k - 60\Delta\mathbf{p}_k = 0, \tag{17.21}$$

where $\Delta\mathbf{p}_k = \mathbf{p}_k - \mathbf{p}_{k-1}$. Such an equation holds for each span $k = 1, \dots, N$ of the spline curve, but the first and last equations, $\mathbf{f}_1(\mathbf{z}_1, \dots, \mathbf{z}_N) = 0$ and $\mathbf{f}_N(\mathbf{z}_1, \dots, \mathbf{z}_N) = 0$, must be modified to embody the prescribed end conditions. The modifications appropriate to (a) given end-derivatives $\mathbf{d}_0, \mathbf{d}_N$ at the points $\mathbf{p}_0, \mathbf{p}_N$; (b) cubic (Tschirnhausen) end spans; and (c) periodic end conditions, are described in [3].

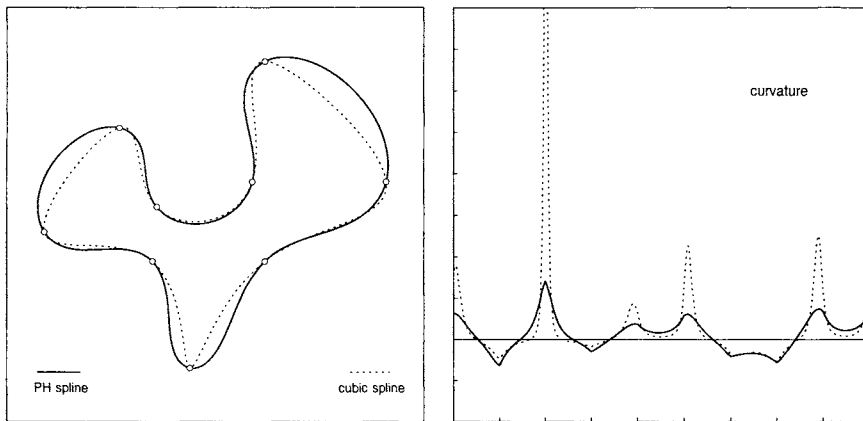


Figure 17.5. Comparison of C^2 PH quintic and “ordinary” C^2 cubic splines interpolating a sequence of points with uniform knots and periodic end conditions — the PH quintic spline yields a much “smoother” curve, as indicated by the curvature profiles on the right.

The system (17.21) is “tridiagonal” in the sense that each equation contains only three consecutive unknowns. Its non-linear nature, however, makes it more challenging to solve than the *linear* tridiagonal system for “ordinary” cubic splines. In general, there are 2^{N+k} distinct solutions,⁶ among which one “good” PH spline may be identified (see Figure 17.5).

⁶Here $k \in \{-1, 0, +1\}$ depends on the adopted end conditions.

For $N \leq 10$, the *homotopy method* [4,53] is a practical means to compute all solutions of this system — using a predictor–corrector method we track [3] the solutions to

$$\mathbf{h}_k(\mathbf{z}_1, \dots, \mathbf{z}_N, \lambda) = (1 - \lambda) \mathbf{g}_k(\mathbf{z}_1, \dots, \mathbf{z}_N) + \lambda \mathbf{f}_k(\mathbf{z}_1, \dots, \mathbf{z}_N) = 0$$

from the known solutions of a “simple” initial system, $\mathbf{g}_k(\mathbf{z}_1, \dots, \mathbf{z}_N) = 0$ at $\lambda = 0$, to the solutions of the desired system, $\mathbf{f}_k(\mathbf{z}_1, \dots, \mathbf{z}_N) = 0$ at $\lambda = 1$. Since the system (17.21) is typically well–conditioned, the homotopy method often yields convergence to machine precision. However, the cost of computing all 2^{N+k} solutions is prohibitive for $N > 10$. As an alternative, a method to compute *only* the “good” solution is described in [19], based on estimating an initial approximation to the solution (by comparison with the ordinary cubic spline), and invoking the Kantorovich conditions for guaranteed convergence [45,55,70] of the multivariate Newton–Raphson method applied to the system (17.21). The Kantorovich test is facilitated by the fact that, in the ∞ –norm, the Jacobian matrix with elements

$$\mathbf{M}_{kl} = \frac{\partial \mathbf{f}_k}{\partial \mathbf{z}_l} \quad \text{for } 1 \leq k, l \leq N$$

satisfies [19] the *global Lipschitz condition*

$$\| \mathbf{M}(\mathbf{x}_1, \dots, \mathbf{x}_N) - \mathbf{M}(\mathbf{y}_1, \dots, \mathbf{y}_N) \|_\infty \leq 120 \| (\mathbf{x}_1, \dots, \mathbf{x}_N) - (\mathbf{y}_1, \dots, \mathbf{y}_N) \|_\infty.$$

Reference [19] also presents a generalization of the system (17.21) to PH quintic splines with non–uniform (rather than integer) knots t_0, \dots, t_N for the points $\mathbf{p}_0, \dots, \mathbf{p}_N$.

17.3.4. Spatial PH quintic Hermite interpolants

The above constructions were concerned with planar PH curves. For PH space curves, a Hermite interpolation algorithm based on the form (17.9) was presented in [32]. However, this form is not rotation invariant — if we interpolate rotated Hermite data, the result is different from that obtained by interpolating first and then rotating the interpolant.

To obtain rotation–invariant interpolants, we must use the form (17.10) — specifically, we employ the representation (17.12) with the quadratic polynomial

$$\mathcal{A}(t) = \mathcal{A}_0(1 - t)^2 + \mathcal{A}_1 2(1 - t)t + \mathcal{A}_2 t^2$$

$\mathcal{A}_r = u_r + v_r \mathbf{i} + p_r \mathbf{j} + q_r \mathbf{k}$ for $r = 0, 1, 2$ being quaternion coefficients, to be determined by matching the Hermite data $\mathbf{p}_0, \mathbf{d}_0$ and $\mathbf{p}_1, \mathbf{d}_1$ (interpreted as “pure vector” quaternions). The interpolation conditions $\mathbf{r}'(0) = \mathbf{d}_0, \mathbf{r}'(1) = \mathbf{d}_1$, and $\int_0^1 \mathbf{r}'(t) dt = \mathbf{p}_1 - \mathbf{p}_0$ thus yield [18] the system of equations

$$\mathcal{A}_0 \mathbf{i} \mathcal{A}_0^* = \mathbf{d}_0, \quad \mathcal{A}_2 \mathbf{i} \mathcal{A}_2^* = \mathbf{d}_1, \tag{17.22}$$

$$\begin{aligned} (3\mathcal{A}_0 + 4\mathcal{A}_1 + 3\mathcal{A}_2) \mathbf{i} (3\mathcal{A}_0 + 4\mathcal{A}_1 + 3\mathcal{A}_2)^* \\ = 120(\mathbf{p}_1 - \mathbf{p}_0) - 15(\mathbf{d}_0 + \mathbf{d}_1) + 5(\mathcal{A}_0 \mathbf{i} \mathcal{A}_2^* + \mathcal{A}_2 \mathbf{i} \mathcal{A}_0^*) \end{aligned} \tag{17.23}$$

for $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2$. This system may be solved by noting that the equation $\mathcal{A} \mathbf{i} \mathcal{A}^* = \mathbf{d}$, where \mathbf{d} is a given pure vector quaternion, admits the one–parameter family of solutions

$$\mathcal{A} = \sqrt{\frac{1}{2}(1 + \lambda)|\mathbf{d}|} \left(-\sin \theta + \cos \theta \mathbf{i} + \frac{\mu \cos \theta + \nu \sin \theta}{1 + \lambda} \mathbf{j} + \frac{\nu \cos \theta - \mu \sin \theta}{1 + \lambda} \mathbf{k} \right) \tag{17.24}$$

where (λ, μ, ν) are the direction cosines of \mathbf{d} , and θ is a free variable. By solving (17.22) for $\mathcal{A}_0, \mathcal{A}_2$ we can substitute for them on the right-hand side of (17.23), which is a pure vector quaternion. Equation (17.23) can then be solved for $3\mathcal{A}_0 + 4\mathcal{A}_1 + 3\mathcal{A}_2$ using (17.24).

Although the solution incurs three indeterminate angular variables $\theta_0, \theta_1, \theta_2$ (associated with $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2$) it can be shown [18] that the interpolants depend only on the *differences* of these angles. Thus we may, without loss of generality, assume that $\theta_1 = 0$ and optimize certain shape properties of the interpolant with respect to the remaining free parameters θ_0, θ_2 (see [18] for further details). Once $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2$ are known, the Bézier control points of the interpolant are given, in quaternion form $\mathbf{p}_r = x_r \mathbf{i} + y_r \mathbf{j} + z_r \mathbf{k}$, by

$$\begin{aligned} \mathbf{p}_1 &= \mathbf{p}_0 + \frac{1}{5} \mathcal{A}_0 \mathbf{i} \mathcal{A}_0^*, \\ \mathbf{p}_2 &= \mathbf{p}_1 + \frac{1}{10} (\mathcal{A}_0 \mathbf{i} \mathcal{A}_1^* + \mathcal{A}_1 \mathbf{i} \mathcal{A}_0^*), \\ \mathbf{p}_3 &= \mathbf{p}_2 + \frac{1}{30} (\mathcal{A}_0 \mathbf{i} \mathcal{A}_2^* + 4 \mathcal{A}_1 \mathbf{i} \mathcal{A}_1^* + \mathcal{A}_2 \mathbf{i} \mathcal{A}_0^*), \\ \mathbf{p}_4 &= \mathbf{p}_3 + \frac{1}{10} (\mathcal{A}_1 \mathbf{i} \mathcal{A}_2^* + \mathcal{A}_2 \mathbf{i} \mathcal{A}_1^*), \\ \mathbf{p}_5 &= \mathbf{p}_4 + \frac{1}{5} \mathcal{A}_2 \mathbf{i} \mathcal{A}_2^*. \end{aligned} \tag{17.25}$$

17.3.5. Geometric Hermite interpolants

Jüttler [42,43] has proposed an alternate approach to the construction of polynomial PH curves, based upon *geometric* rather than *parametric* Hermite interpolation — i.e., intrinsic geometrical properties (tangent directions, curvatures, etc.) are specified in lieu of parametric derivatives. A scheme for interpolating G^1 spatial data (points and tangent directions) by PH space cubics is described in [43], while interpolation of G^2 planar data (points, tangent directions, and curvatures) by plane PH curves of degree 7 is treated in [42]. The methods are well-suited to approximating given (non-PH) curves, from which tangents, curvatures, etc., can be computed. Conditions for (and rates of) convergence to the given curve, as the sampling interval diminishes, are also addressed.

17.3.6. Further constructions

Many other constructions, suited to specific applications, have been discussed by various authors. Walton and Meek consider the imposition of a curvature-monotonicity constraint on PH curves [78,79]; see also [14]. PH curves of degree 9 have been employed as second-order Hermite interpolants in the design of smooth cam profiles [20]. Sabin [66] describes a “B-spline approach” to constructing smooth piecewise PH cubic curves. Least-squares fitting of PH curves to point data has also been investigated [29], as a means of making G code part programs accessible to real-time PH curve CNC interpolators (see §17.4). Finally, several special PH curve constructions have been explored by Ueda [72–75].

17.4. REAL-TIME CNC INTERPOLATORS

Certain properties of polynomial PH curves are especially advantageous in the context of computer-numerical-control (CNC) machining. For a CNC machine to cut a specified

curve⁷ $\mathbf{r}(\xi)$, the tool center must follow the offset path (17.5) where d is the tool radius. As previously noted, PH curves (unlike general polynomial curves) have *rational* offsets, that are amenable to exact representation in CAD systems.

The ability to formulate exact *real-time interpolators*, for constant or variable feedrates, is another fundamental advantage [34] of PH curves. To produce a desired motion, a CNC machine drives several independent axes in a coordinated manner. The controller employs digital descriptions of space and time — the *sampling interval* (~ 0.001 sec) is defined by a “clock” running within the algorithm, while the *basic length unit* or spatial resolution (~ 10 microns) is determined by position encoders on each axis.

The controller compares the actual machine position (measured by the encoders) with the intended position (computed from specified paths and feedrates by the interpolator) in each sampling interval Δt — the error is used to generate control signals for the machine drives, ensuring that the desired paths and feedrates are accurately realized. The timed sequence of curve points computed by the interpolator are called *reference points*; they correspond to a discrete sampling $\xi_k = \xi(k\Delta t)$ of the solution to the differential equation

$$\sigma \frac{d\xi}{dt} = V, \quad (17.26)$$

where $\sigma(\xi) = |\mathbf{r}'(\xi)|$ denotes the parametric speed and $V = ds/dt$ is the feedrate (which may be either a constant, or dependent upon physical variables such as elapsed time t , arc length s along the path, or the local path curvature κ).

For general polynomial curves, equation (17.26) does not admit closed-form integration, even when $V = \text{constant}$. Hence, it is common practice to use piecewise-linear/circular “G code” approximations to curved tool paths. Apart from its data-intensive nature, this approach can severely impede the ability of the machine to achieve and maintain high speeds [71]. Alternately, one may retain the analytic representation of a curved path, and use a Taylor series expansion

$$\xi_k = \xi_{k-1} + \frac{V}{\sigma} \Delta t + \frac{V}{\sigma^2} \left(V' - \frac{\mathbf{r}' \cdot \mathbf{r}''}{\sigma^2} V \right) \frac{(\Delta t)^2}{2} + \dots \quad (17.27)$$

(where primes denote derivatives with respect to ξ , and σ , \mathbf{r}' , \mathbf{r}'' , V , V' , etc., are evaluated⁸ at ξ_{k-1}) to approximate the reference points. Ordinarily, only the linear term is retained, and no attempt is made to account for the accumulation of truncation errors. Moreover, as noted in [35], this method has often been compromised in the context of variable feedrates by erroneous derivations of the appropriate Taylor coefficients.

PH curves circumvent these problems in a simple and elegant manner — their algebraic structure permits closed-form integration of (17.26) yielding an equation of the form

$$F(\xi_k) = c_{k-1}, \quad (17.28)$$

where F is a monotone (usually polynomial) function, and c_k is a constant that is updated at each step. The monotonicity of F allows ξ_k to be computed to machine precision by just a few Newton-Raphson iterations, starting from ξ_{k-1} .

⁷We use ξ as the curve parameter here, since the variable t will be used to denote time.

⁸A varying feedrate V should be specified as a function of a physically meaningful variable (e.g., time t , arc length s , or curvature κ) rather than the curve parameter ξ . Accordingly, the quantity $V' = dV/d\xi$ in (17.27) must be cast in terms of derivatives with respect to such a variable — see [35] for details.

Complete details on PH curve interpolators can be found in [34] for feedrates that are constant or simple functions of the arc length s or curvature κ ; in [21] for feedrates that maintain a constant material removal rate at fixed depth of cut along a curved path; and in [71] for any time-dependent feedrate function that has an elementary indefinite integral (the latter are especially useful in achieving smooth feed accelerations and decelerations).

Dramatic improvements in feedrate performance have been observed [71] on replacing G code interpolators by PH curve interpolators, primarily due to the “block look-ahead” problem associated with G codes. Further practical aspects concerning the use of PH curve interpolators include: conventions for specifying PH curve part programs [22]; control of cutting forces by feedrate variation [21]; path planning for contour machining of surfaces [36]; and the determination of feedrates and feed accelerations, compatible with the known torque and power capacity of the machine drives [37].

17.5. RATIONAL CURVES WITH RATIONAL OFFSETS

Although rational offsets are a key attribute of the polynomial PH curves, these curves are not the only “simple” curves that possess rational offsets. Lü [47,48] has shown that by suitable (improper) re-parameterizations, certain polynomial curves — whose hodographs are not Pythagorean — also admit rational offsets. Moreover, it seems natural to extend the domain of enquiry and ask: what is the complete set of *rational* curves whose offsets are rational? The theory of rational PH curves, as developed by Pottmann [61] and Fiorot and Gensane [38], addresses this problem conclusively. We can only skim the surface of this elegant theory — the reader should consult the references for complete details.

17.5.1. Rational PH curves

A basic difference is apparent upon turning our attention from polynomial to rational PH curves. Whereas the former can be constructed by integrating *any* polynomial hodograph that satisfies the Pythagorean condition (17.3), this fact does not extend to rational PH curves — a rational hodograph satisfying (17.3) does not necessarily define a rational PH curve, since transcendental terms may arise upon integrating rational functions. Thus, a different approach to the construction of rational PH curves is advantageous.

A rational curve $\mathbf{r}(t) = (x(t), y(t))$ can be specified by homogeneous point coordinates $W(t), X(t), Y(t)$ such that $x(t) = X(t)/W(t)$, $y(t) = Y(t)/W(t)$. Alternately, we can use homogeneous line coordinates $K(t), L(t), M(t)$ such that the curve tangent at each point t is described by

$$K(t) + L(t)x + M(t)y = 0. \quad (17.29)$$

As shown by Pottmann [61], the latter approach is preferable in the theory of rational PH curves. Such curves have rational unit normals $\mathbf{n}(t) = (n_x(t), n_y(t))$ expressed in terms of polynomials $u(t), v(t)$ with $\gcd(u, v) = 1$ by

$$n_x(t) = \frac{u^2(t) - v^2(t)}{u^2(t) + v^2(t)}, \quad n_y(t) = \frac{2u(t)v(t)}{u^2(t) + v^2(t)}.$$

Now the tangent line can also be written in the form

$$n_x(t)x + n_y(t)y = f(t), \quad (17.30)$$

where $f(t)$ specifies the (signed) distance of the tangent line from the origin: this function must be rational, since $(x, y) = (x(t), y(t))$ evidently satisfies equation (17.30). Now we may, without loss of generality, set $[u^2(t) + v^2(t)]f(t) = -p(t)/q(t)$, where $p(t)$ and $q(t)$ are polynomials with $\gcd(p, q) = 1$. Comparing (17.29) and (17.30), we see that rational PH curves are defined by line coordinates of the form

$$K(t) : L(t) : M(t) = p(t) : [u^2(t) - v^2(t)]q(t) : 2u(t)v(t)q(t). \quad (17.31)$$

Writing $\mu = \max(\deg(u), \deg(v))$ and $\nu = \max(\deg(p), \deg(q))$, this defines a rational PH curve of class⁹ $m = 2\mu + \nu$. The corresponding point coordinates can be derived [61] as

$$\begin{aligned} W(t) &= 2[u^2(t) + v^2(t)][u(t)v'(t) - u'(t)v(t)]q^2(t), \\ X(t) &= 2[u'(t)v(t) + u(t)v'(t)]p(t)q(t) - 2u(t)v(t)[p'(t)q(t) - p(t)q'(t)], \\ Y(t) &= [u^2(t) - v^2(t)][p'(t)q(t) - p(t)q'(t)] - 2[u(t)u'(t) - v(t)v'(t)]p(t)q(t), \end{aligned} \quad (17.32)$$

from which we deduce that the rational PH curve is of order $n = 4\mu + 2\nu - 2$.

Clearly, the line representation (17.31) is much simpler than the point representation (17.32). Thus, algorithms for the design or construction of rational PH curves, analogous to those described in §17.3 for polynomial PH curves, rely exclusively on the rational dual Bézier form [1,60,62,68,69], introduced by Hoschek [39]. The offsets to rational PH curves are easily constructed by noting that their tangent lines are parallel to those of the original curve at corresponding points. Writing $f(t) + d$ on the right in (17.30) amounts to simply displacing the tangent parallel to itself by distance d . Hence, by the definition of $p(t)$ and $q(t)$, the offset at distance d from a rational PH curve is obtained by replacing $p(t)$ with $p(t) - d[u^2(t) + v^2(t)]q(t)$ in (17.31) or (17.32). This has the remarkable consequence that the offsets to a rational PH curve all have the *same* degree as that curve; recall that the offsets to a (regular) degree- n polynomial PH curve are of degree $2n - 1$.

The arc-length function (17.2) is another important difference between the polynomial and rational PH curves. For polynomial PH curves, $s(t)$ is *always* a polynomial. However, for rational PH curves, $s(t)$ is not always a rational function: although the parametric speed is rational, transcendental terms may arise in its integral. Pottmann [61] has shown that any rational PH curve for which $s(t)$ is rational must be the *evolute*¹⁰ of a rational PH curve and its family of offsets. Conversely, rational PH curves can be characterized as the *involute*s to rational curves with rational arc-length functions $s(t)$.

An elegant exposition of the theory of rational PH curves in the context of Laguerre geometry was subsequently developed by Peternell and Pottmann [58,63] — this reveals interesting connections between rational PH curves and the *caustics* and *anticaustics* of geometrical optics [16], as emphasized in the theory of rational PH curves developed [38] by Fiorot and Gensane. Unlike the polynomial PH curves, rational PH curves also admit a natural generalization to rational surfaces with rational offsets [61], although practical design schemes for such surfaces are not easy to formulate — see, however, [41,44,56,69].

⁹The *class* (the degree of the line representation) indicates the number of curve tangents incident with any point in the plane, and the *order* (the degree of the point representation) indicates the number of curve points incident with any line in the plane [64].

¹⁰The evolute of a given curve is the locus of its centers of curvature (or, equivalently, the envelope of its normals). A locus whose evolute is a given curve is called an involute of that curve — there is an infinite family of involutes, which are all offsets of each other.

For a detailed reconciliation of the properties of polynomial and rational PH curves, and a comparison of their relative advantages, see [26].

17.5.2. Improper parameterizations

For PH curves, the parameterization (17.5) of the offset is induced by that of the original curve $\mathbf{r}(t)$. It is conceivable, however, that polynomial curves exist whose offsets are not rational in the original curve parameter, but become rational under a re-parameterization. This circumstance was completely characterized by Lü, who showed [47,48] that, in the complex representation, it corresponds to complex polynomial hodographs of the form

$$\mathbf{r}'(t) = (\mathbf{k}t + 1) \mathbf{w}^2(t) h(t), \quad (17.33)$$

where $h(t)$ is a real polynomial, $\mathbf{w}(t) = u(t) + i v(t)$ is a complex polynomial, and \mathbf{k} is a complex constant. Clearly, expression (17.33) subsumes the polynomial PH curves as the special case $\mathbf{k} = 0$. The simplest examples with $\mathbf{k} \neq 0$ are those with $\mathbf{w}(t) = 1$ and $h(t)$ either a constant or linear polynomial; they define a parabola and cuspidal cubic. These curves admit rational re-parameterizations that correspond to double tracings: once in the “forward” direction, and once in “reverse.” In terms of these improper parameterizations the *two-sided offset curve*, at distance $\pm d$, is found to admit a rational parameterization of degree 6 for the parabola and 8 for the cuspidal cubic; see [33,47,48].

A subset of the curves with hodographs of the form (17.33) are *algebraically rectifiable* [67] — i.e., the arc length is given by the square root of a polynomial in the curve parameter. Writing $f(t) = |\mathbf{k}t + 1|^2$, the condition for algebraic rectifiability of the curve defined by (17.33) is that

$$h(t) [u^2(t) + v^2(t)] = 3f'(t)g(t) + 2f(t)g'(t)$$

must hold [48,67] for some polynomial $g(t)$ — the cuspidal cubic, for example, satisfies this condition, but the parabola does not.

17.6. MINKOWSKI PH CURVES

Thus far, we have defined PH curves in terms of the Euclidean metric in two and three dimensions. In some application contexts, it is also advantageous to consider curves that exhibit the PH property under certain special, non-Euclidean metrics.

17.6.1. Minkowski metric of special relativity

The *Minkowski metric* of special relativity characterizes the distance between points in a “pseudo-Euclidean” *space-time*, spanned by one temporal and n spatial dimensions. With $n = 2$, for example, the distance d between the two points or “events” (x_1, y_1, t_1) and (x_2, y_2, t_2) is given by

$$d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2 - c^2(t_2 - t_1)^2, \quad (17.34)$$

where c is the speed of light. Events are said to have *space-like*, *time-like*, or *light-like* separation, according to whether $d^2 > 0$, $d^2 < 0$, or $d^2 = 0$. It is convenient to employ time and distance units in which $c = 1$: the Minkowski metric (17.34) then differs from

the usual metric $d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2$ of Euclidean space (x, y, z) only in the *subtraction*, rather than the addition, of the last term: we say that the Euclidean and Minkowski metric have *signatures* $(+++)$ and $(++-)$, respectively.

Moon [50,51] showed that Pythagorean hodographs in the Minkowski metric are very useful in recovering the boundary of a planar domain \mathcal{D} from its *medial axis transform* (MAT). The medial axis is the locus of centers of maximal disks, touching the boundary $\partial\mathcal{D}$ in at least two points, that can be inscribed within the domain \mathcal{D} . If $\mathbf{c}(t) = (x(t), y(t))$ is a parameterization of the medial axis, we may superpose a *radius function* $r(t)$ on it, specifying the size of maximal disks centered on $\mathbf{c}(t)$: the MAT is the three-dimensional locus $(x(t), y(t), r(t))$. Introducing the unit vector

$$\mathbf{m}(t) = \frac{(-r'(t)x'(t) \mp \ell(t)y'(t), -r'(t)y'(t) \pm \ell(t)x'(t))}{x'^2(t) + y'^2(t)},$$

where $\ell(t) = \sqrt{x'^2(t) + y'^2(t) - r'^2(t)}$ is the parametric speed of the MAT in the Minkowski metric, the envelope of the one-parameter family of circles described by $(x(t), y(t), r(t))$ has the parameterization [50]:

$$\mathbf{e}(t) = \mathbf{c}(t) + r(t)\mathbf{m}(t).$$

Clearly, $\mathbf{e}(t)$ is not a rational locus, unless we ensure that the MAT hodograph satisfies the (Minkowski) Pythagorean condition

$$x'^2(t) + y'^2(t) - r'^2(t) = \sigma^2(t) \tag{17.35}$$

for some polynomial $\sigma(t)$. Moon has shown that a MAT hodograph of the form

$$\begin{aligned} x'(t) &= u^2(t) - v^2(t) - p^2(t) + q^2(t), \\ y'(t) &= 2u(t)p(t) - 2v(t)q(t), \\ r'(t) &= 2u(t)q(t) - 2v(t)p(t), \end{aligned} \tag{17.36}$$

and hence $\sigma(t) = u^2(t) - v^2(t) + p^2(t) - q^2(t)$, is a sufficient and necessary condition [51] for the satisfaction of equation (17.35). Thus, MATs with hodographs of the form (17.36) are called *Minkowski Pythagorean-hodograph* (MPH) *curves*.¹¹

Note that, apart from signs, the hodographs (17.10) and (17.36) for PH space curves and MPH medial axis transforms have essentially the same structure — the sign differences ensure satisfaction of the Pythagorean conditions (17.8) and (17.35) under Euclidean and Minkowski metrics; see also [7]. The reconstruction of a rational domain boundary (and of rational offsets to the boundary) from an MPH MAT is discussed in [6].

17.6.2. Minkowski metric defined by convex indicatrix

A generalization of the PH property to a different non-Euclidean metric, also associated with the mathematician/physicist Hermann Minkowski, was introduced by Ait Haddou et al. in [2]. They consider the geometry of the *Minkowski plane* whose metric is defined

¹¹It is universally agreed, even in Europe, that the speed σ of an MPH curve should always be specified in miles-per-hour (never kilometers-per-hour).

by choosing as “unit circle” a strictly-convex, centrally-symmetric locus \mathcal{U} . In terms of this *indicatrix*, the distance between points \mathbf{x} and \mathbf{y} is given by

$$d_{\mathcal{U}}(\mathbf{x}, \mathbf{y}) = 2 \frac{\|\mathbf{x} - \mathbf{y}\|}{\|\mathbf{x}' - \mathbf{y}'\|}, \quad (17.37)$$

where $\mathbf{x}' - \mathbf{y}'$ is the diameter of \mathcal{U} parallel to $\mathbf{x} - \mathbf{y}$, and $\|\cdot\|$ is the usual Euclidean metric. Ait Haddou et al. give characterizations of curves whose \mathcal{U} -offsets under the metric (17.37) are rational: they call such curves *Minkowski isoperimetric-hodograph curves*.

17.7. CLOSURE

By virtue of their special algebraic structure, Pythagorean-hodograph curves provide *exact* solutions to a number of basic geometrical problems in computer-aided design and manufacturing. Apart from the issues of accuracy and data volume, the primary attraction of such exact solutions lies in the enhanced *robustness* of their software embodiment.

Since its inception [30] about a decade ago, the Pythagorean hodograph concept and its various extensions and generalizations have been remarkably fertile fields for further research and practical applications. We have only been able to sketch a bare outline of all these developments, and it seems fitting to conclude by citing the “grand unified theory” of PH curves developed by Choi et al. [7], which employs the Clifford algebra perspective to gain insight into the algebraic structures — such as (17.4), (17.10), and (17.36) — incurred by the PH condition in various spaces of practical interest.

REFERENCES

1. R. Ait Haddou and L. Biard. G^2 approximation of an offset curve by Tschirnhausen quartics. In M. Daehlen, T. Lyche, and L. L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, Vanderbilt Univ. Press, pages 1–10, 1995.
2. R. Ait Haddou, L. Biard, and M. A. Slawinsky. Minkowski isoperimetric-hodograph curves. *Computer Aided Geometric Design*, 17:835–861, 2000.
3. G. Albrecht and R. T. Farouki. Construction of C^2 Pythagorean-hodograph interpolating splines by the homotopy method. *Advances in Computational Mathematics*, 5:417–442, 1996.
4. E. L. Allgower and K. Georg. *Numerical Continuation Methods: An Introduction*. Springer, Berlin, 1990.
5. H. Bruyninckx and D. Reynaerts. Path planning for mobile and hyper-redundant robots using Pythagorean-hodograph curves. *Proceedings, International Conference on Advanced Robotics*, Monterey, CA, pages 595–600, 1997.
6. H. I. Choi, C. Y. Han, H. P. Moon, K. H. Roh, and N-S. Wee. Medial axis transform and offset curves by Minkowski Pythagorean hodograph curves. *Computer-Aided Design*, 31:59–72, 1999.
7. H. I. Choi, D. S. Lee, and H. P. Moon. Clifford algebra, spin representation, and rational parameterization of curves and surfaces. *Advances in Computational Mathematics*, to appear, 2002.
8. D. J. de Solla Price. The Babylonian “Pythagorean triangle” table. *Centaurus*, 10:219–231, 1964.

9. R. Dietz, J. Hoschek, and B. Jüttler. An algebraic approach to curves and surfaces on the sphere and other quadrics. *Computer Aided Geometric Design*, 10:211–229, 1993.
10. G. Elber, I.-K. Lee, and M.-S. Kim. Comparing offset curve approximation methods. *IEEE Computer Graphics and Applications*, 17(3):62–71, 1997.
11. R. T. Farouki. Pythagorean–hodograph curves in practical use. In R. E. Barnhill, editor, *Geometry Processing for Design and Manufacturing*, SIAM, pages 3–33, 1992.
12. R. T. Farouki. The conformal map $z \rightarrow z^2$ of the hodograph plane. *Computer Aided Geometric Design*, 11:363–390, 1994.
13. R. T. Farouki. The elastic bending energy of Pythagorean–hodograph curves. *Computer Aided Geometric Design*, 13:227–241, 1996.
14. R. T. Farouki. Pythagorean–hodograph quintic transition curves of monotone curvature. *Computer-Aided Design*, 29:601–606, 1997.
15. R. T. Farouki. Convergent inversion approximations for polynomials in Bernstein form. *Computer Aided Geometric Design*, 17:179–196, 2000.
16. R. T. Farouki and J.-C. A. Chastang. Curves and surfaces in geometrical optics. In T. Lyche and L. L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design II*, Academic Press, Boston, pages 239–260, 1992.
17. R. T. Farouki, M. al-Kandari, and T. Sakkalis. Structural invariance of spatial Pythagorean hodographs. *Computer Aided Geometric Design*, to appear, 2002.
18. R. T. Farouki, M. al-Kandari, and T. Sakkalis. Hermite interpolation by rotation-invariant spatial Pythagorean–hodograph curves. *Advances in Computational Mathematics*, to appear, 2002.
19. R. T. Farouki, B. K. Kuspa, C. Manni, and A. Sestini. Efficient solution of the complex quadratic tridiagonal system for C^2 PH quintic splines. *Numerical Algorithms*, 27:35–60, 2001.
20. R. T. Farouki, J. Manjunathaiah, and S. Jee. Design of rational cam profiles with Pythagorean–hodograph curves. *Mechanism and Machine Theory*, 33:669–682, 1998.
21. R. T. Farouki, J. Manjunathaiah, D. Nicholas, G.-F. Yuan, and S. Jee. Variable feedrate CNC interpolators for constant material removal rates along Pythagorean–hodograph curves. *Computer-Aided Design*, 30:631–640, 1998.
22. R. T. Farouki, J. Manjunathaiah, and G.-F. Yuan. G codes for the specification of Pythagorean–hodograph tool paths and associated feedrate functions on open-architecture CNC machines. *International Journal of Machine Tools and Manufacture*, 39:123–142, 1999.
23. R. T. Farouki and C. A. Neff. Analytic properties of plane offset curves & Algebraic properties of plane offset curves. *Computer Aided Geometric Design*, 7:83–99 & 101–127, 1990.
24. R. T. Farouki and C. A. Neff. Hermite interpolation by Pythagorean–hodograph quintics. *Mathematics of Computation*, 64:1589–1609, 1995.
25. R. T. Farouki and J. Peters. Smooth curve design with double-Tschirnhausen cubics. *Annals of Numerical Mathematics*, 3:63–82, 1996.
26. R. T. Farouki and H. Pottmann. Polynomial and rational Pythagorean–hodograph curves reconciled. In G. Mullineux, editor, *The Mathematics of Surfaces VI*, Oxford

- Univ. Press, pages 355–378, 1996.
27. R. T. Farouki and V. T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4:191–216, 1987.
 28. R. T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5:1–26, 1988.
 29. R. T. Farouki, K. Saitou, and Y-F. Tasi. Least-squares tool path approximation with Pythagorean-hodograph curves for high-speed CNC machining. In R. Cripps, editor, *The Mathematics of Surfaces VIII*, Information Geometers, Winchester, pages 245–264, 1998.
 30. R. T. Farouki and T. Sakkalis. Pythagorean hodographs. *IBM Journal of Research and Development*, 34:736–752, 1990.
 31. R. T. Farouki and T. Sakkalis. Real rational curves are not “unit speed.” *Computer Aided Geometric Design*, 8:151–157, 1991.
 32. R. T. Farouki and T. Sakkalis. Pythagorean-hodograph space curves. *Advances in Computational Mathematics*, 2:41–66, 1994.
 33. R. T. Farouki and T. W. Sederberg. Analysis of the offset to a parabola. *Computer Aided Geometric Design*, 12:639–645, 1995.
 34. R. T. Farouki and S. Shah. Real-time CNC interpolators for Pythagorean-hodograph curves. *Computer Aided Geometric Design*, 13:583–600, 1996.
 35. R. T. Farouki and Y-F. Tsai. Exact Taylor series coefficients for variable-feedrate CNC curve interpolators. *Computer-Aided Design*, 33:155–165, 2001.
 36. R. T. Farouki, Y-F. Tsai, and G-F. Yuan. Contour machining of free-form surfaces with real-time PH curve CNC interpolators. *Computer Aided Geometric Design*, 16:61–76, 1998.
 37. R. T. Farouki, Y-F. Tsai, and C. S. Wilson. Physical constraints on feedrates and feed accelerations along curved tool paths. *Computer Aided Geometric Design*, 17:337–359, 2000.
 38. J. C. Fiorot and T. Gensane. Characterization of the set of rational curves with rational offsets. In P. J. Laurent, A. Le Méhauté, and L. L. Schumaker, editors, *Curves and Surfaces in Geometric Design*, AK Peters, pages 153–160, 1994.
 39. J. Hoschek. Dual Bézier curves and surfaces. In R. E. Barnhill and W. Boehm, editors, *Surfaces in Computer Aided Geometric Design*, North Holland, pages 147–156, 1983.
 40. B. Jüttler. Generating rational frames of space curves via Hermite interpolation with Pythagorean hodograph cubic splines. In *Geometric Modeling and Processing '98*, Bookplus Press, pages 83–106, 1998.
 41. B. Jüttler. Triangular Bézier surface patches with a linear normal vector field. In R. Cripps, editors, *The Mathematics of Surfaces VIII*, Information Geometers, Winchester, pages 431–446, 1998.
 42. B. Jüttler. Hermite interpolation by Pythagorean hodograph curves of degree seven. *Mathematics of Computation*, to appear, 2000.
 43. B. Jüttler and C. Mäurer. Cubic Pythagorean hodograph spline curves and applications to sweep surface modeling. *Computer-Aided Design*, 31:73–83, 1999.
 44. B. Jüttler and M. L. Sampoli. Hermite interpolation by piecewise polynomial surfaces with rational offsets. *Computer Aided Geometric Design*, 17:361–385, 2000.

45. L. V. Kantorovich and G. P. Akilov. *Functional Analysis* (2nd edition). Pergamon Press, Oxford, 1982.
46. K. K. Kubota. Pythagorean triples in unique factorization domains. *American Mathematical Monthly*, 79:503–505, 1972.
47. W. Lü. Rationality of the offsets to algebraic curves and surfaces. *Applied Mathematics*, 9(Ser. B):265–278, 1994.
48. W. Lü. Offset–rational parametric plane curves. *Computer Aided Geometric Design*, 12:601–616, 1995.
49. W. Lü and H. Pottmann. Pipe surfaces with rational spine curve are rational. *Computer Aided Geometric Design*, 13:621–628, 1996.
50. H. P. Moon. Computing rational offsets via medial axis transform using polynomial speed curves in $\mathbf{R}^{2,1}$. *Geometric Modeling and Processing '98*, Bookplus Press, pages 187–203, 1998.
51. H. P. Moon. Minkowski Pythagorean hodographs. *Computer Aided Geometric Design*, 16:739–753, 1999.
52. H. P. Moon, R. T. Farouki, and H. I. Choi. Construction and shape analysis of PH quintic Hermite interpolants. *Computer Aided Geometric Design*, 18:93–115, 2001.
53. A. P. Morgan. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems*. Prentice–Hall, Englewood Cliffs, NJ, 1987.
54. O. Neugebauer and A. J. Sachs. *Mathematical Cuneiform Texts*. American Oriental Series Vol. 29, American Oriental Society, New Haven, 1945.
55. J. M. Ortega. The Newton–Kantorovich theorem. *American Mathematical Monthly*, 75:658–660, 1968.
56. M. Peternell and H. Pottmann. Designing rational surfaces with rational offsets. In F. Fontanella, K. Jetter, and P. J. Laurent, editors, *Advanced Topics in Multivariate Approximation*, World Scientific Press, pages 286, 1996.
57. M. Peternell and H. Pottmann. Computing rational parameterizations of canal surfaces. *Journal of Symbolic Computation*, 23:255–266, 1997.
58. M. Peternell and H. Pottmann. A Laguerre geometric approach to rational offsets. *Computer Aided Geometric Design*, 15:223–249, 1998.
59. B. Pham. Offset curves and surfaces: a brief survey. *Computer-Aided Design*, 24:223–229, 1992.
60. H. Pottmann. Applications of the dual Bézier representation of rational curves and surfaces. In P. J. Laurent, A. Le Méhauté, and L. L. Schumaker, editors, *Curves and Surfaces in Geometric Design*, AK Peters, pages 377–384, 1994.
61. H. Pottmann. Rational curves and surfaces with rational offsets. *Computer Aided Geometric Design*, 12:175–192, 1995.
62. H. Pottmann. Curve design with rational Pythagorean–hodograph curves. *Advances in Computational Mathematics*, 3:147–170, 1995.
63. H. Pottmann and M. Peternell. Applications of Laguerre geometry in CAGD. *Computer Aided Geometric Design*, 15:165–186, 1998.
64. E. J. F. Primrose. *Plane Algebraic Curves*. Macmillan, London, 1955.
65. J. Roe. *Elementary Geometry*. Oxford Univ. Press, 1993.
66. M. Sabin. Rational speed pseudo–quadratic B–splines, In A. Le Méhauté, C. Rabut, and L. L. Schumaker, editors, *Curves and Surfaces with Applications in CAGD*,

- Vanderbilt Univ. Press, pages 395–402, 1997.
67. T. Sakkalis and R. T. Farouki. Algebraically rectifiable parametric curves. *Computer Aided Geometric Design*, 10:551–569, 1993.
 68. R. Schickentanz. Interpolating G^1 splines with rational offsets. *Computer Aided Geometric Design*, 14:881–885, 1997.
 69. R. Schickentanz. *Interpolation und Approximation durch rationale B-Spline Flächen mit rationalen Offsets*. Dissertation, Technische Universität Darmstadt, 1999.
 70. R. A. Tapia. The Kantorovich theorem for Newton's method. *American Mathematical Monthly*, 78:389–392, 1971.
 71. Y-F. Tsai, R. T. Farouki, and B. Feldman. Performance analysis of CNC interpolators for time-dependent feedrates along PH curves. *Computer Aided Geometric Design*, 18:245–265, 2001.
 72. K. Ueda. Deformation of plane curves preserving Pythagorean hodographs. *Proceedings of the 1997 IEEE Conference on Information Visualization*, IEEE Computer Society, pages 71–76, 1997.
 73. K. Ueda. Spherical Pythagorean-hodograph curves. In M. Daehlen, T. Lyche, and L. L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces II*, Vanderbilt Univ. Press, pages 485–492, 1998.
 74. K. Ueda. Pythagorean-hodograph curves on isothermal surfaces. In R. Cripps, editor, *The Mathematics of Surfaces VIII*, Information Geometers, Winchester, pages 339–353, 1998.
 75. K. Ueda. Helical curves over Pythagorean-hodograph curves. In *Geometric Modeling and Processing '98*, Bookplus Press, pages 115–128, 1998.
 76. K. Ueda. Pythagorean-hodograph space curves by quaternion calculus, preprint, 1998.
 77. M. G. Wagner and B. Ravani. Curves with rational Frenet–Serret motion. *Computer Aided Geometric Design*, 15:79–101, 1997.
 78. D. J. Walton and D. S. Meek. A Pythagorean-hodograph spiral. *Computer-Aided Design*, 29:943–950, 1996.
 79. D. J. Walton and D. S. Meek. G^2 curves composed of planar cubic Pythagorean hodograph quintic spirals. *Computer Aided Geometric Design*, 15:547–566, 1998.
 80. C. Zwikker. *The Advanced Geometry of Plane Curves and Their Applications*. Dover, New York (reprint), 1963.

Chapter 18

Voronoi Diagrams

Kokichi Sugihara

This chapter surveys the Voronoi diagram and related topics. First, the most primitive version of the Voronoi diagram is defined, and its basic properties together with the algorithms for constructing it are summarized. Next possible applications of the Voronoi diagram are presented, where special emphasis is placed on offsetting and interpolations. Finally the Voronoi diagram is generalized in various directions; they include generalization of the metric and the generalization of the generators.

18.1. ORDINARY VORONOI DIAGRAM

The topic in this chapter is a partition of a space into territories, which is called a ‘Voronoi diagram’. This concept is so natural that it was rediscovered in many areas of science independently. Actually it has many names including a ‘Voronoi diagram’, ‘Dirichlet domain’, ‘Thiessen polygons’, ‘plesiohedra’, ‘fundamental areas’, and ‘domain of action’. Refer to Okabe et al. (2000) for the history of this concept. In this chapter we use the most familiar name ‘Voronoi diagram’.

Suppose that we are given a set $S = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$ of n points in the d -dimensional space \mathbf{R}^d . For each \mathbf{p}_i , the region $R(S; \mathbf{p}_i)$ defined by

$$R(S; \mathbf{p}_i) = \{\mathbf{p} \in \mathbf{R}^d \mid \|\mathbf{p} - \mathbf{p}_i\| < \|\mathbf{p} - \mathbf{p}_j\|, \quad \text{for } j \neq i\} \quad (18.1)$$

is called the **Voronoi region** of \mathbf{p}_i , where $\|\mathbf{p} - \mathbf{q}\|$ represents the Euclidean distance between the two points \mathbf{p} and \mathbf{q} . That is, the region $R(S; \mathbf{p}_i)$ consists of the points that are closer to \mathbf{p}_i than to any other point in S . (Georges Voronoi was a mathematician who studied this geometric structure [32].)

The d -dimensional space \mathbf{R}^d is partitioned into the n regions $R(S; \mathbf{p}_1), R(S; \mathbf{p}_2), \dots, R(S; \mathbf{p}_n)$ and their boundaries. This partition is called the **Voronoi diagram** of S , and is denoted by $\text{VD}(S)$. The points in S are called the **generating points** for $\text{VD}(S)$.

Figure 18.1 represents the Voronoi diagram in \mathbf{R}^2 , where the small filled circles represent the points in S , and the solid lines represent the boundaries of the Voronoi regions. As

shown in this figure, the boundary of two Voronoi regions is a part of a line that is the perpendicular bisector of the associated two points. This boundary is called the **Voronoi edge**. The point where three or more Voronoi edges meet is called the **Voronoi point**.

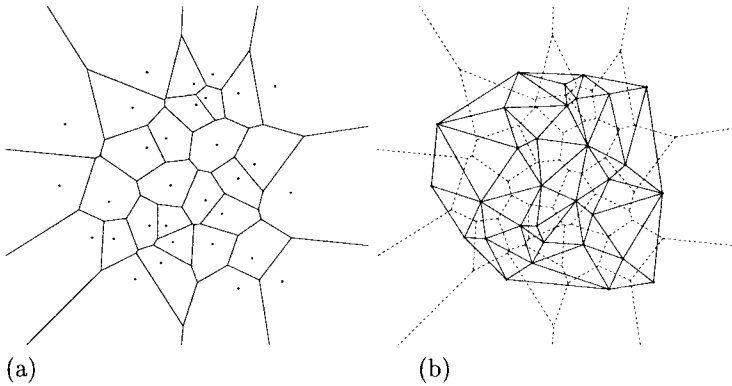


Figure 18.1. A Voronoi diagram in \mathbf{R}^2 .

In the general d -dimensional space \mathbf{R}^d , the boundary of two Voronoi regions is part of the $(d - 1)$ -dimensional hyperplane that is the perpendicular bisector of the associated two generating points.

The Voronoi region $R(S; \mathbf{p}_i)$ can be written as

$$R(S; \mathbf{p}_i) = \bigcap_{j \neq i} \{ \mathbf{p} \in \mathbf{R}^d \mid \|\mathbf{p} - \mathbf{p}_i\| < \|\mathbf{p} - \mathbf{p}_j\| \}; \quad (18.2)$$

that is, $R(S; \mathbf{p}_i)$ is the intersection of the halfspaces bounded by the hyperplanes $\|\mathbf{p} - \mathbf{p}_i\| = \|\mathbf{p} - \mathbf{p}_j\|$. Hence, the Voronoi region is a convex polyhedron.

For a set S of points in \mathbf{R}^d , the smallest convex region containing S is called the **convex hull** of S , and is denoted by $\text{CH}(S)$. The convex hull $\text{CH}(S)$ is a bounded convex polyhedron; some elements of S are on the boundary of $\text{CH}(S)$ and the others are in its interior. The Voronoi region $R(S; \mathbf{p}_i)$ is unbounded if and only if \mathbf{p}_i is on the boundary of $\text{CH}(S)$.

As shown in Figure 18.1(a), each Voronoi point in \mathbf{R}^2 is usually incident to three Voronoi regions. However, as shown in Figure 18.2, if four or more generating points are on a common circle and no other generating points are in the interior of the circle, there arises a Voronoi point that is incident to four or more Voronoi regions. This situation is called **degeneracy**, and S or $\text{VD}(S)$ is said to be **degenerate**.

In a general d -dimensional Voronoi diagram, a Voronoi point is usually incident to $d + 1$ Voronoi regions. However, degeneracy occurs when $d + 2$ or more generating points are on a common hypersphere and no other generating points are in the interior. In this case, the Voronoi regions of those cospherical generating points are incident to a common Voronoi point.

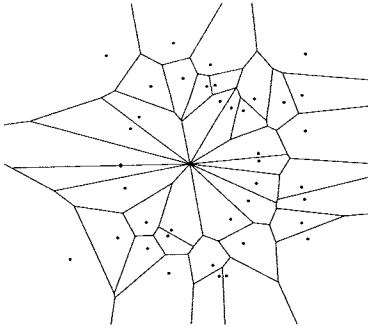


Figure 18.2. A Voronoi diagram for degenerate input.

18.2. DELAUNAY DIAGRAM

Given the Voronoi diagram $VD(S)$ in \mathbf{R}^2 , we connect two points \mathbf{p}_i and \mathbf{p}_j in S by a line segment if and only if $R(S; \mathbf{p}_i)$ and $R(S; \mathbf{p}_j)$ have a common Voronoi edge; in Figure 18.1(b), the solid lines represent line segments constructed in this way, where the original Voronoi diagram is represented by broken lines. Those line segments are called **Delaunay edges**. Thus we obtain another diagram, which partitions the convex hull $CH(S)$ into convex polygons, called **Delaunay polygons**, and their boundaries. This partitioning is called the **Delaunay diagram** of S , and is denoted by $DD(S)$. (Boris Delaunay was a mathematician who studied this geometric structure [7].)

If S is not degenerate, then all the Delaunay polygons are triangles, as shown in Figure 18.1(b). In this case, the Delaunay diagram is called the **Delaunay triangulation**. If S is degenerate, on the other hand, then the Delaunay polygons are not necessarily triangles. However, we can subdivide the nontriangular polygons into triangles by inserting additional diagonal line segments. The resulting partition of $CH(S)$ into triangles is also called the Delaunay triangulation for S .

There is a one-to-one correspondence between the Voronoi points and the Delaunay polygons. Actually a Delaunay polygon is the convex hull of the generating points whose Voronoi regions are incident to the associated Voronoi point. There is further one-to-one correspondence between the Voronoi edges and the Delaunay edges; the corresponding edges are perpendicular to each other.

The correspondence between Voronoi points and the Delaunay polygons can be extended to an arbitrary dimensionality. Suppose that a Voronoi diagram $VD(S)$ is given in \mathbf{R}^d . For each Voronoi point, the convex hull of the generating points whose Voronoi regions are incident to the Voronoi point is called the Delaunay polyhedron corresponding to the Voronoi point. The Delaunay polyhedra corresponding to all the Voronoi points partition $CH(S)$. This partitioning is called the **Delaunay diagram** of S .

A Delaunay polyhedron has the property that all the vertices are on a common hypersphere, and there is no other generator in the interior of the common hypersphere.

If S is not degenerate, then the Delaunay polyhedron is the convex hull of $d + 1$ gen-

erators, and hence it is a d -dimensional simplex. In this case, the Delaunay diagram is called the Delaunay triangulation; in particular, the Delaunay triangulation in \mathbf{R}^3 is also called the Delaunay tetrahedralization.

18.3. BASIC PROPERTIES OF THE VORONOI AND DELAUNAY DIAGRAMS

Let us concentrate on the two-dimensional Voronoi and Delaunay diagrams. Let n denote the number of generating points: $n = |S|$. Let e denote the number of Voronoi edges and v denote the number of Voronoi points. Furthermore, let c denote the number of the generating points that are on the boundary of $\text{CH}(S)$.

Note that, in the Delaunay diagram terminology, e denotes the number of Delaunay edges, and v denotes the number of Delaunay polygons. Since a Delaunay polygon has at least three edges, we get

$$c + 3v \leq 2e. \quad (18.3)$$

On the other hand, from Euler's formula we get

$$n - e + v + 1 = 2. \quad (18.4)$$

From (18.3) and (18.4), we obtain

$$e \leq 3n - c - 3, \quad (18.5)$$

$$v \leq 2n - c - 2. \quad (18.6)$$

Hence $e = O(n)$ and $v = O(n)$. Therefore, the complexity (i.e., the total number of geometric elements) of the Voronoi diagram is linear in the number of the generating points. That is, the complexity of the Voronoi diagram is $O(n)$. This means that memory requirement for a Voronoi diagram in \mathbf{R}^2 is linear in n . Because of this property, the Voronoi diagram in \mathbf{R}^2 is particularly useful as a basic data structure for designing many geometric algorithms.

In general, the complexity of the d -dimensional Voronoi/Delaunay diagram is $O(n^{\lfloor (d+1)/2 \rfloor})$, where $\lfloor x \rfloor$ denotes the smallest integer greater than or equal to x .

One of the most important properties of the Delaunay triangulation in \mathbf{R}^2 is that it gives an optimal triangulation, in the sense that the smallest angle among all the triangles is the largest. Consider four nondegenerate generating points forming a convex quadrilateral, as shown in Figure 18.3. There are two possible diagonals which will create a triangulation, one of which gives the Delaunay triangulation of the four points. In Figure 18.3, the solid diagonal gives the Delaunay triangulation while the dotted diagonal gives the other triangulation. Let angle a in the figure be the smallest among the six angles of two triangles in the Delaunay triangulation. Then the triangle containing this angle has the circumcircle that does not contain the other generating point. Therefore, angle b , which is an angle of the other triangulation, is smaller than a . Thus, the Delaunay triangulation of four generating points has a larger smallest angle than the other triangulation. This property holds in the general Delaunay triangulation in the following sense.

The collection T of triangles with vertices in S is called a triangulation spanning S if (i) the union of all the triangles coincides with the convex hull $\text{CH}(S)$, (ii) the intersection of

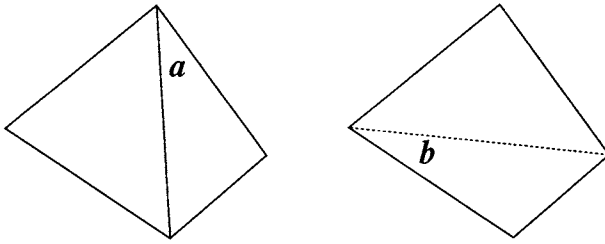


Figure 18.3. Two possible diagonals of a convex quadrilateral.

any two triangles is empty, or a point or a line segment, and (iii) when any two triangles share a common line segment, this line segment coincides with an edge of either triangle. Let $\theta(T) = (\theta_1, \theta_2, \dots, \theta_k)$ be the list of the angles of all the triangles in T , arranged in increasing order: that is, θ_1 is the smallest, θ_2 is the second smallest, and so on. Let $\theta(T') = (\theta'_1, \theta'_2, \dots, \theta'_k)$ be the list of angles defined similarly for another triangulation T' spanning S . We say that $\theta(T)$ is **lexicographically smaller** than $\theta(T')$, if there is m ($1 \leq m \leq k$) such that $\theta_1 = \theta'_1, \theta_2 = \theta'_2, \dots, \theta_{m-1} = \theta'_{m-1}$ and $\theta_m < \theta'_m$. Then the next property holds, which is called the equi-angularity property [27].

Property 3.1 (Equi-angularity). Among all the triangulations spanning S in \mathbf{R}^2 , the Delaunay triangulation gives the lexicographically largest list $\theta(T)$ of angles.

Because of this property, the Delaunay triangulation avoids triangles with small angles as much as possible. This property is useful for many applications such as interpolation based on the triangulation and finite-element mesh generation for the solution of partial differential equations [5,14].

The equi-angularity property is peculiar to the two dimensional case. It does not hold in three or higher-dimensional space. Because of this fact, the Delaunay triangulation in an arbitrary dimension is not straightforward to use as in two-dimensional space.

Detailed discussion of the properties of the Voronoi and Delaunay diagrams can be found in [3,9,13,22].

18.4. ALGORITHMS

For a given set S of generating points, constructing the Voronoi diagram and the Delaunay diagram are equivalent, because each diagram can easily be obtained from the other. Here we will concentrate on the algorithm for constructing the Delaunay diagram.

Many algorithms have been proposed for this purpose. They include the divide-and-conquer method [23], the plane sweep method [12], the incremental method [21], and the lift-up method [6,11]. The lift-up method is most general in the sense it can be applied to any dimension.

Suppose that the set S of generating points in \mathbf{R}^2 is not degenerate. Then, three generating points $\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k$ form a Delaunay triangle if and only if the circle passing through the three points has no point from S in its interior. Let $\mathbf{p} = (x, y)$ be a general

point, and let us define $F(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k, \mathbf{p})$ by

$$F(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k, \mathbf{p}) = \begin{vmatrix} 1 & x_i & y_i & x_i^2 + y_i^2 \\ 1 & x_j & y_j & x_j^2 + y_j^2 \\ 1 & x_k & y_k & x_k^2 + y_k^2 \\ 1 & x & y & x^2 + y^2 \end{vmatrix}. \quad (18.7)$$

The equation $F(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k, \mathbf{p}) = 0$ represents a circle because its quadratic term is $c(x^2 + y^2)$ for some constant c . Moreover, this circle passes through $\mathbf{p}_i, \mathbf{p}_j$ and \mathbf{p}_k , because if we substitute these points to \mathbf{p} , the associated matrix contains two identical rows and hence the determinant vanishes. Therefore, the sign of $F(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k, \mathbf{p})$ changes according to whether \mathbf{p} is inside this circle or not.

Suppose that the plane \mathbf{R}^2 is assigned a counterclockwise coordinate system, and $\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k$ are on the common circle in counterclockwise order. Then, they form a Delaunay triangulation if and only if

$$F(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k, \mathbf{p}_l) \equiv \begin{vmatrix} 1 & x_i & y_i & x_i^2 + y_i^2 \\ 1 & x_j & y_j & x_j^2 + y_j^2 \\ 1 & x_k & y_k & x_k^2 + y_k^2 \\ 1 & x_l & y_l & x_l^2 + y_l^2 \end{vmatrix} > 0 \quad (18.8)$$

holds for all other generating points \mathbf{p}_l .

For each $i = 1, 2, \dots, n$, let \mathbf{p}_i^* be the point in \mathbf{R}^3 defined by

$$\mathbf{p}_i^* = (x_i, y_i, x_i^2 + y_i^2), \quad (18.9)$$

as shown in Figure 18.4. That is, \mathbf{p}_i^* is the point obtained by lifting the point \mathbf{p}_i up to the surface of revolution of the parabola $z = x^2 + y^2$. Then, the inequality (18.8) implies that the point \mathbf{p}_l^* is above the plane passing through $\mathbf{p}_i^*, \mathbf{p}_j^*$ and \mathbf{p}_k^* . Hence, the Delaunay triangulation for S is obtained if the lower part of the three-dimensional convex hull for the lifted points $\mathbf{p}_1^*, \mathbf{p}_2^*, \dots, \mathbf{p}_n^*$ is projected on to the xy -plane orthogonally. Thus, the construction of the two-dimensional Delaunay triangulation is reduced to the construction of the convex hull in a space with dimensionality increased by one.

The three-dimensional convex hull of n points can be constructed in $O(n \log n)$ time by the divide-and-conquer algorithm [24]. Consequently, the two-dimensional Delaunay diagram, and thus the two-dimensional Voronoi diagram, can be constructed in $O(n \log n)$ time.

The same scheme is valid for any dimensionality. In other words, for a set $S = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$ of generating points in \mathbf{R}^d , we define the set $S^* = \{\mathbf{p}_1^*, \mathbf{p}_2^*, \dots, \mathbf{p}_n^*\}$ in such a way that the point \mathbf{p}_i^* is obtained by lifting \mathbf{p}_i up on to the surface of revolution of a parabola $x_{n+1} = x_1^2 + x_2^2 + \dots + x_n^2$. Then, the orthographic projection of the $(d+1)$ -dimensional convex hull of the set $\{\mathbf{p}_1^*, \mathbf{p}_2^*, \dots, \mathbf{p}_n^*\}$ gives the Delaunay triangulation. The d -dimensional convex hull ($d \geq 3$) is obtained in $O(n^{\lfloor (d+1)/2 \rfloor})$ time [9]. Therefore, the d -dimensional Delaunay and Voronoi diagrams for n points can be constructed in $O(n^{\lfloor (d+2)/2 \rfloor})$ time.

For practical implementation robustness is also important: if we ignore robustness, the resulting algorithms fail because of inconsistency caused by numerical errors. Many robust algorithms have been proposed: they include topology-oriented methods [30,31], exact arithmetic methods [18] and symbolic perturbation [10,34].

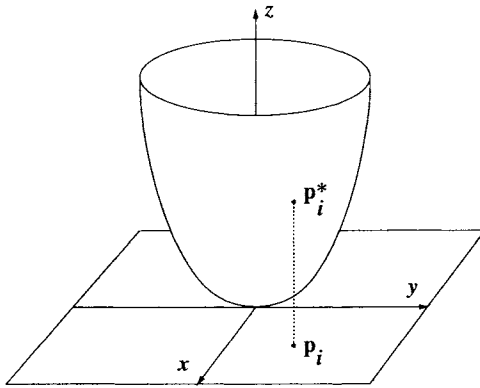


Figure 18.4. Lifting the generating points up to the surface of revolution of a parabola.

18.5. APPLICATIONS

18.5.1. Site retrieval

Suppose that a person is injured by a traffic accident, and that he needs medical care. Which ambulance should go to help him? This problem can be solved efficiently by the Voronoi diagram.

Let S be the set of points at which ambulances are located. We construct the Voronoi diagram $VD(S)$ beforehand, and store it. Then we can retrieve the nearest ambulance just by finding the Voronoi region containing the locus of the traffic accident.

Similarly, if we construct another Voronoi diagram whose generating points are the sites of hospitals, then we can find the nearest hospital by identifying the Voronoi region which contains the locus of the accident.

Given a Voronoi diagram and a point, finding the Voronoi region containing the point is a typical point location problem. This problem can be solved by the “slab method” in $O(\log n)$ time if we modify the Voronoi diagram into a slab data structure [24].

Thus, nearest-site retrieval among n sites can be done in $O(\log n)$ time if we construct the Voronoi diagram for the sites, and appropriately preprocess the data structure of the Voronoi diagram [21,30]. This is a basic technique in geographic information retrieval systems.

18.5.2. Medial axis

Let A be a two-dimensional bounded region. Suppose that we want to extract a concise description of the rough shape of A . A typical way is to replace A by a collection of lines, just as we might describe an animal by its skeleton. Such a collection of line segments is called the **medial axis**, which is defined in the following way.

As shown in Figure 18.5(a), let C be a circle touching the boundary of A at two or more points. If C moves continuously, keeping in touch with the boundary of A at two or more points, then the center of C traces along (generally curved) lines. The collection

of such lines in A is called the medial axis of A . Refer to the chapter 19 on ‘Medial Axis Transform’ by Choi and Han.

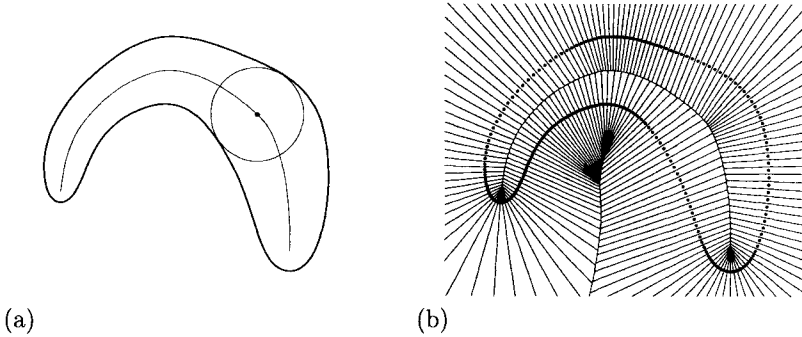


Figure 18.5. Extraction of the medial axis using the Voronoi diagram.

The Voronoi diagram can be used to extract the medial axis. For a given region A , let S be a finite set of points densely sampled on the boundary of A . We construct the Voronoi diagram of S , as shown in Figure 18.5(b) Voronoi edges are the set of points that are closest to at least two points in S . Hence, they are candidate elements for the medial axis. From them we select those Voronoi edges that are in A and are also boundaries of Voronoi regions whose generating points are not near to each other along the boundary of A . The medial axis in Figure 18.5(a) was obtained by this method.

For a three-dimensional region A , the medial surface is similarly defined. The **medial surface** is the set of points that are the centers of spheres (of various radii) touching three or more points on the boundary of A . The three-dimensional Voronoi diagram can be used for the extraction of the medial surface.

The medial axis and the medial surface have many applications; refer to the chapter 19 on the Medial Axis Transform by Choi and Han.

18.5.3. Offset curves and surfaces

Let A be a bounded region in \mathbf{R}^2 . For positive real h , the **offset curve** with offset h is the set of points whose distance from A is h . Similarly, for a bounded region A in \mathbf{R}^3 , the **offset surface** with offset h is the set of points whose distance from A is h .

The offset curves and surfaces are useful, for example, in finding the path of the center of a ball cutter in NC machining [15]. Also refer to the chapter 22 on ‘NC Machining’ by Choi et al.

The Voronoi diagram can be used to generate offset curves and surfaces. Let A be a region in \mathbf{R}^2 . We first locate sufficiently many sample points on the boundary of A . Let S denote the set of those sample points. We next construct the Voronoi diagram $VD(S)$. Then, for each sample point $\mathbf{p}_i \in S$, we draw a circular arc centered at \mathbf{p}_i , with radius h in the region $R(S; \mathbf{p}_i) - A$. A collection of these circular arcs gives an approximation of

the curve offset from A by distance h . As the density of the sample points is raised, this approximation converges to the true offset curve.

The same idea can be applied to the construction of the offset surface of a three-dimensional region A . For this purpose, we choose a set S of sufficiently many sample points on the boundary of A , and construct the three-dimensional Voronoi diagram. For each sample point $\mathbf{p}_i \in S$, we generate a spherical surface element centered at \mathbf{p}_i with radius h in $R(S; \mathbf{p}_i) - A$. The collection of these spherical surface elements constitutes the approximation of the offset surface which is a distance h from A .

As shown in the next section, the Voronoi diagram can be generalized by replacing the generating points with other geometric elements such as line segments and surface patches. Using such generalized Voronoi diagrams, we can construct offset curves and surfaces more exactly, because we can treat the boundary shape exactly instead of choosing sample points.

18.5.4. Interpolation

Let $S = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$ be a set of n points, called **data sites**, in the plane. Suppose that at each site \mathbf{p}_i the height z_i is given. We want to construct a ‘reasonable’ surface $z(\mathbf{p})$, $\mathbf{p} \in \text{CH}(S)$, defined within the convex hull of S , such that the surface realizes the given heights at the data sites, i.e., $z(\mathbf{p}_i) = z_i$ for $i = 1, 2, \dots, n$. This is the problem of interpolation for irregularly located data sites. The function $z(\mathbf{p})$ is called an **interpolant**, and for each point $\mathbf{p} \in \text{CH}(S)$, the value of $z(\mathbf{p})$ is called the **interpolated height** at the **target point** \mathbf{p} .

The interpolation problem is not well defined, because the meaning of the ‘reasonable’ is not unique; it depends on the context. Many interpolation methods have been proposed; one class of interpolation methods is based on the Voronoi and Delaunay diagrams.

Piecewise-linear interpolation

A naive approach is piecewise-linear interpolation based on the triangulation spanning the set S of data sites. That is, we triangulate the convex hull of S with the vertices at the data sites, and the triangles are lifted into the third dimension in such a way that the vertices realize the heights given at the data sites. Thus, we get the piecewise-linear continuous surface composed of triangles.

There are many possible triangulations. Among them the Delaunay triangulation is preferable because of its equi-angularity property. Since the surface is produced as a collection of triangles, the interpolated height $z(\mathbf{p})$ at the target point \mathbf{p} is determined by the heights at the three closest sites. If this triangle is long and thin, the height $z(\mathbf{p})$ is determined by data that are far from the target point \mathbf{p} . On the other hand, if the triangle is fat and nearly equilateral, the height is determined by the data at relatively close sites.

Since the Delaunay triangulation maximizes the smallest angle, Delaunay triangles can be expected to give a good interpolant as long as we restrict ourselves to piecewise-linear interpolants.

The advantage of the Delaunay triangulation can also be understood in terms of the following property. Suppose that the height data are sampled at a finite number of sites from the surface of revolution of a hyperbola $z = x^2 + y^2$, which is convex downward.

Then, as we saw in the lift-up method, a Delaunay triangulation of the sites is the only triangulation that gives a piecewise-linear interpolant that is convex downward. Hence, the Delaunay triangulation is most suitable for the interpolation.

Laplace's interpolation

Let S be a set of data sites and \mathbf{p} be the target point in the interior of $\text{CH}(S)$. In the Voronoi diagram $\text{VD}(S \cup \{\mathbf{p}\})$, the Voronoi region of \mathbf{p} is bounded. Let $N(\mathbf{p})$ denote the set of data sites whose Voronoi regions share common boundary edges with the Voronoi region of \mathbf{p} . For $\mathbf{p}_i \in N(\mathbf{p})$, let $l(\mathbf{p}, \mathbf{p}_i)$ denote the length of the Voronoi edge shared by the Voronoi regions of \mathbf{p} and \mathbf{p}_i . We define

$$s^0(\mathbf{p}, \mathbf{p}_i) = \frac{l(\mathbf{p}, \mathbf{p}_i)}{\|\mathbf{p} - \mathbf{p}_i\|}, \quad \mathbf{p}_i \in N(\mathbf{p}), \quad (18.10)$$

and normalize it by

$$\bar{s}^0(\mathbf{p}, \mathbf{p}_i) = \frac{s^0(\mathbf{p}, \mathbf{p}_i)}{\sum_{\mathbf{p}_j \in N(\mathbf{p})} s^0(\mathbf{p}, \mathbf{p}_j)}, \quad \mathbf{p}_i \in N(\mathbf{p}), \quad (18.11)$$

so that

$$\sum_{\mathbf{p}_i \in N(\mathbf{p})} \bar{s}^0(\mathbf{p}, \mathbf{p}_i) = 1 \quad (18.12)$$

holds. Then we can prove the equality [16]

$$\mathbf{p} = \sum_{\mathbf{p}_i \in N(\mathbf{p})} \bar{s}^0(\mathbf{p}, \mathbf{p}_i) \mathbf{p}_i. \quad (18.13)$$

This equality implies that \mathbf{p} is represented by the affine combination of the neighboring generating points with coefficients $\bar{s}^0(\mathbf{p}, \mathbf{p}_i)$.

Hence, it is natural to consider the interpolant defined by

$$z(\mathbf{p}) = \sum_{\mathbf{p}_i \in N(\mathbf{p})} z_i \bar{s}^0(\mathbf{p}, \mathbf{p}_i). \quad (18.14)$$

Actually this definition gives a C^0 interpolant, which is called Laplace's interpolant [16]. The coefficients $\bar{s}^0(\mathbf{p}, \mathbf{p}_i)$'s are called the Laplace coordinates of \mathbf{p} .

Sibson's interpolation

As before, let S be the set of data sites and \mathbf{p} be the target point. As shown by broken lines in Figure 18.6, the Voronoi region $R(S \cup \{\mathbf{p}\}; \mathbf{p})$ of \mathbf{p} is partitioned into subregions according to the nearest data site in S . Let $s^1(\mathbf{p}, \mathbf{p}_i)$ be the area of the subregion of $R(S \cup \{\mathbf{p}\}; \mathbf{p})$ that is nearer to \mathbf{p}_i than to any other site in S , and let us normalize it by

$$\bar{s}^1(\mathbf{p}, \mathbf{p}_i) = \frac{s^1(\mathbf{p}, \mathbf{p}_i)}{\sum_{\mathbf{p}_j \in N(\mathbf{p})} s^1(\mathbf{p}, \mathbf{p}_j)}, \quad \mathbf{p}_i \in N(\mathbf{p}), \quad (18.15)$$

so that

$$\sum_{\mathbf{p}_i \in N(\mathbf{p})} \bar{s}^1(\mathbf{p}, \mathbf{p}_i) = 1 \tag{18.16}$$

holds. Sibson [28] proved the equality

$$\mathbf{p} = \sum_{\mathbf{p}_i \in N(\mathbf{p})} \bar{s}^1(\mathbf{p}, \mathbf{p}_i) \mathbf{p}_i. \tag{18.17}$$

On the basis of this equality, he proposed the interpolant defined by

$$z(\mathbf{p}) = \sum_{\mathbf{p}_i \in N(\mathbf{p})} z_i \bar{s}^1(\mathbf{p}, \mathbf{p}_i), \tag{18.18}$$

which is C^1 except at the data site [28]. This is called the Sibson's interpolant, and $\bar{s}^1(\mathbf{p}, \mathbf{p}_i)$ are called Sibson coordinates.

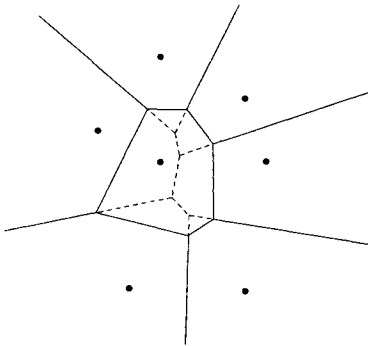


Figure 18.6. Subdivision of a Voronoi region according to the second-nearest generator.

Higher-continuity interpolation

The Laplace coordinates $\bar{s}^0(\mathbf{p}, \mathbf{p}_i)$ give a C^0 interpolant and the Sibson coordinates $\bar{s}^1(\mathbf{p}, \mathbf{p}_i)$ give a C^1 interpolant. These two kinds of coordinates have close relations; the latter can be obtained by an integration of the former, as shown below, and this relation may be generalized so that higher-continuity interpolants are obtained.

As we will see in the next section, we can generalize the Voronoi diagram by replacing the Euclidean distance with the metric $\alpha_L(\mathbf{x}, \mathbf{p}_i) = \|\mathbf{x} - \mathbf{p}_i\|^2 - r_i^2$, where r_i is the weight assigned to data site \mathbf{p}_i . Partition of the space into regions according to the nearest site rule is called the power diagram [2] or the Laguerre Voronoi diagram [17], and is denoted by $VD_L(S)$. The power diagram is similar to the ordinary Voronoi diagram in that every edge is part of a straight line.

Let us assign the same nonnegative weight r to all the data sites and the weight 0 to the target point. Let $l_L(\mathbf{p}, \mathbf{p}_i; r)$ denote the length of the edge shared by the regions of \mathbf{p} and \mathbf{p}_i in the power diagram $\text{VD}_L(S \cup \{\mathbf{p}\})$, with the convention that $l_L(\mathbf{p}, \mathbf{p}_i; r) = 0$ if the two regions do not share an edge. We define

$$s^0(\mathbf{p}, \mathbf{p}_i; r) = \frac{l_L(\mathbf{p}, \mathbf{p}_i; r)}{\|\mathbf{p} - \mathbf{p}_i\|}, \quad \mathbf{p}_i \in N(\mathbf{p}). \quad (18.19)$$

Moreover, let

$$l_N = \min_{\mathbf{p}_i \in N(\mathbf{p})} \|\mathbf{p} - \mathbf{p}_i\|. \quad (18.20)$$

Then, we can obtain the relation [16]

$$s^1(\mathbf{p}, \mathbf{p}_i) = l_N \int_0^\infty s^0(\mathbf{p}, \mathbf{p}_i; r) dr. \quad (18.21)$$

In this sense, the Sibson coordinates are obtained by the integration of the Laplace coordinates.

We can generalize this last relation. Let us define

$$s^k(\mathbf{p}, \mathbf{p}_i; r) = l_N \int_0^r s^{k-1}(\mathbf{p}, \mathbf{p}_i; \tilde{r}) d\tilde{r} \quad \text{for } k = 1, 2, \dots, \quad (18.22)$$

and

$$s^k(\mathbf{p}, \mathbf{p}_i) = s^k(\mathbf{p}, \mathbf{p}_i; r^*), \quad k = 1, 2, \dots \quad (18.23)$$

where r^* is the smallest weight r that makes the region of the target point \mathbf{p} empty in the power diagram $\text{VD}_L(S \cup \{\mathbf{p}\})$.

Then, for $k = 1$, $s^k(\mathbf{p}, \mathbf{p}_i)$ defined in this way coincides with the area $s^1(\mathbf{p}, \mathbf{p}_i)$ of the subregion defined previously. Furthermore, normalizing $s^k(\mathbf{p}, \mathbf{p}_i)$ by

$$\overline{s^k}(\mathbf{p}, \mathbf{p}_i) = \frac{s^k(\mathbf{p}, \mathbf{p}_i)}{\sum_{\mathbf{p}_j \in N(\mathbf{p})} s^k(\mathbf{p}, \mathbf{p}_j)}, \quad \mathbf{p}_i \in N(\mathbf{p}), \quad (18.24)$$

we get a similar equality:

$$\mathbf{p} = \sum_{\mathbf{p}_i \in N(\mathbf{p})} \overline{s^k}(\mathbf{p}, \mathbf{p}_i) \mathbf{p}_i. \quad (18.25)$$

Hence, we obtain interpolants

$$z(\mathbf{p}) = \sum_{\mathbf{p}_i \in N(\mathbf{p})} \overline{s^k}(\mathbf{p}, \mathbf{p}_i) z_i, \quad \text{for } k = 1, 2, \dots, \quad (18.26)$$

which can be shown to be C^k continuous except at the data sites.

18.6. EXTENSIONS

18.6.1. Voronoi diagrams for general distances

The Voronoi diagram is the partition of a space according to the nearest-site rule, using the usual definition of Euclidean distance. If we replace Euclidean distance with other distances, we can define a variety of different diagrams, which are called generalized Voronoi diagrams. Here we present a framework for this type of generalization [19] and, in succeeding subsections, we will give individual generalizations.

Let $S = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$ be the set of generating points and let \mathbf{p} be an arbitrary point. We denote by $h(\mathbf{p}, \mathbf{p}_i)$ a real number called a **generalized distance** (or ‘distance’ for short) from \mathbf{p} to \mathbf{p}_i . Here, we use the term ‘distance’ in a broad sense; it need not satisfy the distance axiom or even be nonnegative. We understand that $h(\mathbf{p}, \mathbf{p}_i)$ implies the closeness of \mathbf{p} to \mathbf{p}_i ; the smaller is the value, the closer is \mathbf{p} to \mathbf{p}_i .

We define

$$R_h(S; \mathbf{p}_i) = \bigcap_{\substack{\mathbf{p}_j \in S \\ j \neq i}} \{\mathbf{p} \in \mathbf{R}^d \mid h(\mathbf{p}, \mathbf{p}_i) < h(\mathbf{p}, \mathbf{p}_j)\}, \quad (18.27)$$

and call it the Voronoi region of \mathbf{p}_i with respect to the distance h . The space \mathbf{R}^d is partitioned into $R_h(S; \mathbf{p}_1), R_h(S; \mathbf{p}_2), \dots, R_h(S; \mathbf{p}_n)$ and their boundaries; this partition is called the Voronoi diagram of S with respect to the distance h .

18.6.2. Additively weighted Voronoi diagram

Suppose that the real number w_i , called a **weight**, is assigned to \mathbf{p}_i , $i = 1, 2, \dots, n$. Consider the distance defined by

$$h_a(\mathbf{p}, \mathbf{p}_i) = \|\mathbf{p} - \mathbf{p}_i\| - w_i. \quad (18.28)$$

The Voronoi diagram with respect to this distance is called the **additively weighted Voronoi diagram** [1]. An example of this diagram is shown in Figure 18.7, where the weights are positive and are represented by the radii of the circles centered at the generating points. As a weight becomes larger, the associated generating point acquires a larger Voronoi region.

The distance $h_a(\mathbf{p}, \mathbf{p}_i)$ can be interpreted as the Euclidean distance from \mathbf{p} to the circle centered at \mathbf{p}_i with radius w_i . Hence, this Voronoi diagram can also be considered as the partition of the plane according to the nearest-neighbor rule applied to the given circles using Euclidean distance.

The loci that are on equal distance from two circles form a hyperbolic curve. Therefore, the Voronoi edges of the additively weighted Voronoi diagram are parts of hyperbolic curves.

18.6.3. Multiplicatively weighted Voronoi diagram

Given generating points \mathbf{p}_i , with positive weight w_i , we define a multiplicatively weighted distance by

$$h_m(\mathbf{p}, \mathbf{p}_i) = \frac{1}{w_i} \|\mathbf{p} - \mathbf{p}_i\|. \quad (18.29)$$

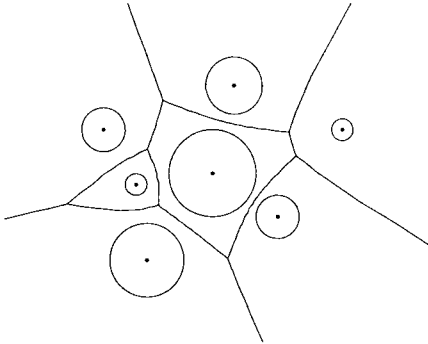


Figure 18.7. An additively weighted Voronoi diagram.

The Voronoi diagram with respect to this distance is called the **multiplicatively weighted Voronoi diagram** [4]. An example of this diagram is shown in Figure 18.8. As the weight w_i becomes larger, the associated generating point \mathbf{p}_i acquires a larger region; to attain this property, we must actually multiply $1/w_i$ instead of w_i in the definition of distance.

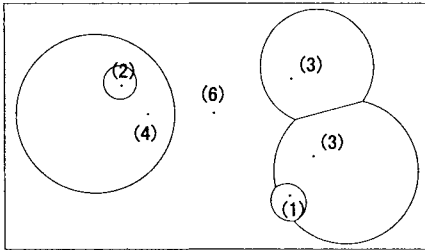


Figure 18.8. A multiplicatively weighted Voronoi diagram.

The loci at which the ratio of the Euclidean distances to two points \mathbf{p}_i and \mathbf{p}_j is constant form a circle, called Apollonius circle. Hence, the Voronoi edges of the multiplicatively weighted Voronoi diagram are circular arcs.

18.6.4. Power diagram

Given generating points \mathbf{p}_i , with weight w_i , we define a **power distance** (also called the **Laguerre distance**) by

$$d_L(\mathbf{p}, \mathbf{p}_i) = \|\mathbf{p} - \mathbf{p}_i\|^2 - w_i. \quad (18.30)$$

The Voronoi diagram with respect to this distance is called the **power diagram** [2] or the **Laguerre Voronoi diagram** [17]. An example of this diagram is shown in Figure 18.9.

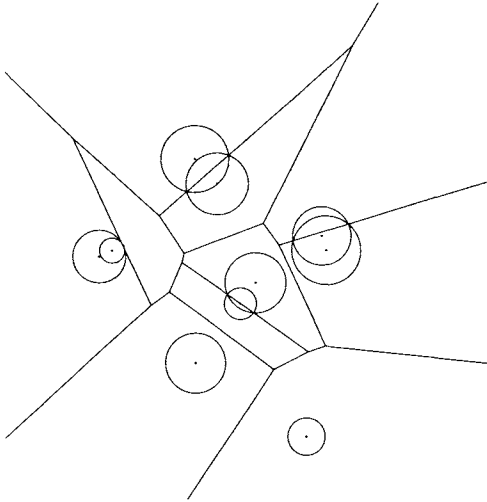


Figure 18.9. A power diagram.

A remarkable property of this diagram is that the Voronoi edges are straight lines, and they are perpendicular to the line segments connecting the associated generating points. In this sense, this diagram is similar to the ordinary Voronoi diagram.

If w_i is nonnegative, the Laguerre distance $d_L(\mathbf{p}, \mathbf{p}_i)$ can be interpreted as the square of the length of the line segment starting at \mathbf{p} and tangent to the circle centered at \mathbf{p}_i with radius $\sqrt{w_i}$. Hence, the circle centered at \mathbf{p}_i with radius $\sqrt{w_i}$ is said to be the circle associated with \mathbf{p}_i .

If the two circles associated with \mathbf{p}_i and \mathbf{p}_j have points of intersection, the Voronoi edge separating these two regions is on the line passing through the two points of intersection. This property is useful for computing the area and the boundary length of the union of circles in the following way.

Suppose that we are given n circles c_i centered at \mathbf{p}_i with radius $\sqrt{w_i}$, $i = 1, 2, \dots, n$. We construct the power diagram for \mathbf{p}_i with weight w_i , $i = 1, 2, \dots, n$. Let A_i be the intersection of c_i and the associated Voronoi region. Then, the union of all the circles is the disjoint union of all the intersections A_i , $i = 1, 2, \dots, n$. Hence, the area of the union is the sum of the areas of A_i , and the length of the boundary of the union is the sum of the lengths of the circular arcs on the boundaries of the intersections A_i .

A similar property holds for the power diagram in \mathbf{R}^3 . Hence, the three-dimensional power diagram is useful to compute the volume and the boundary area of a union of

spheres.

18.6.5. Voronoi diagram based on L_p distance

Let $\mathbf{p} = (x_1, x_2, \dots, x_d)$ and $\mathbf{q} = (y_1, y_2, \dots, y_d)$ be two points in \mathbf{R}^d . The L_p distance between \mathbf{p} and \mathbf{q} is defined by

$$h_p(\mathbf{p}, \mathbf{q}) = \{|x_1 - y_1|^p + |x_2 - y_2|^p + \dots + |x_d - y_d|^p\}^{1/p}. \quad (18.31)$$

L_2 is the Euclidean distance. L_1 corresponds to the length of the shortest path from \mathbf{p} to \mathbf{q} along horizontal and vertical streets just like the roads in Manhattan area in New York; this distance is also called the Manhattan distance.

We can define L_∞ distance by

$$h_\infty(\mathbf{p}, \mathbf{q}) = \max\{|x_1 - y_1|, |x_2 - y_2|, \dots, |x_d - y_d|\}. \quad (18.32)$$

L_∞ can be considered the limit of L_p distance with $p \rightarrow \infty$. An example of L_∞ distance is the time required for the quill of an NC machine to move from \mathbf{p} to \mathbf{q} using two independent motors, one for the horizontal direction and the other for the vertical direction.

The Voronoi diagram with respect to this distance is called the Voronoi diagram with an L_p metric [20]. An example of this diagram for $p = \infty$ is shown in Figure 18.10(a). In this diagram, the Voronoi edges usually consist of horizontal lines, vertical lines and lines running in the $\pi/4$ or $3\pi/4$ direction. However, this diagram may have unusual edges. For example, as shown by the shaded area in Figure 18.10(b), if two generating points are vertically aligned, the set of points on equal distance from the two generating points forms a region with positive area. Similar unusual edges can happen when two generating points are on a common line slanted 45 degrees in the Voronoi diagram with the L_1 metric.

18.6.6. Voronoi diagram based on elliptic distance

Given two points $\mathbf{p} = (x_1, x_2)$, $\mathbf{q} = (y_1, y_2)$ in the plane, we define

$$h_e(\mathbf{p}, \mathbf{q}) = \sqrt{a(x_1 - y_1)^2 + 2b(x_1 - y_1)(x_2 - y_2) + c(x_2 - y_2)^2}, \quad (18.33)$$

where a, b, c are reals that satisfy $ac > b^2$. Suppose that \mathbf{p} is fixed. Then, for positive constant s , the point \mathbf{q} that satisfies $d_e(\mathbf{p}, \mathbf{q}) = s$ traces an ellipse. Hence $d_e(\mathbf{p}, \mathbf{q})$ is called an elliptic distance. The Voronoi diagram with respect to this distance is called the **elliptic-distance Voronoi diagram**.

Figure 18.11(a) shows an example of the elliptic-distance Voronoi diagram, where the ellipse represents the locus of points that have the same elliptic distance from a fixed point.

The elliptic-distance Voronoi diagram is closely related to the Euclidean-distance Voronoi diagram. Indeed, we can construct the elliptic-distance Voronoi diagram by transforming the plane using an affine transformation in such a way that the associated ellipse is transformed to a circle, as shown in Figure 18.11(b); next we construct the Euclidean-distance Voronoi diagram; and finally we inversely transform the resulting diagram back to the original plane.

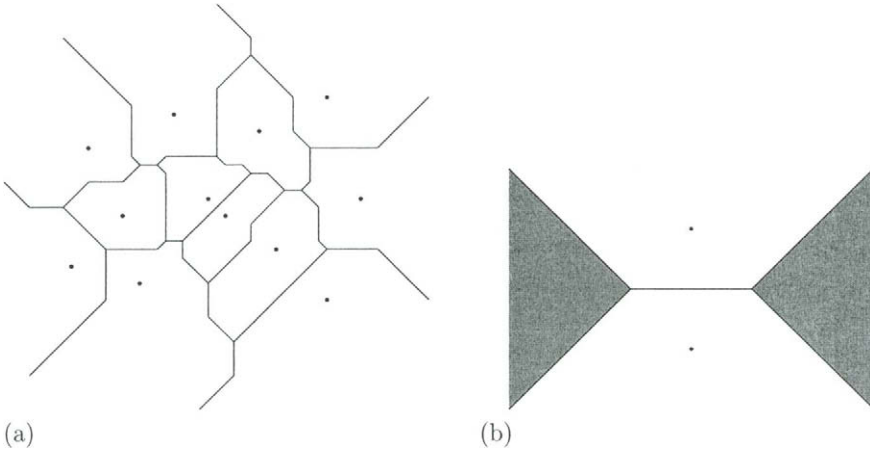


Figure 18.10. An L_∞ metric Voronoi diagram.

18.6.7. Obstacle-avoidance Voronoi diagram

Let B_1, B_2, \dots, B_k be mutually disjoint regions in \mathbb{R}^2 , and let $B = B_1 \cup B_2 \cup \dots \cup B_k$. We call B_1, B_2, \dots, B_k obstacles. Let $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n$ be generating points that are not included in any obstacles. For any point $\mathbf{p} \in \mathbb{R}^2 - B$, we define $d_o(\mathbf{p}, \mathbf{p}_i)$ as the length of the shortest path from \mathbf{p}_i to \mathbf{p} that does not pass through the interior of the obstacles.

The Voronoi diagram with respect to this distance is called the **obstacle-avoidance Voronoi diagram**. The obstacle-avoidance distance represents realistic distance for travel in a city, if we assign obstacles to the areas where we cannot enter such as lakes, rivers and buildings. Hence the obstacle-avoidance Voronoi diagram is a practical tool in the analysis of some geographic problems.

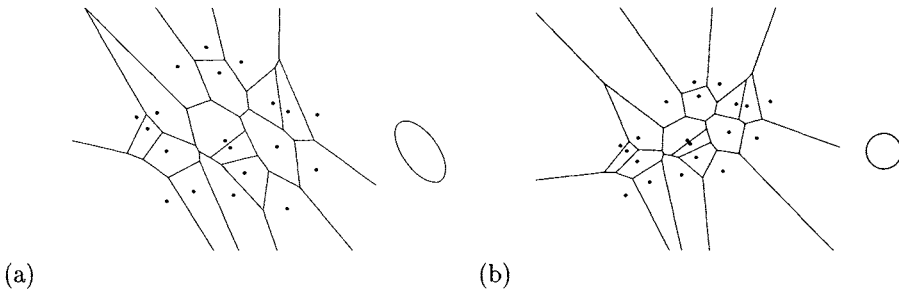


Figure 18.11. An elliptic-distance Voronoi diagram and the associated Euclidean-distance Voronoi diagram.

18.6.8. Voronoi diagram in a river

Suppose that water flows from left to right (i.e., in the positive x direction) at a constant speed u . Let $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n$ represent small islands between which ferry-boats ply. Suppose that every boat runs at a constant speed v . For any point \mathbf{p} in \mathbf{R}^2 , we define $h_r(\mathbf{p}, \mathbf{p}_i)$ as the smallest time required for the boat at \mathbf{p}_i to reach \mathbf{p} .

The Voronoi diagram with respect to this distance is called the Voronoi diagram in a river. It is known that the topological structure of the Voronoi diagram in a river is the same as the topological structure of the Euclidean-distance Voronoi diagram if $u \leq v$ [29].

18.6.9. Crystal Voronoi diagram

Suppose that crystals grow from points $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n$ at different speeds until they touch other crystals. Then, we obtain a partition of the space into crystals. This partition is called the **crystal Voronoi diagram** [25].

The crystal Voronoi diagram is similar to the multiplicatively weighted Voronoi diagram, but it is not the same. Figure 18.12 shows the crystal Voronoi diagram for two points, where one crystal grows twice as fast as the other crystal. The part of the boundary of the two crystals that is visible from both of the generating points is the same as part of the boundary of the multiplicatively weighted Voronoi diagram where the weights are the growth rates. However, the other part of the boundary is different from that of the multiplicatively weighted Voronoi diagram, because in the crystal Voronoi diagram the distance is measured as the length of the shortest path, avoiding other crystal areas.

To construct this diagram, we need to simulate the growth of the crystals; one promising approach is to use the fast marching method [26]. An example of the crystal Voronoi diagram for more than two crystals, generated by this method, is shown in Figure 18.13, where (a) and (b) show the intermediate stages of crystal growth, and (c) shows the final shapes of the crystals.

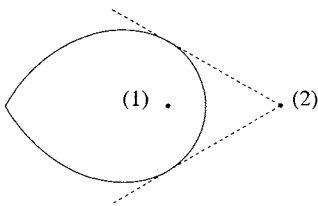


Figure 18.12. A crystal Voronoi diagram for two points.

18.6.10. Voronoi diagram for lines and polygons

Let us replace generating points with a set $S = \{l_1, l_2, \dots, l_n\}$ of n mutually disjoint line segments in \mathbf{R}^2 . For any point in \mathbf{R}^2 , we define $d(\mathbf{p}, l_i)$ as the Euclidean distance from \mathbf{p} to the nearest point in l_i . The nearest-neighbor rule with respect to this distance

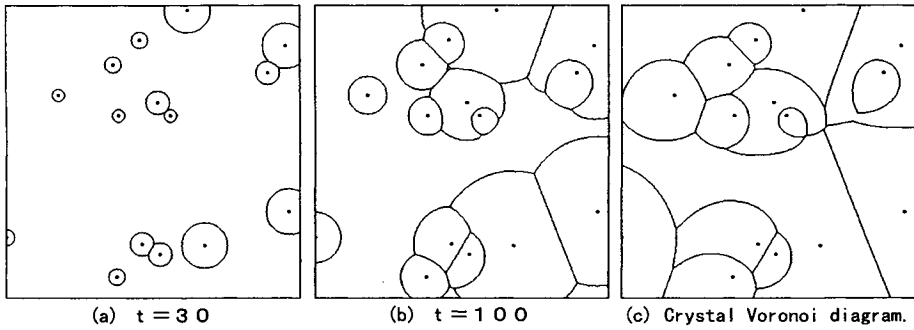


Figure 18.13. A crystal Voronoi diagram.

partitions of the plane into regions corresponding to the line segments l_1, l_2, \dots, l_n . This partition is called the Voronoi diagram for line segments [8].

The Voronoi diagram for n line segments can be constructed in $O(n \log n)$ time by the plane sweep method [33]. An example of this Voronoi diagram is shown in Figure 18.14. A similar Voronoi diagram can be defined for the set of line segments forming a polygon. Those Voronoi edges inside the polygon give the medial axis of the polygon.

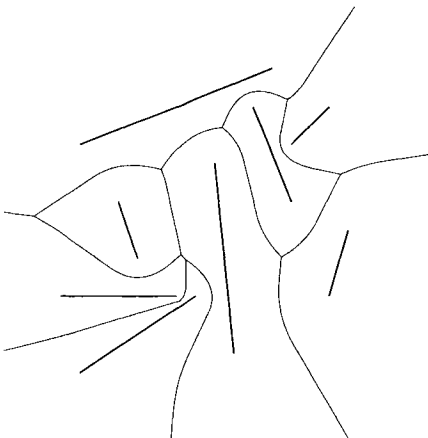


Figure 18.14. A Voronoi diagram for line segments.

18.6.11. Voronoi diagram for general figures

Line segments can be extended to general figures. Let $S = \{f_1, f_2, \dots, f_n\}$ be a set of mutually disjoint figures in \mathbf{R}^2 . We define $d(\mathbf{p}, f_i)$ as the Euclidean distance from \mathbf{p} to the nearest point in f_i . The Voronoi diagram with respect to this distance is called the Voronoi diagram for general figures. An example of this diagram is shown in Figure 18.15.

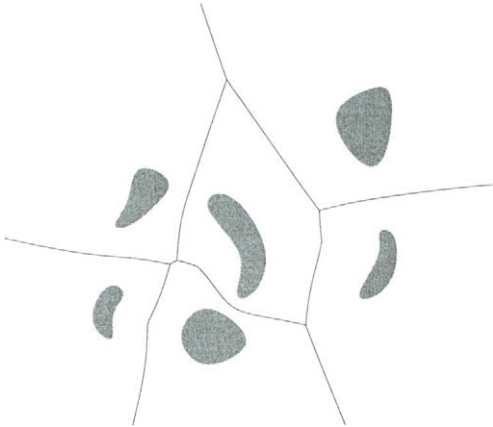


Figure 18.15. A Voronoi diagram for general figures.

If the figures in S represent empty regions in a map such as parks and school grounds. Then, the associated Voronoi diagrams might tell us to which empty regions we should go when a big earthquake takes place.

18.7. CONCLUSION

We have surveyed the basic properties and algorithms of the Voronoi diagrams, and their applications and generalizations. Here special emphasis was placed on geometric design in the sense that the section on applications treats design-related topics such as interpolation, offset and medial axes.

The concept of the Voronoi diagram is related to a wide variety of areas from the analysis of natural forms to the analysis of social and geographic systems. For the extensive survey of the Voronoi diagrams, refer to Aurenhammer [3], Fortune [13] and Okabe et al. [22].

REFERENCES

1. P. F. Ash and E. D. Bolker. Generalized Dirichlet tessellations. *Geometricae Dedicata*, 20:209–243, 1986.
2. F. Aurenhammer. Improved algorithms for dishes and balls using power diagram. *Journal of Algorithms*, 9:151–161, 1987.

3. F. Aurenhammer. Voronoi diagrams — A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23:345–405, 1991.
4. F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighted Voronoi diagram in the plane. *Pattern Recognition*, 17:251–257, 1984.
5. M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In F. K. Hwang and D.-Z. Du, editors, *Computing in Euclidean Geometry*, World Scientific, Singapore, 1992.
6. K. Q. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, 9:223–228, 1979.
7. B. Delaunay. Sur la sphère. *Bulletin of the Academy of Sciences of the U.S.S.R. Classe des Sciences Mathématiques et Naturelles*, Series 7, 6:793–800, 1934.
8. R. L. Drysdale and D.-T. Lee. Generalized Voronoi diagram in the plane. *Proceedings of the 16th Annual Allerton Conference on Communication, Control and Computing*, pages 833–842, 1978.
9. H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Berlin, 1987.
10. H. Edelsbrunner and E. P. Mücke. Simulation of simplicity—A technique to cope with degenerate cases in geometric algorithms. *Proceedings of the 4th Annual ACM Symposium on Computational Geometry*, pages 118–133, Urbana-Champaign, June 1988.
11. H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete and Computational Geometry*, 1:25–44, 1986.
12. S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.
13. S. Fortune. Voronoi diagrams and Delaunay triangulations. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, World Scientific Publishing, pages 193–233, Singapore, 1992.
14. P.-L. George. *Delaunay Triangulation and Meshing*. Hermès, Paris, 1998.
15. M. Held. *On the Computational Geometry of Pocket Machining*. Lecture Notes in Computer Science 500, Springer-Verlag, Berlin, 1991.
16. H. Hiyoshi and K. Sugihara. Voronoi-based interpolation with higher continuity. *Proceedings of the 16th Annual ACM Conference on Computational Geometry*, pages 242–250, Hong Kong, June 2000.
17. H. Imai, M. Iri and K. Murota. Voronoi diagram in the Laguerre geometry and its applications. *SIAM Journal of Computing*, 14:93–105, 1985.
18. M. Karasick, D. Lieber and R. Nackman. Efficient Delaunay triangulation using rational arithmetic. *ACM Transactions on Graphics*, 10:71–91, 1991.
19. R. Klein. *Concrete and Abstract Voronoi Diagrams*. Lecture Notes in Computer Sciences, no. 400, Springer-Verlag, Berlin, 1989.
20. D.-T. Lee. Two-dimensional Voronoi diagrams in the L_p -metric. *Journal of the ACM*, 27:604–618, 1980.
21. T. Ohya, M. Iri and K. Murota. Improvements of the incremental method for the Voronoi diagram with computational comparisons of various algorithms. *Journal of the Operations Research Society of Japan*, 27:306–336, 1984.
22. A. Okabe, B. Boots, K. Sugihara and S. N. Chiu. *Spatial Tessellations — Concepts*

- and Applications of Voronoi Diagrams*, Second Edition, John Wiley and Sons, Chichester, 2000.
23. F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20(2):87–93, 1977.
 24. F. P. Preparata and M. I. Shamos. *Computational Geometry — An Introduction*. Springer-Verlag, New York, 1985.
 25. B. F. Schaudt and R. L. Drysdale. Multiplicatively weighted crystal growth Voronoi diagram. *Proceedings of the Second Canadian Conference in Computational Geometry*, pages 214–223, North Conway, 1991.
 26. J.A. Sethian. Fast marching methods. *SIAM Review*, 41(2):199–235, 1999.
 27. R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21:243–245, 1978.
 28. R. Sibson. A vector identity for the Dirichlet tessellation. *Mathematical Proceedings of the Cambridge Philosophical Society*, 87:151–155, 1980.
 29. K. Sugihara. Voronoi diagrams in a river. *International Journal of Computational Geometry and Applications*, 2:29–48, 1992.
 30. K. Sugihara and M. Iri. Construction of the Voronoi diagram for ‘one million’ generators in single-precision arithmetic. *Proceedings of IEEE*, 80:1471–1484, 1992.
 31. K. Sugihara and M. Iri. A robust topology-oriented incremental algorithm for Voronoi diagrams. *International Journal of Computational Geometry and Applications*, 4:179–228, 1994.
 32. G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques, deuxième memoire, recherches sur les paralleloèdres primitifs. *Journal für die Reine und Angewandte Mathematik*, 134:198–287, 1908.
 33. C.-K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete and Combinatorial Geometry*, 2:365–393, 1987.
 34. C.-K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *Proceedings of the 4th Annual ACM Symposium on Computational Geometry*, pages 134–142, Urbana-Champaign, June 1988.

Chapter 19

The Medial Axis Transform

Hyeong In Choi and Chang Yong Han

The medial axis transform is a one-dimensional graph extracted from a planar shape. It has been a prime area of study, not only in computer-aided geometric design, but also in such diverse areas as computer graphics, computer vision, pattern recognition, image processing, NC tool path planning, mesh generation and font design. We review many research results concerning its basic mathematical properties and various algorithms for its accurate and efficient computation.

19.1. INTRODUCTION

The *medial axis* of a planar shape is the locus of the centers of a set of disks that maximally fit into the shape; and the *medial axis transform* is the medial axis together with the corresponding radius values. Points on the medial axis, i.e., the centers of such disks, are called the *medial axis points* of the shape, and *medial axis transform points* are similarly defined. There are many other definitions of the medial axis. Some define it as the closure of the collection of points in the shape that have the same minimum distance to the shape boundary at least two distinct boundary points. Others define the medial axis as the set of quench points of fire lines, imagining that the shape is covered by grass and is set on fire all around the boundary simultaneously. (The flame propagation speed is assumed to be constant everywhere.) In this case each radius value of the quench point corresponds to the time required for the fire to get to it. All these definitions are essentially equivalent and we choose the one that is mathematically cleanest to handle. The concept of medial axis dates as far back as to the time of Dirichlet [17] and Voronoi [48]. What they studied is now called the Dirichlet tessellation or the Voronoi diagram. It is constructed as follows: given discrete points $\mathbf{p}_1, \dots, \mathbf{p}_n$ scattered over a plane, their Voronoi diagram is the locus of points of equal distance to at least two of the given points. The Voronoi diagram partitions the plane into mutually disjoint regions V_1, \dots, V_n such that points of V_i are closer to \mathbf{p}_i than any other \mathbf{p}_k for $i \neq k$. The concept of the Voronoi diagram can mesh well with the concept of the medial axis. In particular, given discrete points $\mathbf{p}_1, \dots, \mathbf{p}_n$

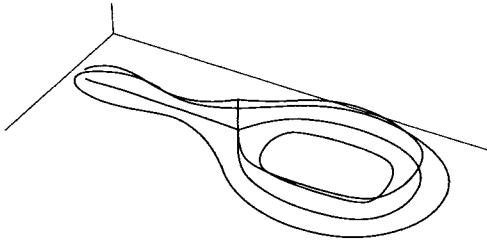


Figure 19.1. The medial axis transform as a graph in \mathbb{R}^3 .

in the plane, their Voronoi diagram is the medial axis of the region $\mathbb{R}^2 - \{\mathbf{p}_1, \dots, \mathbf{p}_n\}$.

It remains possible to define the Voronoi diagram even when a point set is replaced by some more complex geometric objects. In the case where the boundary of an object is composed of lines and circular arcs, there has been a wealth of results since Persson in the late 1970s used the Voronoi diagram to compute the tool path for NC machining. Although the medial axis and the Voronoi diagram of a shape are very similar and have many common parts, they are never the same concept, because the Voronoi diagram depends how the boundary is segmented while the medial axis doesn't. (The medial axis or the Voronoi diagram does not include the other completely either.) Refer to Sugihara's chapter 18 of this book for more information on the Voronoi diagram.

The modern incarnation of the medial axis (transform) is generally attributed to Blum [6]. In the 1960s, he felt that the geometry of the past two millennia was irrelevant to the study of amorphous shapes arising in the biological or medical sciences, since it had been developed in close relation to physical sciences. In Blum's attempted theory of shape he introduced the medial axis transform as one of the shape attributes. He wanted to use the medial axis transform—which is stable, to a certain degree, under deformation—to classify various shapes. The medial axis transform was, to him, a new kind of geometry between topology and congruent geometry (i.e., Euclidean or projective geometry). He found that he was able to approach some problems of psychology and physiology using the medial axis transform.

The medial axis transform has been widely used in many different contexts: for example, it proved to be an indispensable tool in the mesh generation for the finite-element method in numerical analysis, NC tool path generation for pocket milling, and font generation for Korean or Chinese characters, to name only a few applications. It is, however, not clear how much of Blum's original expectations were fulfilled. But, certainly, the medial axis found an important use in compactly representing the information of a shape in vision and pattern recognition, especially for bitmap images. (The image processing community calls the medial axis a skeleton.)

In this chapter, we primarily concentrate on the medial axis transform of shapes with well defined mathematical boundaries i.e. lines or smooth curves. Section 19.2 summarizes the basic mathematical facts that may serve as a solid foundation upon which future work

can be based. In Section 19.3, we give a few well established algorithms. We conclude this chapter with some remarks in Section 19.4.

19.2. MATHEMATICAL THEORY OF THE MEDIAL AXIS TRANSFORM

When the medial axis was first studied, little attention was paid to its mathematical properties, and it was more or less taken for granted that the medial axis (transform) of a reasonably nice shape is a nice one-dimensional object, usually a *finite* graph embedded in the plane. Researchers concentrate on finding various algorithms for computing the medial axis. However, providing careful proof of the ‘nice’ of the medial axis is not at all trivial. Of course, there has been some work along these lines. But the most significant results, as far as we know, are due to Hoffmann and Chiang [27] and Choi et al. [13].

In [27], Hoffmann and Chiang studied some of the mathematical properties of the medial axis transform. To get mathematically meaningful properties, they restricted the category of shapes under consideration—in particular they imposed a ‘smoothness’ requirement on the boundary curves. To this end, they defined three kinds of compact domains whose boundary is a simple closed curve \mathcal{D}_2 : with bounded curvature variation and twice differentiable, \mathcal{D}_1 : differentiable and almost twice differentiable, and \mathcal{D}_0 : almost twice differentiable. Their results can be summarized as follows: (1) the medial axis transform uniquely exists for each domain, (2) the medial axis transform is divisible, (3) the medial axis transform is connected and has a tree graph structure, and (4) the original domain can be recovered from its medial axis transform. They also mentioned the medial axis transform of multiply connected domains and non-manifold domains.

On the other hand, Choi et al. [13] obtained more definitive and comprehensive results on the medial axis transform of multiply-connected domains. To this end, they required the boundary of the domain to consist of real analytic curves, which condition guarantee the finiteness of various important geometric objects. However, this condition can be easily recast in the language of Hoffmann and Chiang, as condition \mathcal{D}_2 .

In this section, we summarize some mathematical results about the medial axis transform in a planar domain. For more details, see Choi et al. [13] in which mathematically rigorous proofs are given. We also follow the notation and the terminology in this previous paper; but for the convenience of the reader, we provide some prerequisites here.

19.2.1. Assumptions on the domain

A domain Ω is always assumed to satisfy the standing assumption [13] which we is given below. In fact, this assumption turns out to be the *optimal* one in the sense that:

- (1) The medial axis and the medial axis transform of Ω are *geometric graphs*, if Ω satisfies our assumptions. We can construct examples of domains whose medial axes and medial axis transforms are not geometric graphs if some of the assumptions are violated.
- (2) The class of the domains satisfying our assumptions includes almost all practical situations.

Standing assumption

We will assume that the domain Ω is a non-circular domain which satisfies the following two conditions.

- (1) Ω is the closure of a connected bounded open domain in \mathbb{R}^2 bounded by a finite number of mutually disjoint simple closed curves. (By ‘simple closed curve’ we mean an embedding of the unit circle in \mathbb{R}^2 .)
- (2) Each simple closed curve in $\partial\Omega$ consists of a finite number of pieces of real analytic curves.

We exclude the circular domain (i.e., the disk) since the disk poses an exception to many of our results, although everything is known about its medial axis transform. We take Ω to be closed since this simplifies many of the details. The simple closed curve bounding the unbounded region of $\mathbb{R}^2 \setminus \Omega$ is called the *outer boundary (curve)*, and the others are called the *inner boundary (curves)*. The number of inner boundary curves in $\partial\Omega$ is called the *genus* of Ω . A domain has an inner boundary curve if and only if it is not simply connected (i.e., multiply connected). This situation is typically described as ‘ Ω having a hole (holes), or homology.’

Of all the conditions in this standing assumption, the real analyticity condition is the most important, and needs some explanation. We say that a simple closed curve $\gamma : [a, b] \rightarrow \mathbb{R}^2$ ($\gamma(a) = \gamma(b)$) consists of a finite number of pieces of real analytic curves, if there are numbers $a = t_0 < \dots < t_n = b$, such that $\gamma|_{[t_{i-1}, t_i]}$ is a real analytic curve for each $i = 1, \dots, n$. In fact, one needs a slightly more restrictive condition on real analyticity in the sense that γ has to be real analytically extended to an open neighborhood of $[a, b]$. This real analyticity assumption is not so restrictive as one might think, since all curves in practical use are rational, and hence real analytic. One thing to be careful about is that the curvature does not go to infinity at the end point. Provided that this condition is met, the curve can be real analytically extended to an open neighborhood. This real analyticity condition is needed in order to make sure that there are only finitely many important objects like bifurcation points. This condition can be phrased in terms of curvature fluctuation, as Hoffmann and Chiang have done [27]; but this kind of curvature non-fluctuation condition is automatically met for the curves that are real analytic in our sense. Furthermore, as mentioned above, we can easily construct some pathological examples without the real analyticity condition. The following examples (from [13]) illustrate this fact.

Example 1 Let Ω be the domain whose boundary consists of two C^∞ curves α and β , where α is an arc portion of the unit circle $\{\zeta \in \mathbb{C} : |\zeta| = 1\}$ and β is a curve represented by

$$\beta(\theta) = (1 + e^{-1/\theta^2} \sin^2 \frac{1}{\theta})e^{i\theta},$$

for sufficiently small $|\theta|$, and α and β are joined in such a way to form a closed C^∞ curve, as shown in Figure 19.2.

Then it is easy to see that the point \mathbf{p} is in the medial axis of Ω and that there are infinitely many segments in the medial axis of Ω emanating from \mathbf{p} . (Such point \mathbf{p} is an ∞ -prong point in the language of Definition 3 below.)

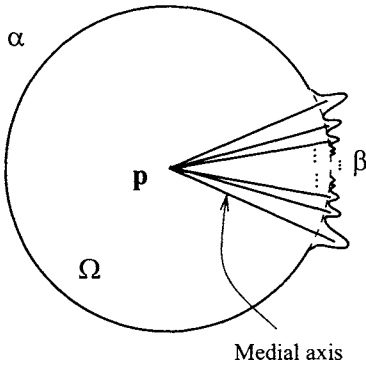


Figure 19.2. A medial axis having an ∞ -prong points.

Example 2 Let Ω be the domain whose boundary consists of two C^∞ curves γ and δ , where γ is a portion of the boundary of a stadium like shape and δ is a curve represented by

$$\delta(t) = (t, e^{-1/t^2} \sin^2 \frac{1}{t}),$$

for sufficiently small $|t|$, and γ and δ are joined in such a way to form a closed C^∞ curve, as shown in Figure 19.3.

Then it is easy to see that the medial axis of Ω has infinitely many bifurcation points.

19.2.2. Medial axis transform

Now we define the medial axis and the medial axis transform.

Let $B_r(\mathbf{p})$ denote the closed disk of radius r centered at \mathbf{p} . We define the set $\mathcal{D}(\Omega)$ by $\mathcal{D}(\Omega) = \{B_r(\mathbf{p}) \mid B_r(\mathbf{p}) \subset \Omega\}$.

That is, $\mathcal{D}(\Omega)$ is the set of all disks contained in Ω .

The core of a domain Ω is the set of all maximal disks in Ω , that is,

$$\mathbf{CORE}(\Omega) = \{B_r(\mathbf{p}) \in \mathcal{D}(\Omega) \mid B_s(\mathbf{q}) \in \mathcal{D}(\Omega) \text{ and } B_r(\mathbf{p}) \subset B_s(\mathbf{q}) \text{ implies } B_r(\mathbf{p}) = B_s(\mathbf{q})\}.$$

A disk $B_r(\mathbf{p})$ in $\mathbf{CORE}(\Omega)$ is called a maximal disk, and in this case $\partial B_r(\mathbf{p})$ is called a maximal circle or contact circle.

Definition 1 (Medial axis and medial axis transform)

The medial axis of a domain Ω is the set of all centers of disks in $\mathbf{CORE}(\Omega)$. That is,

$$\mathbf{MA}(\Omega) = \{\mathbf{p} \in \Omega \mid B_r(\mathbf{p}) \in \mathbf{CORE}(\Omega)\}.$$

The medial axis transform of a domain Ω is the set of all ordered pairs of centers and radii of disks in $\mathbf{CORE}(\Omega)$. That is,

$$\mathbf{MAT}(\Omega) = \{(\mathbf{p}, r) \in \Omega \times \mathbb{R} \mid B_r(\mathbf{p}) \in \mathbf{CORE}(\Omega)\}.$$

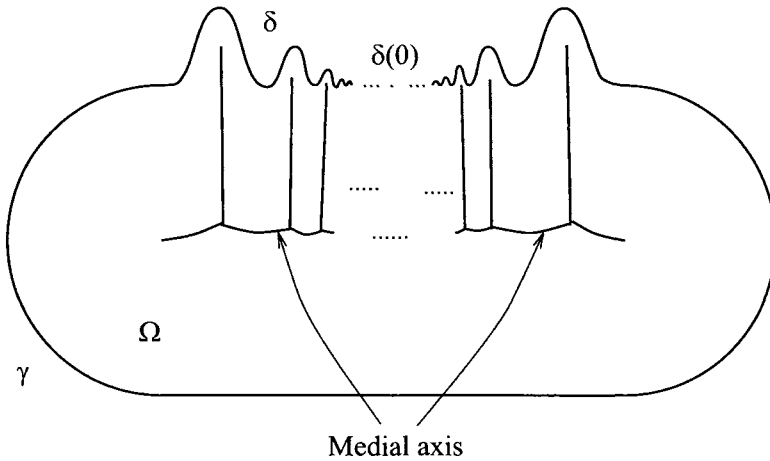


Figure 19.3. A medial axis having infinitely many bifurcation points.

Remark 1 In this case, we allow $r = 0$ and consider $B_r(\mathbf{p})$ as $\{\mathbf{p}\}$. Such cases occur exactly at the sharp corners of $\partial\Omega$.

A boundary point is a *corner (point)* if the unit tangent vector field is discontinuous at that point. It is called a *sharp corner* if the interior angle is strictly less than π , and a *dull corner* if the interior angle is strictly greater than π .

For a medial axis point \mathbf{p} , $B(\mathbf{p})$ denotes the disk $B_r(\mathbf{p})$ in $\text{CORE}(\Omega)$ with center \mathbf{p} .

Definition 2 Let $B(\mathbf{p})$ be a disk in $\text{CORE}(\Omega)$. Then we define the contact set of \mathbf{p} (or of $B(\mathbf{p})$, or of $\partial B(\mathbf{p})$), denoted by $C(\mathbf{p})$, as

$$C(\mathbf{p}) = \partial B(\mathbf{p}) \cap \partial\Omega.$$

A point in $C(\mathbf{p})$ is called a contact point of \mathbf{p} (or of $B(\mathbf{p})$, or of $\partial B(\mathbf{p})$). A connected component of $C(\mathbf{p})$ is called a contact component of \mathbf{p} (or of $B(\mathbf{p})$, or of $\partial B(\mathbf{p})$). A contact component is called an isolated contact point if it is a point, and a contact arc if it is an arc containing its two end points. Finally, $\partial B(\mathbf{p})$ is called a contact circle.

We note that a contact component is either an isolated contact point or a contact arc.

Now we can characterize the medial axis points by the number of their contact components.

Definition 3 A point \mathbf{p} in $\text{MA}(\Omega)$, which is not a sharp corner point, is called an n -prong point ($n \geq 1$), if $C(\mathbf{p})$ has n contact components. We classify sharp corner points as 1-prong points [14]. An n -prong point \mathbf{p} for $n \geq 3$ is called a bifurcation point. A 1-prong point \mathbf{p} is also called a terminal point.

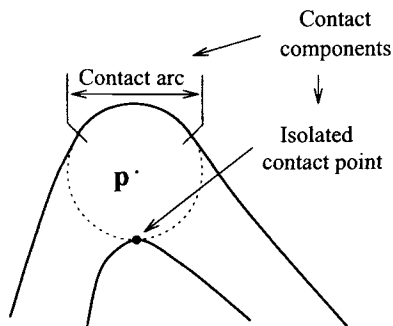


Figure 19.4. A contact arc and contact point.

Let (\mathbf{p}, r) be in $\mathbf{MAT}(\Omega)$. We call $B_r(\mathbf{p})$ a bifurcation disk if \mathbf{p} is a bifurcation point. In this case, $\partial B_r(\mathbf{p})$ is called a bifurcation circle. A disk $B(\mathbf{p}) \in \mathbf{CORE}(\Omega)$ is called an osculating disk at $\mathbf{q} \in \partial\Omega$, if $\partial B(\mathbf{p})$ is an inscribed circle which osculates $\partial\Omega$ at \mathbf{q} .

In fact, around an n -prong point \mathbf{p} ($n \geq 1$), the medial axis $\mathbf{MA}(\Omega)$ has exactly n ‘prongs’ emanating from \mathbf{p} . As we will see in Section 19.2.4, this is a consequence of the graph structure of the medial axis (transform).

It is a fact that a terminal (i.e., 1-prong) point which is not a sharp corner is the center of an inscribed osculating circle, where an *inscribed osculating circle* of Ω is a circle contained in Ω which osculates $\partial\Omega$ at some point of $\partial\Omega$. Also, it can be easily be shown geometrically that the curvature of $\partial\Omega$ takes a local maximum at an osculating point of an inscribed osculating circle. See Theorem 3.1 in [13] for a proof.

19.2.3. Finiteness results

From the real analyticity of the boundary of the domains in our class, we can derive some finiteness results about the medial axis (transform). First we require some definitions.

Definition 4 A 2-prong point \mathbf{p} in $\mathbf{MA}(\Omega)$ is a generic 2-prong (point), if the following conditions are satisfied.

- (1) The two contact components of \mathbf{p} are isolated contact points (denoted by \mathbf{q}_1 and \mathbf{q}_2).
- (2) If \mathbf{q}_i ($i = 1, 2$) is not a dull corner, then $\partial\Omega$ near \mathbf{q}_i is real analytic, and \mathbf{p} is within the focal locus of a small piece of $\partial\Omega$ near \mathbf{q}_i .
- (3) If \mathbf{q}_i ($i = 1, 2$) is a dull corner, then $\overrightarrow{\mathbf{q}_i\mathbf{p}}$ is in a purely interior direction of \mathbf{q}_i .

See [13] for the definition of ‘being within the focal locus.’ Now we state the finiteness result.

Theorem 1 Each of the following is finite in $\mathbf{MA}(\Omega)$.

- (1) The number of contact components of a point.
- (2) The number of 1-prongs.
- (3) The number of bifurcation points.
- (4) The number of 2-prongs which are not generic.

19.2.4. Graph structure of medial axis transform

It has previously [13] been shown that the medial axis (transform) is path-connected. Furthermore, the medial axis (transform) of a domain has all the topological information of the original domain in the sense that:

Theorem 2 $\mathbf{MA}(\Omega)$ is a strong deformation retract of Ω , and in particular, $\mathbf{MA}(\Omega)$ and $\mathbf{MAT}(\Omega)$ are homotopic to Ω .

The main result [13] is that $\mathbf{MA}(\Omega)$ and $\mathbf{MAT}(\Omega)$ have the structure of the *geometric graph*. We call a set in \mathbb{R}^2 (or in \mathbb{R}^3) a *geometric graph*, if it is topologically a usually connected graph with a finite number of vertices and edges, where a vertex is a point in \mathbb{R}^2 (or in \mathbb{R}^3) and an edge is a real analytic curve with a finite length, whose limits of tangents at the end points exist.

Theorem 3 (Graph structure of medial axis (transform))

$\mathbf{MA}(\Omega)$ ($\mathbf{MAT}(\Omega)$) is a finite geometric graph.

In fact, $\mathbf{MA}(\Omega)$ and $\mathbf{MAT}(\Omega)$ are isomorphic as graphs, and besides the above theorem, we can obtain the following correspondence between the vertex degree of a point in $\mathbf{MA}(\Omega)$ ($\mathbf{MAT}(\Omega)$) and the geometric property of that point. Let (\mathbf{p}, r) be a point in $\mathbf{MAT}(\Omega)$.

- (1) If \mathbf{p} is in an edge of $\mathbf{MA}(\Omega)$, then \mathbf{p} is a generic 2-prong, and thus $\mathbf{MA}(\Omega)$ is real analytic at \mathbf{p} (*resp.*, (\mathbf{p}, r)).
- (2) If \mathbf{p} is a vertex of degree 1 in $\mathbf{MA}(\Omega)$, then \mathbf{p} is either a sharp corner or the center of an inscribed osculating circle with one contact component.
- (3) If \mathbf{p} is a vertex of degree 3 or higher in $\mathbf{MA}(\Omega)$, then \mathbf{p} is a bifurcation point.
- (4) If \mathbf{p} is a vertex of degree 2 in $\mathbf{MA}(\Omega)$, then \mathbf{p} is a 2-prong which is not generic.

The same results hold for (\mathbf{p}, r) and $\mathbf{MAT}(\Omega)$.

19.2.5. Domain decomposition lemma

In this section, we introduce our fundamental tool called the domain decomposition lemma which allows us to decompose a given domain into smaller and simpler subdomains so that the medial axis transform of the original domain is preserved as the union of the medial axis transforms of the subdomains. Using this domain decomposition lemma the medial axis transform is ‘localized’ so that, no matter how complicated the original domain is, its medial axis transform can be built out of simple building blocks which are easy to handle.

This technique makes it easy to analyze the mathematical properties of the medial axis transform. Moreover, since each building block is trivial to handle, the algorithm for finding the medial axis transform essentially boils down to bookkeeping for the domain decomposition procedure, which was greatly exploited by Choi et al. [14]. See Figure 19.5 for an illustration of the basic idea of domain decomposition.

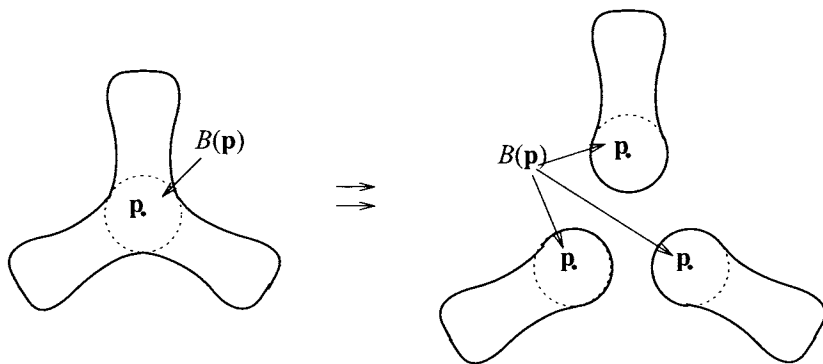


Figure 19.5. The basic idea of domain decomposition.

Theorem 4 (Domain decomposition lemma.)

For any fixed medial axis point $\mathbf{p} \in \mathbf{MA}(\Omega)$, let $B(\mathbf{p})(= B_r(\mathbf{p}))$ be the corresponding maximal disk, i.e., $B(\mathbf{p}) \in \mathbf{CORE}(\Omega)$. Suppose A_1, \dots, A_n are the connected components of $\Omega \setminus B(\mathbf{p})$. Denote $\Omega_i = A_i \cup B(\mathbf{p})$ for $i = 1, \dots, n$. Then

$$\mathbf{MA}(\Omega) = \bigcup_{i=1}^n \mathbf{MA}(\Omega_i)$$

and

$$\mathbf{MAT}(\Omega) = \bigcup_{i=1}^n \mathbf{MAT}(\Omega_i).$$

Moreover, we have

$$\mathbf{MA}(\Omega_i) \cap \mathbf{MA}(\Omega_j) = \{\mathbf{p}\}$$

and

$$\mathbf{MAT}(\Omega_i) \cap \mathbf{MAT}(\Omega_j) = \{(\mathbf{p}, r)\},$$

for every distinct i and j .

This lemma has been proved elsewhere [13]. The following definition describes the simplest building blocks out of which the entire MAT can be built.

Definition 5 (Fundamental domain)

A domain Ω is a fundamental domain, if $\mathbf{MA}(\Omega)$ has no bifurcation points.

19.3. ALGORITHMS

19.3.1. Piecewise linear and circular arc boundary

The first efficient algorithm for the computation of the medial axis was developed by Preparata [33]. However, it is restricted to convex domains with simple polygonal boundaries. Preparata's algorithm for computing the medial axis of a convex polygon G may be described in two steps.

- (Reduction step) A sequence of convex polygons $G = G_n, G_{n-1}, \dots, G_4, G_3$ are generated: each polygon G_i of the sequence is obtained from its predecessor G_{i+1} by an edge removal/vertex addition process. The criterion to determine which edge of the $(i + 1)$ -sided polygon G_{i+1} must be removed to yield the i -sided polygon G_i is given elsewhere [33]. Clearly, completion of the edge removal process always results in a triangle G_3 , whose medial axis may be trivially computed.
- (Construction step) The medial axes of the polygons G_4, \dots, G_n are incrementally 'updated' from the medial axes of their predecessors by an edge addition/vertex removal process. The same edge that was removed from the polygon G_{i+1} during the edge removal process is now included in the polygon G_i .

The algorithm to update the medial axis of G_{i+1} from the medial axis of G_i during the edge addition/vertex removal process is presented in detail elsewhere [33]. Applying it iteratively, the medial axis of the original polygon $G_n = G$ is computed. The running time of the algorithm has been shown to be $O(n \log n)$. A slightly modified algorithm, with a running time of $O(n^2)$, has also been suggested [33] for computing the medial axes of non-convex polygons.

In 1982, Lee presented [29] an $O(n \log n)$ algorithm for computing the medial axis of a planar shape represented by an n -edged simple (non-convex) polygon. Prior to this work, Kirkpatrick had presented [28] an asymptotically optimal $O(n \log n)$ algorithm for finding continuous skeletons of a set of disjoint objects. In fact Lee's algorithm in [29] performed the computation of the Voronoi diagram of a simple polygon. Since one can obtain the medial axis of a simple polygon from its Voronoi diagram by removing the edges of the Voronoi diagram that connect to the reflex vertices (which is the dull corner in our language), computing the medial axis from the Voronoi diagram does not increase the time-complexity of the algorithm.

Basically, Lee's approach was a divide and conquer. Let G be a simple polygon with vertices q_1, \dots, q_n and let e_i denote the line segment $\overline{q_i, q_{i+1}}$. For each maximally contiguous sequence of reflex vertices q_j, \dots, q_k , he defined a chain of the edges of G as a sequence of elements $e_{j-1}, q_j, e_j, q_{i+1}, \dots, q_{k-1}, e_{k-1}, q_k, e_k$. His idea is to divide G into two lists G_1 and G_2 and thus recursively to construct the Voronoi diagrams $\text{VD}(G_1)$ and $\text{VD}(G_2)$, and then to merge both Voronoi diagrams to form the Voronoi diagram of G . Since the merging process takes $O(n)$ time, the overall running time is $O(n \log n)$. In implementing this algorithm, he actually partitioned G into several chains C_1, \dots, C_h . The reason for this was that the Voronoi diagram of a chain could be computed in time proportional to the number of elements in the chain. This result has been a basis of numerous algorithms dealing with polygonal boundary.

Lee's algorithm [29] cannot compute Voronoi diagrams of multiply-connected polygonal domains. Two algorithms have been independently developed by Held [26] and by Srinivasan and Nackman [45] for this purpose. While Held's algorithm is capable of handling multiply-connected domains bounded by piecewise linear and circular segments, the algorithm developed by Srinivasan and Nackman can only accommodate polygons with a finite number of interior polygonal holes. Both of these methods are extensions of Lee's algorithm, and they employ a similar divide-and-conquer strategy, which may be summarized as follows:

- (1) Compute the interior Voronoi diagram of the outer boundary, neglecting the presence of the inner boundaries. This may be accomplished by Lee's algorithm.
- (2) Sort the inner boundaries into a decreasing order determined by the y -coordinate of the topmost vertex of each inner boundary.
- (3) For each of the inner boundaries (in sorted order):
 - (a) Compute the exterior Voronoi diagram of the inner boundary, neglecting the presence of all the outer boundaries.
 - (b) Compute the 'merge' curve between the Voronoi diagram of the inner boundary and the merged Voronoi diagram computed thus far.
 - (c) Discard edges of the 'old' Voronoi diagrams that do not belong to the 'new' Voronoi diagram.

The merge curve in Step 3b is basically the contiguous set of edges of the new Voronoi diagram which did not exist in the old one. We also note that an extension of Lee's algorithm is necessary to execute Step 3a, since the 'exterior' Voronoi diagram of the inner boundaries needs to be computed.

19.3.2. Domains with free-form boundaries

Chou [15] was the first to develop an algorithm for computing the Voronoi diagram of a curvilinear domain. In his approach terminal points of the Voronoi diagram are first located at the convex, i.e., sharp, corners and the centers of the interior osculating circles of the boundary curve. The edges of the Voronoi diagram are then traced as a sequence of bisector points for the appropriate boundary segments; the terminal points of the Voronoi diagram are chosen as the starting points of this tracing scheme. The algorithm may also be extended to compute the Voronoi diagram of the region exterior to the boundary.

Farouki et al. [18–21] and Ramamurthy et al. [35–38] also made a great contributions to the development of both theory and algorithms for computing Voronoi diagrams and medial axes of planar domains with curvilinear (polynomial or rational) boundaries. Their studies can roughly be separated in two parts: the study of curve/curve bisectors and the study of general Voronoi diagram and medial axis algorithms.

Since the edges of Voronoi diagrams and medial axes are point/curve or curve/curve bisectors, algorithms for computing these bisector forms must be invoked by any construction algorithm. While an algorithm for computing generic point/curve bisectors (for

a fixed point and a rational curve) was already available in literature, until then there existed no algorithm for computing the bisectors of pairs of rational curve segments.

Hence, the first important contribution of Farouki et al. and Ramamurthy et al. was the development of an algorithm for computing curve/curve bisectors. The important features of this algorithm are: (i) the bisectors of curved segments are directly computed without approximating the input curves; (ii) bisector segments having exact (rational) parameterizations are explicitly captured and represented in that form; (iii) bisector segments requiring approximation are guaranteed to have their geometric error (deviation from the exact curve) less than any prescribed tolerance; and (iv) tangent discontinuities of bisector loci are explicitly identified and included in their representation.

It has also been observed that certain degenerate bisector forms routinely arising in Voronoi diagram and medial axis constructions of planar domains cannot be accommodated by generic bisector algorithms. These include: (i) the bisector of a point and a curve in the case when the point lies on the curve; (ii) the bisector of a curve with itself (i.e., its self-bisector) and (iii) the bisector of two curves having a common endpoint. Owing to the complex nature of these degenerate bisector forms, substantial theoretical advances over the generic-case bisector algorithms have been necessary.

Farouki et al. and Ramamurthy et al. also developed theory and algorithms for computing Voronoi diagrams and medial axes of planar domains with curvilinear boundaries. As part of these theoretical developments, the precise relationship and differences between Voronoi diagrams and medial axes was also fully explored. To construct topologically faithful Voronoi diagrams and medial axes, methods to compute the exact coordinates of the bifurcation or branch points of Voronoi diagrams and medial axes are required. Toward this end, a thorough classification of the different types of bifurcation points that may be present in the Voronoi diagrams and medial axes of planar domains was given [19–21,35–38], and the numerical schemes required to compute the exact coordinates of the bifurcations of each type was discussed.

19.3.3. Global decomposition algorithm

Most algorithms for finding the medial axis are local, in the sense that they first attempt to find the curve/curve bisectors and then try to join them together where they meet. This is a common thread in all of the algorithms described above.

In this subsection, we now describe the algorithm developed by Choi et al. [14]. For the lack of a suitable name, let us call it the global decomposition algorithm. Its overriding philosophy is quite different from the algorithms described above. First of all, it is global in nature: the main forte of this algorithm is its use of the domain decomposition lemma to decompose the original domain, no matter how complicated it is, so that it eventually becomes a collection of simple subdomains, called fundamental domains. If the original domain is subdivided for enough, then each fundamental domain becomes very simple, meaning that its medial axis transform is free of bifurcation points, and hence it is a real analytic curve. For this kind of fundamental domain, many curve/curve bisector algorithms are suitable for accurate computation of the medial axis (transform). For free-form curves, the algorithm proposed by Farouki et al. [18–21] seems to be the best.

The crux of the global decomposition algorithm is a procedure that successively decomposes the subdomains and keeps track of its data, using an appropriate data structure and

operations on it. One important benefit of this algorithm is that computation is localized to each subdomain: whatever happens in each subdomain does not affect other subdomains. This increases the stability of the algorithm. Since the medial axis transform of a domain with a free-form boundary is not a rational object, it is impossible to express it analytically. So one must try to obtain a good *approximate* medial axis transform. This is achieved by successively inserting circles so that the insertion of each circle becomes a domain decomposition step whose basic procedure is summarized as follows:

- Killing homology effectively makes the domain simply connected for the purpose of the algorithm.
- Divide the original domain into smaller and simpler domains and organize the data in such a way that the data structure makes transparent the existence of necessary bifurcation circles, inscribed osculating circles, etc., each of which requires a special algorithm.
- Compute an *approximate* medial axis transform by employing suitable curve/curve bisector algorithms.
- Using the above approximate medial axis transform data, run the recovery procedures for the medial axis transform and the boundary for each subdomains. If the error is within tolerance, the subdomain is left alone. Otherwise, decompose further the subdomain where the error is the greatest.
- Domain decomposition and recovery procedures are guaranteed to stop at finite steps because of various finiteness theorems.

The finiteness results in Theorem 1 guarantees that this algorithm eventually terminates in finite steps in the sense that the domain is decomposed into fundamental domains each of which satisfies the error bound criterion. In the above, the killing homology procedure is a device to treat a multiply-connected domain as if it were simply-connected. More detail is given by Choi et al. [14].

We now briefly describe the data structure. The basic scheme is that each maximal circle is denoted by a vertex or a node of a graph, and the edges of the graph are formed according to the usual adjacency rule. The diagram on the left in Figure 19.6 denotes a subdomain, possibly a fundamental domain, and the diagram on the right represents part of the graph where the two nodes correspond to the two circles and the edge connecting them signifies the subdomain bounded by these two circles. Figure 19.7 represents a similar situation. The difference here is that the osculating circle (labelled '2') represents a terminal node, i.e., the vertex of degree 1 in the graph.

Figure 19.8 shows a quite different picture. In this example, the node marked 'V' does not correspond to any circle in the domain. However, by a topological argument, the domain bounded by the three circles 1, 2, and 3 must contain a bifurcation point (circle). The problem here is that, although the existence of a bifurcation point is guaranteed, it has not yet been found. Our algorithm is designed to detect the existence of such implied bifurcation points, and in Figure 19.8 it is marked V to signify that it is a *virtual* node.

Figure 19.9 represents a similar situation except that in this case the virtual node has to be a terminal node.

The overall performance of the algorithm is illustrated in Figures 19.10 to 19.17. Figure 19.10 shows a given domain, and Figure 19.11 shows the stage reached by algorithm after several circles have been inserted. Its corresponding graph is given in Figure 19.12. In Figure 19.13, several more circles are inserted, and Figure 19.14 shows the recovered domain drawn on top of the original domain. In this illustration, domain recovery is simply performed by using the quadratic curves obtained from the contact data of the circles. Figure 19.15 shows a case where yet more circles have been added, and Figure 19.16 shows the recovered domain drawn on top of the original domain; in this case the domains look identical. Figure 19.17 shows the medial axis computed this way.

The global decomposition algorithm has been used in practical applications for many years, and we have found it to perform very robustly.

19.4. CONCLUDING REMARKS

We conclude this chapter by commenting on some important developments not mentioned above.

One important area not very well understood is the medial axis transform of a general three-dimensional domain. It is in general very difficult to understand and is perhaps prohibitively expensive to describe so that no one was able to come up with a complete and mathematically rigorous treatment. But there are some progress: Sherbrooke et al. [42,43] developed an algorithm for general three-dimensional polyhedral solids of arbitrary genus without cavities, with non-convex vertices and edges. Their algorithm is based on a classification scheme which relates different pieces of the medial axis to one another even in the presence of degenerate medial axis points. Wolter [51] showed that the ‘cut locus’ concept offers a common framework lucidly unifying, regardless of dimension, different concepts such as Voronoi diagrams, medial axes and equidistancial point sets.

Since its inception in the 1960s in the biological sciences, the medial axis transform has been quite useful in compactly representing the information of shapes ranging from crude bitmap images to general Riemannian manifolds. A salient topic in this regards is the ‘inverse’ problem of the medial axis transform, i.e., the boundary regeneration of the shape from its medial axis transform. Vermeer [47] provided the details for the conversion of the medial axis transform of a set of two- and three-dimensional objects to a boundary representation. She demonstrated certain smoothness properties of the medial axis transform and showed the relationship between the tangent to the medial axis transform at a point and the boundary points related to that medial axis point. Amenta et al. [2,3] introduced the ‘power crust’ algorithm to construct piecewise-linear approximations to both the object surface and the medial axis transform given an input point sample from the object surface. Their algorithm first uses the sample points to approximate the medial axis transform, and then apply an inverse transform to it to produce a piecewise-linear surface approximation.

There is a volume of literature on many other aspects of the medial axis transform. For example, there are a lot of works in the application of the medial axis transform to the mesh generation in the finite element method in numerical analysis. Another example is

the so-called skeleton of the bitmap image which is essentially the same as the medial axis transform in the context of bitmap image [23].

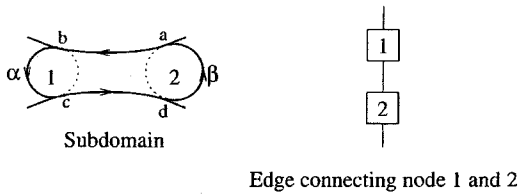


Figure 19.6. A subdomain and its data structure.

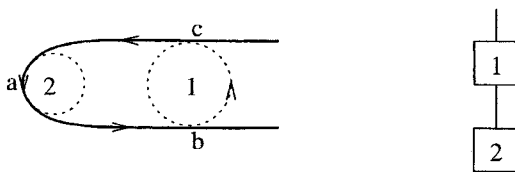


Figure 19.7. A terminal node

REFERENCES

1. H. Alt and O. Schwarzkopf. The Voronoi diagram of curved objects. In *Proc. of the 11th ACM Comput. Geom. Symp.*, pages 89–97, Vancouver, B.C., 1995.
2. N. Amenta, S. Choi, and R. Kolluri. The power crust. In *Proceedings of 6th ACM Symposium on Solid Modeling*, pages 249–260, 2001.
3. N. Amenta and R. Kolluri. The medial axis of a union of balls. In *Canadian Conference on Computational Geometry*, pages 111–114, 2000.
4. C.G. Armstrong, T.K.H. Tam, D.J. Robinson, R.M. McKeag, and M.A. Price. Automatic generation of well structured meshes using medial axis and surface subdivision. In G.A. Gabriele, editor, *Advances in Design Automation*, pages 139–149, ASME, 1991.
5. N.I. Badler and C. Dane. The medial axis of a coarse binary image using boundary smoothing. In *Proc. Pattern Recognition and Image Processing*, pages 286–291, August 1979.
6. H. Blum. A transformation for extracting new descriptors of shape. In W. Wathen-

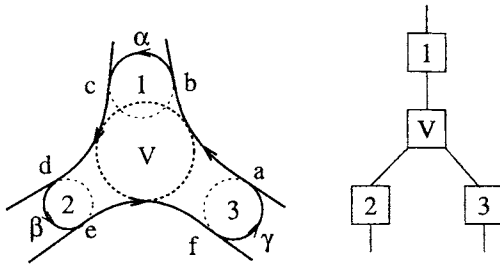


Figure 19.8. An implied bifurcation circle and its corresponding virtual node.

- Dunn, editor, *Models for the Perception of Speech and Visual Form* (Cambridge, MA, 1967), pages 362–380, MIT Press.
7. H. Blum. Biological shape and visual science (part I). *J. Theor. Biol.* 38:205–287, 1973.
 8. H. Blum and R.N. Nagel. Shape description using weighted symmetric axis features. *Pattern Recog.* ,10:167–180, 1978.
 9. J.W. Brandt. *Theory and Application of the Skeleton Representation of Continuous Shapes*. PhD thesis, Univ. of California, Davis, CA, 1991.
 10. J.W. Brandt, A.K. Jain, and V.R. Algazi. Medial axis representation and encoding of scanned documents. *J. Visual Commun. Image Represent.*, 2:151–165, June 1991.
 11. C.S. Chiang. *The Euclidean Distance Transform*. PhD thesis, Purdue University, 1992.
 12. H.I. Choi, S.W. Choi, C.Y. Han, H.P. Moon, K.H. Roh, and N.-S. Wee. New algorithm for offset of plane domain via MAT. In *Differential and Topological Techniques in Geometric Modeling and Processing '98*, pages 161–185, Bookplus Press, Pohang, Korea, 1998.
 13. H.I. Choi, S.W. Choi, and H.P. Moon. Mathematical theory of medial axis transform. *Pacific J. Math.* , 181:57–88, 1997.
 14. H.I. Choi, S.W. Choi, H.P. Moon, and N.-S. Wee. New algorithm for medial axis transform of plane domain. *Graph. Models and Image Proc.*, 59:463–483, 1997.

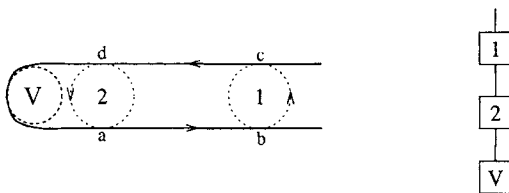


Figure 19.9. A terminal virtual node.

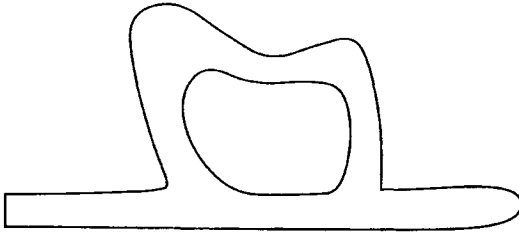


Figure 19.10. A domain.

15. J.J. Chou. *Numerical Control Milling Machine Toolpath Generation for Regions Bounded by Free Form Curves*. PhD thesis, University of Utah, 1989.
16. J.J. Chou. Voronoi diagrams for planar shapes. *IEEE Comput. Graph. Applic.* 15(3):52–59, 1995.
17. G.L. Dirichlet. Über die reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. *J. of Reine Angew, Math.*, 40:209–227, 1850.
18. R.T. Farouki and J.K. Johnstone. The bisector of a point and a plane parametric curve. *Computer Aided Geometric Design*, 11:117–151, 1994.
19. R.T. Farouki and J.K. Johnstone. Computing point/curve and curve/curve bisectors. In R.B. Fisher, editor, *Design and Application of Curves and Surfaces (The Mathematics of Surfaces V)*, pages 327–354. Oxford University Press, 1994.
20. R.T. Farouki and R. Ramamurthy. Degenerate point/curve and curve/curve bisectors arising in medial axis computations for planar domains with curved boundaries. *Computer Aided Geometric Design*, 15:615–635, 1998.
21. R.T. Farouki and R. Ramamurthy. Specified-precision computation of curve/curve bisectors. *Int. J. Comput. Geom. Applic.*, 8:599–617, 1998.
22. S.M. Gelston and D. Dutta. Boundary surface recovery from skeleton curves and surfaces. *Computer Aided Geometric Design*, 12:27–51, 1995.
23. R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Addison-Wesley, 1992.
24. H.N. Gursoy. *Shape Interrogation by Medial Axis Transform for Automated Analysis*.

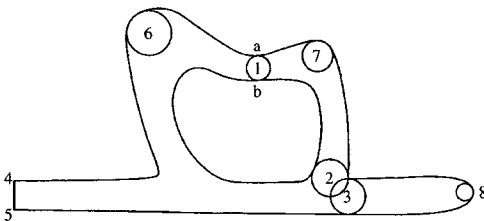


Figure 19.11. The domain of Figure 19.10 with a few circles inserted.

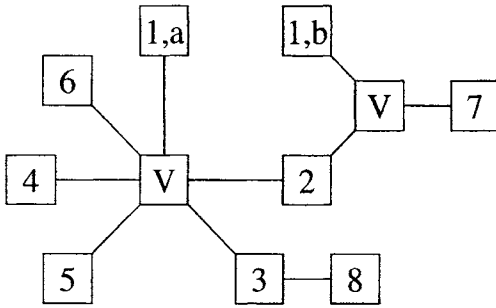


Figure 19.12. The data structure corresponding to Figure 19.11.

- PhD thesis, MIT, 1989.
25. H.N. Gursoy and N.M. Patrikalakis. An automatic coarse and fine surface mesh generation scheme based on the medial axis transform, I: Algorithms. *Engin. Comput.*), 8:121–137, 1992.
 26. M. Held. *On the Computational Geometry of Pocket Machining*. Springer, Berlin, 1991.
 27. C.M. Hoffmann and C.-S. Chiang. The medial axis transform for 2D regions. Preprint.
 28. D.G. Kirkpatrick. Efficient computation of continuous skeletons. In *Proc. 20th Annu. Symp. Found. Computer Sci.*, pages 18–27, October 1979.
 29. D.T. Lee. Medial axis transformation of a planar shape. *IEEE Trans. Pattern Anal. Machine Intell.* PAMI-4:363–369, 1982.
 30. D.T. Lee and R.L. Drysdale. Generalization of Voronoi diagrams in the plane. *SIAM J. of Computing*, 10:73–87, 1981.
 31. H. Persson. NC machining of arbitrarily shaped pockets. *Computer-Aided Design* , 10:169–174, 1978.
 32. S.M. Pizer, W.R. Oliver, and S.H. Bloomberg. Hierarchical shape description via

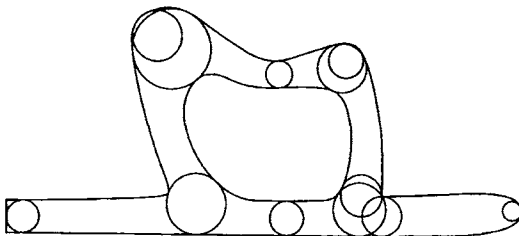


Figure 19.13. The domain of Figure 19.10 after several more circles have been inserted.

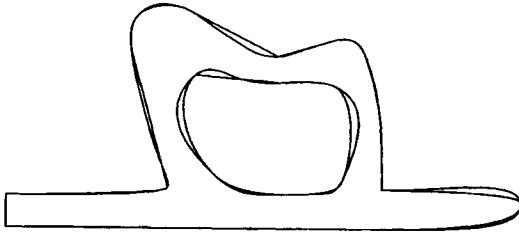


Figure 19.14. The domain of Figure 19.11 recovered using the data shown in Figure 19.13.

- the multiresolution symmetric axis transform. *IEEE Trans. Pattern Anal. Machine Intell.*, 9:505–511, 1987.
33. F.P. Preparata. The medial axis of a simple polygon. In *Proc. 6th Symp. Math. Foundations of Comput. Sci.*, pages 443–450, 1977.
 34. F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*, Second Edition. Springer, New York, 1988.
 35. R. Ramamurthy. *Voronoi Diagrams and Medial Axes of Planar Domains with Curved Boundaries*. PhD thesis, University of Michigan, 1998.
 36. R. Ramamurthy and R.T. Farouki. Topological and computational issues in Voronoi diagram and medial axis constructions for planar domains with curved boundaries. In *Differential and Topological Techniques in Geometric Modeling and Processing '98*, pages 1–26, Bookplus Press, Pohang, Korea, 1998.
 37. R. Ramamurthy and R.T. Farouki. Voronoi diagram and medial axis algorithm for planar domains with curved boundaries I: Theoretical foundations. *J. Comput. Appl. Math.*, 102(1):119–141, 1999.
 38. R. Ramamurthy and R.T. Farouki. Voronoi diagram and medial axis algorithm for planar domains with curved boundaries II: Detailed algorithm description. *J. Comput. Appl. Math.*, 102(2):253–277, 1999.
 39. D. Shaked and A.M. Bruckstein. Pruning medial axes. *Comput. Vis. Image Under-*

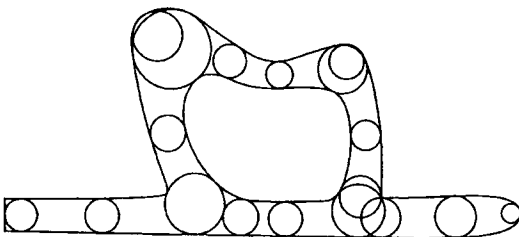


Figure 19.15. The domain of Figure 19.11 with further circles added.

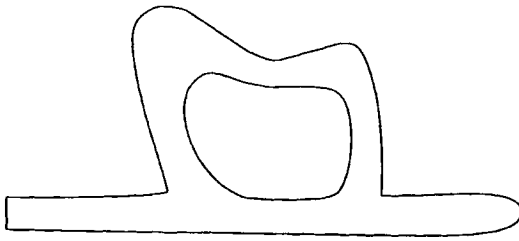


Figure 19.16. The domain of Figure 19.11 recovered using the data in Figure 19.15.

- standing*, 69:156–169, 1998.
40. D.J. Sheehy, C.G. Armstrong, and D.J. Robinson. Shape description by medial surface construction. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):62–72, 1996.
 41. E.C. Sherbrooke. *3-D Shape Interrogation by Medial Axis Transform*. PhD thesis, MIT, 1995.
 42. E.C. Sherbrooke, N.M. Patrikalakis, and E. Brisson. Computation of medial axis transform of 3-D polyhedra. In *Proceedings of the Third ACM Solid Modeling Conference*, pages 187–199, May 1995.
 43. E.C. Sherbrooke, N.M. Patrikalakis, and E. Brisson. An algorithm for the medial axis transform of 3D polyhedral solids. *IEEE Transactions on Visualization and Computer Graphics*, 2:44–61, 1996.
 44. E.C. Sherbrooke, N.M. Patrikalakis, and F.E. Wolter. Differential and topological properties of medial axis transforms. *Graph. Models and Image Proc.*, 58:574–592, 1996.
 45. V. Srinivasan and L.R. Nackman. Voronoi diagram for multiply-connected polygonal domains I: Algorithm. *IBM J. Res. Develop.*, 31:361–372, 1987.
 46. T.K.H. Tam and C.G. Armstrong. 2D finite element mesh generation by medial axis subdivision. *Adv. Engin. Software*, 13:313–324, 1991.

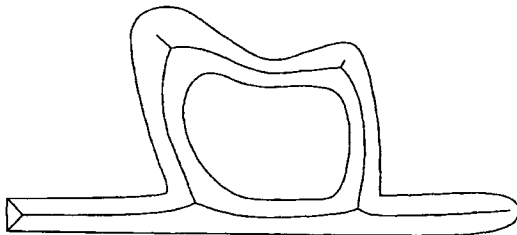


Figure 19.17. The domain of Figure 19.11 and its medial axis.

47. P.J. Vermeer. *Medial Axis Transform to Boundary Representation Conversion*. PhD thesis, Purdue University, 1994.
48. G.M. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *J. Reine Angew. Math.*, 134:198–287, 1908.
49. F.E. Wolter. Distance function and cut loci on a complete Riemannian manifold. *Arch. Math.*, 32:92–96, 1979.
50. F.E. Wolter. *Cut Locus in Bordered and Unbordered Riemannian Manifolds*. PhD thesis, Tech. Univ. of Berlin, 1985.
51. F.E. Wolter. Cut locus and medial axis in global shape interrogation and representation. MIT Ocean Engineering Design Laboratory Memorandum 92-2, January 1992.
52. C.-K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete Comp. Geom.*, 2:365–393, 1987.

Supported in part by the Research Institute of Mathematics, Seoul National University.

Chapter 20

Solid Modeling

Vadim Shapiro

Solid modeling is a consistent set of principles for mathematical and computer modeling of three-dimensional solids. The collection evolved over the last thirty years, and is now mature enough to be termed a discipline. Its major themes are theoretical foundations, geometric and topological representations, algorithms, systems, and applications.

Solid modeling is distinguished from other areas in geometric modeling and computing by its emphasis on informational completeness, physical fidelity, and universality. This article revisits the main ideas and foundations of solid modeling in engineering, summarizes recent progress and bottlenecks, and speculates on possible future directions.

20.1. INTRODUCTION

20.1.1. A premise of informational completeness

The notion of solid modeling, as practiced today,¹ was developed in the early to mid-1970's, in response to a very specific need for *informational completeness* in mechanical geometric modeling systems. This important notion has been promoted largely through the work at the University of Rochester [127] and remains central to understanding the nature, the power and the limitations of solid modeling. The concept may appear academic and redundant in the context of any one particular geometric application or algorithm, because it simply implies that the computed results should always be correct. But solid modeling was conceived as a *universal* technology for developing engineering languages and systems that must guarantee and maintain their integrity in the following precise sense.

- Any constructed representation should be *valid* in the sense that it should corre-

¹It is possible to trace the origins and techniques of solid modeling to the early beginning of computerized geometry systems in 50's and 60's [87]; there are also interesting connections to the well documented evolution of engineering drawings and descriptive geometry [12], and even to the earlier methods of synthetic geometry employed by Greeks and Egyptians more than two thousand years ago.

spond to some real physical object;

- Any constructed representation should represent *unambiguously* the corresponding physical object;
- The representation should support (at least in principle) *any and all geometric queries* that may be asked of the corresponding physical object.

Informational completeness is often expressed as the last of these requirements because in fact it is assumed to imply the first two. Implicit in the above requirements is recognition that the same physical object may be represented in more than one way, and any two such valid representations must be consistent. The difficulties arise because all requirements are stated in terms of ‘physical objects’ that cannot be used for objective tests or unambiguous comparisons. To be clear, a *human* operator may indeed be able to render an ambiguous and subjective judgement of whether a computer program performs as desired, but no computer program can check if given computer representations are informationally complete until the notion of physical object is *defined* in terms of computable mathematical properties and independently of any particular computer representation. Without such definitions, there can be no guarantees, no automation, no standards, and no solid modeling.

Similar reasoning led Requicha and Voelcker to propose a modeling paradigm in Figure 20.1 that shaped the field of solid modeling as we know it today [80,86]. The real world artifacts and the associated processes are abstracted by *postulated* mathematical models. The space of mathematical models and operations serves as the definition for the corresponding data type (class) that can be represented on a computer (in more than one way) by a representation ‘scheme’. Formally, a representation scheme can be defined as a mapping from a computer structure to a well-defined mathematical object [80]. Finally, representation schemes and accompanying algorithms are organized into systems and software applications that emulate the behavior of the real world artifacts and processes.

20.1.2. Outline

Following the modeling paradigm in Figure 20.1, it is common to survey the field of solid modeling in terms of developments related to mathematical models, representations and representation conversions, algorithms, systems, and applications. This paper adopts a similar approach. However, solid modeling is now a mature field with hundreds of relevant papers published every year in each of the above categories; many of these developments are covered in other chapters of this volume or other recent surveys. Accordingly, this chapter focuses on those aspects of solid modeling that distinguish it from other areas in geometric computing – specifically, on informational completeness, physical fidelity, and universality of representations and algorithms. As pointed out in [96], graphics, visualization, video, imaging, and many other scientific and consumer applications use and rely on solid modeling, but until now they have not driven the development of this field, perhaps because they do not appear to be critically dependent on its key characteristics. The concluding section provides a brief summary and speculates on the future of solid modeling.

Readers who are interested in a more traditional exposition to solid modeling techniques will do well by reading the landmark paper by Requicha [80], the earlier surveys of solid

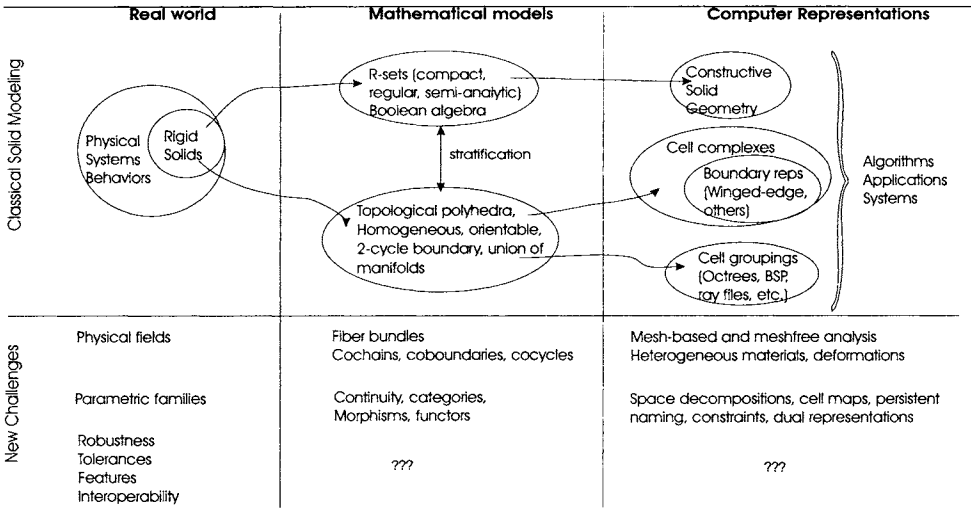


Figure 20.1. Modeling paradigm from [80] updated to reflect current understanding of concepts, mathematical modeling, and computer representations in solid modeling

modeling [28,35,73,82,87,88,93,95,96] and the Proceedings of the ACM Symposia on Solid Modeling and Applications [1]. Several monographs treat important subtopics in solid modeling [33,54] at various levels of detail, but the field has developed rapidly and no comprehensive up-to-date text on solid modeling is available as of this writing.

20.2. MATHEMATICAL MODELS

20.2.1. First postulates

Early efforts in solid modeling focused on replacing engineering drawings with geometrically unambiguous computer models capable of supporting a variety of automated engineering tasks, including geometric design (shaping) and visualization of mechanical components and assemblies, computation of their integral (mass, volume, surface) properties, simulations of mechanisms and numerically controlled machining processes, and interference detection. These developments are described in several often cited articles [87,88,127] and culminated in the Requicha's paper [80] postulating the desired properties of solid objects. All manufactured mechanical components have *finite size*; they should also have *well-behaved boundaries* that can be displayed and manipulated on a computer; the initial focus was on the *rigid* parts made of *homogeneous isotropic* material that could be *added* or *removed*. These postulated properties can be translated into properties of subsets of the three-dimensional Euclidean space E^3 .

To have a finite size, the subsets must be bounded, and rigidity is readily formulated in terms of congruence under rotations and translations.² The requirement of well-behaved

²It should be clear from the other chapters of this handbook that more general transformations in the four-dimensional projective spaces offer great computational advantages for representation and visualization

boundary is usually interpreted to mean that the set's boundary can be described by a finite collection of piecewise smooth patches, or equivalently can be finitely triangulated. In addition, the collection of the sets should be closed under several set operations: material addition and removal roughly correspond to the set union and difference operations, while interference between two such sets can be modeled by a set intersection. The class of *semi-analytic* sets satisfies all these requirements and is defined to include all those sets that can be represented by finite Boolean combinations of inequalities of the form $f_i(x, y, z) \geq 0$, where f_i is an analytic function (in the sense of admitting the Taylor series expansion about any point in space). By definition, semi-analytic sets are closed under the Boolean set operations and include the subclass of semi-algebraic sets, as well as all sets represented by polynomial and rational equalities and inequalities. Closure, projection, and connected components of a semi-analytic set are all semi-analytic, and bounded semi-analytic sets are finitely triangulable [31,52,79].

But not all bounded semi-analytic subsets of Euclidean space correspond to the intuitive notion of "solid". Semi-analytic sets may be open, closed, or neither; they may also be heterogeneous in dimension. A proper solid should be homogeneous in dimension and should contain its boundary. This notion of solidity can be characterized in more than one way mathematically. The two common approaches to defining solidity rely respectively on the point-set topology and the (combinatorial) algebraic topology. Both are important because they give rise to complementary models and computer representations: the point-set model defines the local test for solidity, while the combinatorial model specifies how solids can be built up from simple pieces (cells).

20.2.2. Continuum point set model of solidity

For any subset X of the three-dimensional Euclidean space E^3 , the points of E^3 can be classified according to their neighborhoods³ with respect to X : a full ball neighborhood indicates that the point belongs to the interior of X ; points with partial neighborhoods belong to the boundary of X . For X to be considered solid, it should contain only interior points and all those boundary points that have some interior points nearby; all other points with eroded lower-dimensional neighborhoods indicate the lack of solidity. See example in Figure 20.2.

Formally, set X is called *closed regular*⁴ if

$$X = \text{closure}(\text{interior}(X)) \quad (20.1)$$

Based on this definition, introduced and studied in [79,84,122], we can now check the neighborhoods of individual points in the set X to see if they pass the neighborhood test. If all points pass, X is indeed solid; otherwise the set is not solid, but we can *regularize* (and therefore solidify) any set X by taking the closure of its interior, as shown in Figure 20.2. Obviously, regularization can both add points to and/or remove points from the otherwise non-regular set.

of curves and surfaces in three-dimensional Euclidean space.

³In this context, the neighborhood of a point p in set X is an open ball of sufficiently small radius, centered at p , intersected with X .

⁴The dual definition of solidity using *open regular* sets was advocated in [5] to allow overlap of boundaries for assembly modeling, but it did not catch on.

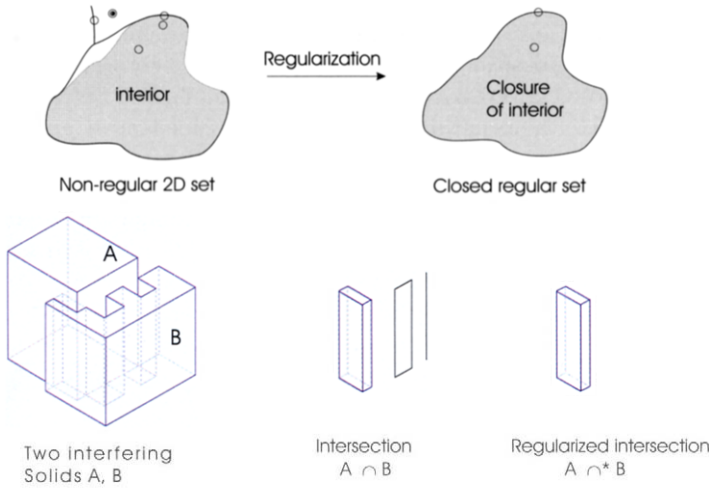


Figure 20.2. Closed regular sets capture the continuum notion of solidity in terms of neighborhoods of individual points in the set. Solids are not closed under non-regularized set operations, but closed regular sets form a Boolean algebra with regularized set operations.

But now we appear to have a problem: Figure 20.2 shows that the intersection of two closed regular sets need not be regular, and the set difference of two closed regular sets is usually not even closed. The problem is solved by requiring that the results of the usual set operations are followed by the additional step of regularization. These new regularized set operations are denoted by \cap^* , \cup^* , and $-^*$ respectively; they indeed guarantee the solidity of the results, even if the outcome may sometimes be counter-intuitive. For example, the regularized intersection of two solids with overlapping boundaries is empty by this definition, as long as their interiors do not intersect. There may seem to be little reason to expect that these new operations should possess any algebraic properties related to the usual non-regularized set operations, but they do: closed regular sets form a new Boolean algebra under the regularized set operations \cap^* , \cup^* , and $-^*$ [43,57,84].⁵

Closed regular semi-analytic and bounded sets are called *r-sets* following Requicha [79, 80]. The same formalism conveniently applies to planar shapes and surface patches (two-dimensional solids), solid lines and curve segments (solid curve), and so on – by simply changing the dimension and/or type of the reference universal set, which in turn modifies what we mean by a neighborhood or a ball.

20.2.3. Combinatorial model of solidity

Another way to characterize the set as solid is combinatorially, i.e. as composed of many solid but simple pieces (not necessarily of the same dimension), usually called *cells*. As subsets of Euclidean space, all cells are required to be orientable, with one of two orientations corresponding to an arbitrarily chosen sense of direction. Theoretically, the

⁵Note that the commonly used term “Boolean operations” is ambiguous in the larger context of geometric modeling because it is used to refer to either standard, or regularized, or both types of set operations.

particular choice of cells is not very important, because a solid is characterized by the mere existence of a non-unique decomposition into primitive solid cells.⁶ For example, all semi-analytic sets can be decomposed into very coarse disjoint manifold subsets of various dimensions as shown in Figure 20.3; the resulting submanifolds are called *strata* and the corresponding decomposition a *stratification* [131]. Alternatively, any semi-analytic set may be *triangulated*, i.e., decomposed into a collection of curved triangles (points, curve segments, triangular surface patches, and tetrahedral elements) [52]; triangles often play the role of the simplest common denominator cells in the sense that all other cells may be further triangulated.

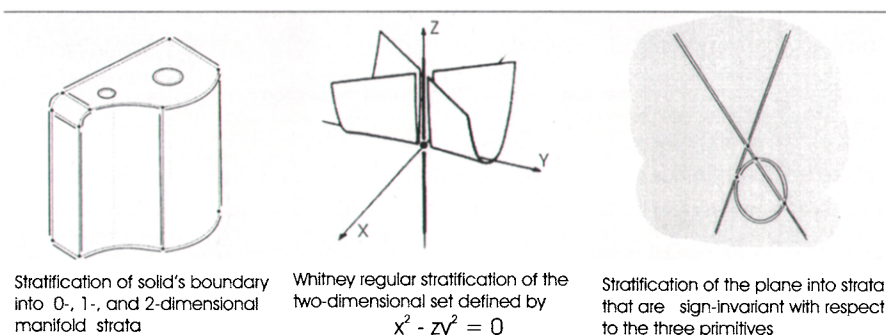


Figure 20.3. A minimal Whitney regular stratification of a set into connected manifold strata satisfies the frontier condition: the boundary of every stratum is a union of other strata.

Because a combinatorial model is defined in a cell-by-cell fashion, all geometric computations are reduced to presumably simpler computations on individual cells. The cells are usually chosen to be disjoint (for open cells) or to have disjoint interiors (for closed cells) and properly joined together into a *cell complex* so that they can also provide finite 'spatial addresses' for points in an otherwise innumerable continuum.⁷ Formally, the proper joining of cells amounts to satisfying the *frontier condition* which requires that the (relative) boundary of every cell is a finite union of other cells in the complex. Intuitively, this means that all points in any stratum are alike: their neighbourhoods are homeomorphic to each other and they all meet the same other strata. This combinatorial model of solidity is usually summarized by saying that, in addition to being semi-analytic bounded subsets of Euclidean space, solids are homogeneously n -dimensional topological polyhedra [3,4,79,80].

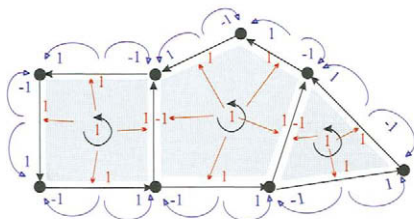
⁶One should not confuse this theoretical issue with practical representational and algorithmic consideration in *representing* solids on a computer using cellular structures which we consider in section 20.3.2.

⁷The notions of open, closed, closure, interior, etc. are all relative to the larger set; in our context, usually a curve, a surface, a volume, or a k -manifold.

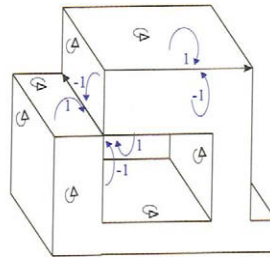
Arguably the most important property of a topological polyhedron is its combinatorial *boundary* which is itself a lower dimensional polyhedron and can be obtained by a pure algebraic computation using the concept of *chains*. Given a cell complex K , a p -dimensional *chain*, or simply p -chain, is a formal (i.e., cell-by-cell) sum

$$a_1\sigma_1 + a_2\sigma_2 + \dots + a_n\sigma_n, \tag{20.2}$$

where σ_i are p -dimensional cells of K and a_i are integer coefficients. Two p -chains on a cell complex can be added together by collecting and adding the coefficients on the same cells. The collection of all p -chains on K form a group, and using chains we can replace incidence, adjacency, and orientation computations with a simple algebra. In particular, Figure 20.4 illustrates how the oriented boundary ∂ operation can be defined algebraically in terms of elementary chains using only three coefficients from the set $\{-1, 0, +1\}$ [3,32,62].



Boundary operation on k -chain transfers the coefficients from every k -cell to all incident $(k-1)$ -cells with \pm sign depending on relative orientation. Addition cancels the interior k -cells, and yields the $(k-1)$ -boundary. Repeating the operation produces a $(k-2)$ -chain with all 0-coefficients.



Boundary of a three-dimensional solid must be a 2-chain whose boundary is 0, i.e. a 2-cycle. It does not have to be 2-manifold, but every 1-cell (edge) is shared by an even number of 2-cells (faces).

Figure 20.4. Combinatorial model of solidity allows algebraic definition of topological properties in terms of k -dimensional chains with coefficients 0,1,-1.

Starting with a 3-dimensional solid represented by a 3-chain, the boundary operation ∂ produces an oriented 2-chain (and the corresponding 2-dimensional cell complex) that defines the 2-dimensional boundary of the original solid. If we apply the boundary operation again to the 2-chain, we obtain a 1-chain with all zero coefficients. In other words, the fundamental property of ∂ is that

$$\partial(\partial(X)) = 0 \tag{20.3}$$

and guarantees that the boundary set is one or more “closed surfaces” sometimes also called “shells.” Formally, such a set whose boundary is a zero chain is called a 2-cycle [32, 80].

For historical reasons [8,16], a more restrictive model of solidity has often been adapted where topological polyhedra are restricted to the orientable three-dimensional *manifolds*

with boundary.⁸ With this restriction, the combinatorial boundary, defined as above, is not only a 2-cycle, but also a 2-manifold. This manifold model of solidity claims two theoretical advantages: (1) it rules out non-manifold sets that are sometimes deemed non-physical, and (2) the connected manifold boundaries are completely classified in terms of their Euler characteristics (see section 20.4.1). As we explain below, manifold models are also easier to represent on a computer. But unfortunately, this restriction to manifold solids violates one of the important postulated properties – that solids be closed under the regularized set union and intersection. (The union of two manifold sets touching at a point is a non-manifold set.) On the other hand, it is important to recognize that every 2-cycle is indeed a union of 2-manifolds.

20.2.4. Generalizations

More general computer representations of mechanical artifacts often include unbounded and trimmed curves and surfaces (used for aesthetic, reference, manufacturing, and other purposes), which are combined with traditional solid models (to represent cracks, or material heterogeneity, to simulate surface forming, and so on). This in turn means that the paradigm in Figure 20.1 needs to be expanded to include more general mathematical models, both continuum and combinatorial. The generalizations are relatively straightforward in principle, because they essentially amount to relaxing some of the original constraints. General continuum models correspond to closed semi-analytic subsets of E^3 , while the general combinatorial model is readily seen as an arbitrary collection of cells from some dimensionally heterogeneous cell complex [64,129].

Thus there appear to be two competing mathematical theories of solid modeling: point-set continuum and algebraic topological combinatorial. Fortunately, the two theories are entirely consistent, and we can use the two mathematical models interchangeably [79], relying on either continuum or combinatorial properties whenever needed. The key fact: every closed semi-analytic set is a topological polyhedron that can be represented by some finite cell complex. The class of closed regular subsets of E^d coincides precisely with that of homogeneously d -dimensional polyhedra, and furthermore, it can be shown that every 2-cycle in E^3 bounds some unique solid. We know immediately that every solid may be represented unambiguously by its boundary, and that the boundary has a combinatorial structure of a 2-d homogeneous polyhedron whose points all have homogeneously 2-dimensional neighborhoods.

20.3. COMPUTER REPRESENTATIONS

There are several ways to group various computer *representation schemes*, including Requicha's widely accepted description of six 'pure' representation schemes [80], but as more new and hybrid representation schemes are being proposed, their complete classification appears impractical. This survey takes a slightly different view that there are really only two fundamental ways to represent a point set, closely related to the two mathematical models of solidity identified above.

⁸In a k -manifold with boundary, every point has a neighborhood that is homeomorphic to either a k -dimensional open ball (if the point is an interior point), or a k -dimensional half-ball (if the point lies on the boundary).

- *Implicit* (and constructive) representations give rules for *testing* which points belong to the set and which are not; such representations are naturally supported by the point set continuum model of solidity; and
- *Enumerative* (and combinatorial) representations specify the rules for *generating* points in the set (and no other points); such representations are closer in spirit to the combinatorial view of solidity.

This classification of representation methods is probably the coarsest possible, but recall that any informationally complete representation should in principle support any and all geometric queries. Thus, one (even if inefficient) way to test if a point belongs to the set or not is to see whether the point is among the points being generated by some enumerative representation; and a perfectly valid way to generate points in the set is to test all candidate points against some implicit representation to determine whether they belong to the set or not. Such views are not as extreme as they may appear at a first glance, and in fact all representations are ‘misused’ in similar fashion to support applications for which they were not originally designed. In our higher-level view of representation schemes, the representation rules (implicit or enumerative) are essentially computable implementations of semi-analytic functions.⁹

20.3.1. Implicit and constructive

A very general method for defining a set of points X is to specify a predicate A that can be evaluated on any point p of space:

$$X = \{p \mid A(p) = \text{true}\} \quad (20.4)$$

In other words, X is defined *implicitly* to consist of all those points that satisfy the condition specified by the predicate A . The simplest form of the predicate is the condition on the sign of some real-value function $f(p)$, resulting in the familiar representations of sets by equalities and inequalities [65,91]. For example, if $f = ax + by + cz + d$, conditions $f(p) = 0$, $f(p) \geq 0$, and $f(p) < 0$ represent respectively a plane, a closed linear halfspace, and an open linear halfspace respectively.

More complex functions can be employed to define progressively more complex geometric shapes, giving rise to the whole discipline of implicit modeling [11]. A natural approach for semi-analytic sets is to define more complex predicates *constructively* using logical combinations of simple “primitives”, which is equivalent to using the standard set operations (\cap , \cup , $-$) on the sets defined by primitive predicates. (See examples in Figure 20.5.) Furthermore, the theory of R -functions [103,99] (see also the Chapter on Finite Element Approximation with Splines by Klaus Hollig) allows conversion of such representations into a single function inequality for any closed semi-analytic set.

Representation by point classification

Given an arbitrary point, there are at least two distinct methods for deciding its membership in the represented set. One could replace the constructive representation with

⁹In this sense, the two representation methods correspond to the two common methods for describing functions: implicitly and parametrically, the latter being essentially the continuous form of enumeration.

a single inequality predicate whose truth is determined by testing the sign of some real-valued function at the given point. This approach leads to increasingly complex arithmetic computations for what is essentially a logical computation, and therefore appears to be a poor computational strategy except when such functions are constructed directly. A more attractive alternative is to represent the constructive geometric representation on a computer using the usual tree data structure, with the primitive predicates (defining the primitive halfspaces) stored in the leaves of tree and the logical (set) operations are stored at the interior nodes. Then, the algorithm for point membership query on this data structure can be implemented naturally by proceeding recursively down the tree and “inverting” every set construction (op) in terms of the corresponding logical operation (\odot):

$$p \in (X op Y) \implies (p \in X) \odot (p \in Y) \tag{20.5}$$

Such algorithms are appealing because they require only arithmetic computations (at the leaves of the tree) and logical operations (at the internal nodes); the corresponding Boolean structure facilitates various speed-up techniques for pruning, localizing, restructuring, and parallelizing the computations [97,120].

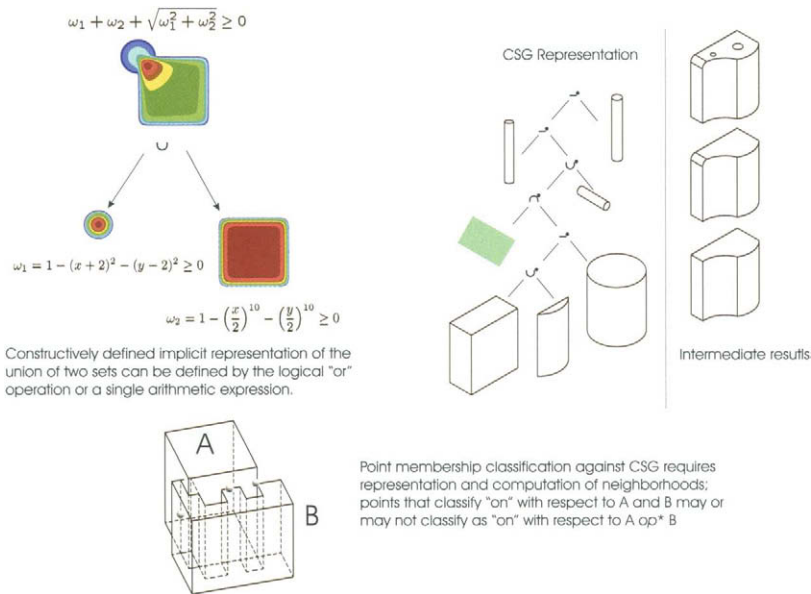


Figure 20.5. Constructive representations and Constructive Solid Geometry.

Constructive solid geometry

Using such constructive representations for solid modeling requires several extensions, as dictated by the mathematical properties postulated for solids in section 20.2. Rigid

body motions of the constructively defined sets are represented by the usual method of coordinate transformations that can also be stored as an intermediate node in the constructive tree; point membership test against a set transformed by motion M is also inverted in a straightforward fashion:

$$p \in M[X] \implies M^{-1}[p] \in X \quad (20.6)$$

Recall that semi-analytic sets are closed under the standard set operations but do not guarantee solidity; to implement the algebra of closed regular sets, the constructive representations are modified in two ways: (1) every primitive set is required to be closed regular; and (2) only regularized set operations are allowed. The latter may appear to be a relatively minor matter of replacing every set operation (op) by the corresponding regularized op^* . But in fact, this dramatically changes the computational properties of the constructive representations: point membership queries against the regularized constructions require substantially more work than is implied by Equation (20.5). Specifically, combining results of two classifications with respect to sets A and B requires not only the logical information, but also representing and combining the neighborhoods of p with respect to A , B , and Aop^*B [119,122] (see the example in Figure 20.5.)

The constructive representation scheme relying on closed regular primitives, rigid body motions, and the regularized set operations is called *Constructive Solid Geometry* or CSG [85]. By design, the use of CSG is limited by availability of solid primitives and by the necessity to represent and maintain the neighborhood information for points on primitives and their combinations. The latter task is particularly non-trivial for points with non-manifold neighborhoods and/or lying on high-degree tangent surfaces. Complete solutions have been worked out for solids bounded by planar and second degree surfaces, with only limited results available for more complex solids. The attractive properties of CSG include conciseness, guaranteed validity (by definition), computationally convenient Boolean algebraic properties, and natural control of the solid's shape in terms of high-level parameters defining the solid primitives and their positions and orientations. The relatively simple data structures and the elegant recursive algorithms further contributed to the popularity of CSG in academia and early commercial systems.

Other constructive representations

In principle, many other constructions may be added to the lexicon of implicit representations, notably offsetting [94], blending [133], convolutions [10], and other skeletal-based representations, Minkowski operations [25,61], and sweeping [37,115]. Such constructions have numerous applications in mechanical design, analysis, and planning tasks; they also flourished in computer graphics [113] where computational time and guarantee of correctness are often deemed less important than the visually pleasing results. But while such formal definitions are sometimes straightforward, they do not always guarantee the solidity of the result and do not always support a clear point membership query – which came to be regarded as a formal test for any unambiguous representation. The most popular of these constructions is the sweep representation shown in Figure 20.6 (considered a distinct representation scheme in [80]), defined for a given set X and continuous motion

$M(t)$ by the infinite union operation:

$$\text{sweep}(X, M) = \bigcup_{q \in M(t)} X^q \quad (20.7)$$

where X^q denotes set X positioned at the configuration q . In other words, $\text{sweep}(X, M)$ is the set of all points swept (or occupied) by X at some time during the motion. Sweeps are relatively well understood [2] and are useful for variety of representational tasks: computing space occupied by a moving object, material removed by a moving cutter, extrusion of a planar cross-section along one-dimensional path, and so on. The point membership test for sweep follows naturally from the studies of the dual infinite intersection operation and also inverts the construction: $p \in \text{sweep}(S, M)$ if and only if the inverted trajectory of the point $\text{sweep}(p, M^{-1})$ intersects the original solid S [39] (see Figure 20.6).

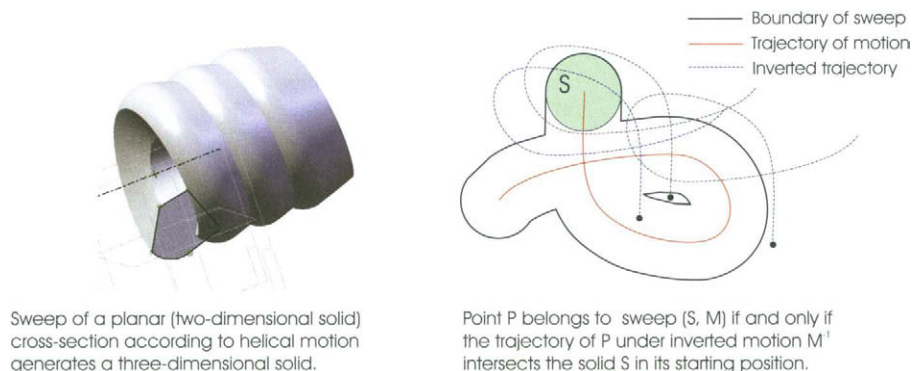


Figure 20.6. $\text{Sweep}(S, M)$ is a constructive representation for a set of points occupied by S under motion M ; a point membership procedure inverts the construction.

By definition, the constructive approach can be utilized for representing any and all semi-analytic sets, relying only on a finite set of analytic primitives and set operations. The operations of *closure* and *connected component* preserve semi-analyticity and may also be used effectively [105]. Particular representation schemes have been proposed to include extension of the classical CSG representations using topological operations in [83] and constructive representations for n -dimensional semi-analytic sets [13].

All constructive approaches are limited by their ability to compute and manage topological neighborhood information of the points in the represented sets. A related significant drawback of CSG and all other implicit representations is the lack of explicit representation and parameterization of the solid's interior and particularly its boundary.¹⁰ This leads to several practical complications, including computational difficulties in generating ordered points for the purpose of display and/or engineering analysis. Without the explicit representation, spatial locations of points in the solid or its portions are not known

¹⁰Observe that explicit representation of the boundary also implies explicit representation of the neighborhood information for all points of the solid.

a priori. In other words, the represented subsets are not *spatially addressable*, and therefore cannot be referenced persistently, for instance for attaching labels and engineering attribute information. Comparison of implicitly-represented solids is also problematic: the same solid admits infinitely many constructive representations, and even deciding if the represented set is empty may require non-trivial computations [120].

20.3.2. Enumerative and combinatorial Representation by enumeration

A seemingly more direct way to define which points belong to the solid and which do not is to enumerate the points by an explicit parametric rule. Thus, a planar curve is commonly defined by a mapping $[0, 1] \rightarrow E^2$. For every value of $t \in [0, 1]$, the points on the curve are defined by the pair of functions $x(t), y(t)$. Generating points in the defined curve segment is a matter of marching along the unit interval in small increments and evaluating the coordinate functions for every value of t . Testing if a given point belongs to the curve segment requires a more complex numerical procedure but is clearly well-defined and computable task. Similarly, one can define parametric surface patches and tri-variate solids by mappings from the unit square or cube into E^3 . Many chapters in this handbook deal with non-trivial issues related to representation of such parametric curves and surfaces for complex shapes. As the next best thing, we can represent a complex solid by enumerating not individual points but simple solid chunks or *cells*, relying on the combinatorial properties of solids as topological polyhedra. This approach leads to persistent and spatially addressable data structures that support development of cell-by-cell *traversal* algorithms and can be controlled locally and incrementally, which is particularly convenient for point generations and local modifications. The point membership query reduces to a search procedure aimed to determine which of the represented cells (if any) contains the given point. The main drawbacks of the combinatorial data structures have to do with their size and apparent lack of means to create, validate, and manipulate such structures directly.

Broadly, all combinatorial representations can be classified according to (1) the choice of the cells; and (2) restrictions on how the cells must fit together. What makes a good cell? The common requirements include dimensional homogeneity, connectedness, boundedness, and semi-analyticity. Cells may be relatively open or closed (depending on how they fit together to form a closed set); they may or may not be required to be smooth or simply-connected; but their representation must support one or both of the two fundamental computations: point testing or point generation. In other words, the cells should be simple enough to be unambiguously representable: either implicitly or enumeratively, or preferably both. Depending on the type of cells, solids may be assembled from cells in at least one of two distinct ways described below: as groupings or as cell complexes.

Groupings

*Groupings*¹¹ of solid cells of the same geometric type and dimension is probably the simplest way to represent the set. Example of groupings include: collections of three-

¹¹We prefer the term *k-grouping* (for *k*-dimensional grouping) introduced recently by [24] in favor of 'spatial enumerations,' or 'sampled' representations, because we do not wish to imply an approximation or to specify the source of the represented data.

dimensional cubes (called voxels) [111], ray-files of ray segments (finite size rectangular columns) [26], union of overlapping spherical balls, and files of two-dimensional polygons that are commonly used in computer graphics. The common principle underlying all groupings is that they are assembled from closed cells representing small chunks of space. The points in the interior of a cell are characterized by the *constant neighborhood*: all points have the same type of neighborhood with respect to E^3 , and the cells are chosen simple enough to admit both implicit and parametric representations. Because all cells are of the same geometric type and dimension, groupings support development of simple and brute-force algorithms. For example, the point membership query reduces to a point-cell test that is repeated for every cell. Depending on their particular geometric type, groupings may be organized into more compact and efficient computational structures (trees, hierarchical graphs, etc.) supporting efficient queries, processed by parallel algorithms and specialized hardware. One of the most popular representations in this category is called *octree*: a hierarchical method for representing a grouping of orthogonal three-dimensional boxes (usually cubes), where each box intersecting the boundary of the solid is subdivided into eight smaller boxes, and so on until the desired level of resolution is reached[58,18]. A grouping of convex cells defined implicitly by intersecting linear halfspaces may be more efficiently represented by a *binary space partition* (BSP) tree; each BSP node corresponds to a particular sequence of halfspaces and thus to a convex subregion of space[118].

By construction, groupings are guaranteed to be valid solids, but only some solids – those with geometry representable by the unions of cells in the grouping – can be represented exactly. Other solids can be approximated, for example, by a multi-resolution grouping, or groupings are often enhanced with additional geometric representations specifying precisely what geometry is being approximated [18,60]. Creating groupings may be expensive and operating on them may be difficult; restrictions on groupings (for example, shape of cells or orientation) may imply that the represented solids are not closed under common transformations. Even the usual operations of rigid body motions or set operations may require substantial processing and reconstruction.

Another serious drawback of groupings has to do with lack of explicit representation for the incidence or adjacency between the cells in the grouping: no conditions are usually imposed on the neighborhoods of points with respect to neighboring cells. While the corresponding topological polyhedron can be computed in principle, it may not be a grouping itself, which implies the need for additional data structures and algorithms. For example, it is not immediately clear how to define, compute, and represent the (well-defined but lower-dimensional) boundary of a solid represented by a d -grouping. Last but not least, because groupings are inherently homogeneous combinatorial structures, they are not suitable for representing mixed-dimensional point sets.

Cell complexes

The key difference between a grouping and a cell complex is that the latter requires neighborhoods for points in a cell to be constant not only with respect to E^3 , but also with respect to the other cells. In other words, a cell complex is a representation of a stratification of a solid. Cell complex representations implement the combinatorial model of solidity directly, requiring that every solid is a topological polyhedron and may therefore

be decomposed into a finite cell complex assembled from solid cells of different dimension. This effectively turns all geometric and topological queries on a given solid into simple algebraic (syntactic) computations. A number of data structures have been proposed for representing general (heterogeneous) cell complexes [30,64,129]. Further generalizations appear to be taking place, both in geometry and topology of the sets representable by solid modeling systems. A cellular representation of n -dimensional manifolds and sets has been proposed by [17,48], and a functional programming language whose semantics is defined by operations on n -dimensional polyhedral cell complexes has recently emerged [70].

Any such representation must contain enough information to determine the geometry of every cell *and* the incidence between the cells – and therein lies the main problem difficulty with all such representations: geometric and incidence information are not independent. In principle, defining geometry of every cell in the cell complex is sufficient to *also* represent the incidence information through geometric tests. But in practice, geometric tests are imprecise, searching for adjacent cells is inefficient, and specifying geometry of adjacent cells independently is redundant and wasteful since incidence constrains the corresponding geometries to ‘match’. Specifically, but without loss of generality, let us assume that every cell σ is a subset of a larger set that must include at least the closure of σ and is called the *carrier*¹² of σ ; then by the definition of cell complex: (1) every k -cell belongs to the intersection of the carriers of all incident $(k + 1)$ -cells (cofaces); and (2) the carrier of every k -cell is an interpolation of all incident $(k - 1)$ cells (faces). Therefore, the incidence relations themselves are a convenient method for defining the geometry of the incident carriers. The two conditions apply simultaneously and independently, even though only one of them suffices to define the complex mathematically. For example, in a simplicial complex, only coordinates of the 0-simplices (points) are represented geometrically; carriers of higher-dimensional simplices are defined as interpolations (usually linear): edges interpolate points, triangles interpolate edges, and so on. By contrast, geometry of many solid cells is defined by set operations, for example curves of intersection between second degree surfaces are commonly found in many mechanical parts. In such situations it may be more convenient to define the carriers of the 2-cells (surfaces) and to define the carriers of the 1-cells (edges) implicitly as subsets of the corresponding intersection curves. Thus, it should be clear that many cellular representations may be devised that would differ in the choice of which geometric carriers are specified explicitly and which are implied by the incidence relations between adjacent cells [132]. This determines in part which incidence information can be determined in constant time and which requires searching. Representing *all* incidence relationships in a cell complex substantially increases the size of the resulting data structure and is therefore deemed impractical.

The main advantage of the cell complexes over groupings is that they represent exactly and explicitly all topological information about the solid, including its interior, boundary, dimensional skeletons, and connectivity. No additional numerical computations are required to answer such topological queries, and all solids may be represented exactly – at least in principle. On the other hand, the graph representation of the incidence between the various cells and the high degree of geometric redundancy turns cell complex

¹²In other words, carrier is a generic dimension-independent term for possibly unbounded curve, surface, volume, etc.

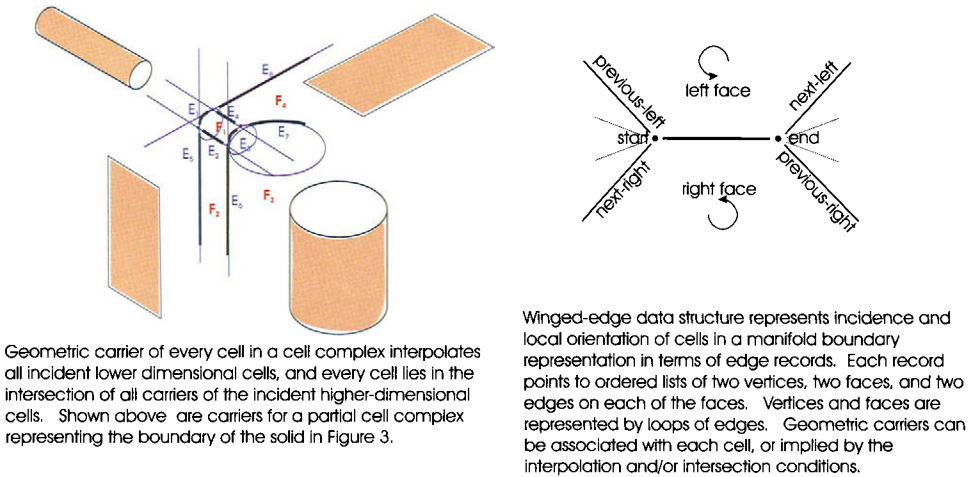


Figure 20.7. Geometric compatibility (intersection, interpolation) conditions and incidence information in cellular representations are not independent and must be maintained at all times.

representations into inherently serial linear size structures and complicates development of efficient algorithms. Maintaining the validity of cellular representations is a non-trivial task requiring a guarantee that all redundant information remains consistent at all times: all represented geometric carriers must satisfy all intersection and interpolation conditions, while the cells themselves must remain disjoint as required by the definition of the cell complex [101].

20.3.3. Boundary representation: a compromise

Historically, boundary representation was one of the first computer representations to be used for description of polyhedral three-dimensional objects [8,15], but both its strengths and weaknesses as a representation scheme can be appreciated better when examined in terms of properties of implicit and combinatorial representations. Recall that every solid X has the well-defined boundary ∂X , and the boundary of every three-dimensional solid in E^3 uniquely determines it.¹³ We can use this fact to represent the solid *implicitly* by its boundary, without enumerating points in the solid's interior. In other words, we represent the solid X by the predicate:

$$X = \{p \mid p \in \text{set bounded by } \partial X\} \tag{20.8}$$

and rely on the Jordan-Brouwer theorem (a generalization of the planar Jordan curve theorem) guaranteeing that ∂X separates the Euclidean space E^3 into exactly two sub-

¹³This statement is not as trivial as it may appear: the boundary of a semi-infinite set or the boundary of a curved face (for example lying on a sphere or a torus) in general does not uniquely determine the set it bounds, because there is usually another set with the same boundary.

sets, one of which is the bounded interior of X and the other is unbounded exterior space. Before we can declare this apparently implicit representation of solids a *bona fide* representation scheme, we need to give a computable point membership query on such a representation of X .

Suppose we have some representation for ∂X and we *know* that the represented set is the boundary of some unique solid X . If we pick two arbitrary points: point a in the interior of X and point b outside of X , then *any* path connecting a and b must intersect the boundary ∂X an odd number of times. Suppose we pick the point b to be always outside the solid X , then we can test if point a is inside or outside the solid by simply counting the number of times the path from a to b intersects ∂X : odd means a is *in* X , even means *out*. By far the easiest way to implement this is to choose any linear path from a to the point b at infinity, which reduces the point membership test to intersecting a linear ray with the set ∂X .

But how do we represent ∂X ? The answer is: any way we like, as long as we can guarantee that it is indeed an unambiguous representation of some solid's boundary and that the intersection with an arbitrary line segment can be computed. One could choose to represent the boundary ∂X using any of the implicit, constructive, parametric, or combinatorial methods we already described above. Most of the methods and various combinations have been tried, but the main challenge for any boundary representation is to assure that the represented set is indeed a boundary of a valid solid. The validity conditions follow clearly from the combinatorial model of solidity, making the combinatorial representation a natural choice. Specifically, since the boundary of every solid is a 2-cycle in E^3 , it must: (1) be a valid cell complex (disjoint cells satisfying the frontier); (2) be homogeneously two-dimensional (every lower-dimensional cell is in the closure of some 2-dimensional cell); (3) have every edge shared incident on an even number of faces; (4) be orientable, which means that the material side can be defined on every face in a globally consistent manner.

Furthermore, *every* such structure is guaranteed to represent a boundary of some valid solid.¹⁴ Many cellular representations have been proposed for boundary representations, but the oldest and the most popular representations enforced the above conditions only for manifold solids whose boundary points have neighborhoods homeomorphic to two-dimensional disks. In such boundary representations, every edge is shared by *exactly* two faces, as typified by the popular *winged-edge* data structure [8] illustrated in Figure 20.7. The incidence information in a winged-edge boundary representation consists of linked lists of edge records, with each record using eight pointers (pointing to two vertices, two faces, and four other edges) to enforce locally the orientability and manifoldness conditions. In this scheme, vertex and face records are defined by the ordered lists of the incident edges. Geometric carriers may be associated with any of the cells in the data structure parametrically or constructively, in a manner that satisfies all implied interpolation and intersection conditions, but the global non-interference conditions on cells are not enforced automatically and must be checked. A variety of other, similar in spirit, manifold boundary representations have been proposed in efforts to optimize the space, ease of manipulation, or handling of specific geometric carriers [23,29,132].

¹⁴In fact, every 2-cycle in E^3 is orientable, but explicit representation of orientation allows to check for this condition locally and to extend applicability of boundary representations to more general sets.

Boundary representations for non-manifold solids can be designed directly as representations of 2-cycles in Euclidean space or by treating the boundary of a non-manifold solid as a union of manifold shells.¹⁵ Both approaches also apply to boundary representations of arbitrary heterogeneous cell complexes, that can be viewed as a union of boundary-represented manifolds of various dimensions [30,46,64,129,134].

As cellular structures, boundary representations have a number of attractive properties. Because the boundary of a solid is unique, it is possible to invoke additional conditions (e.g., smoothness, connectedness, sign-invariance, orientation, and so on) in order to define and construct the unique canonical decomposition of the boundary [112]. Due to the dimensional reduction, boundary representations can be much smaller than most three-dimensional cellular representations of the same solid. Boundary representations also inherit the disadvantages of the cellular structures mentioned above, and in particular they are non-trivial to construct and maintain. Regularized set operations and other constructions that guarantee solidity can be implemented directly on boundary representations, but this approach requires support for non-manifold boundary representations because manifold boundary representations are not closed under such operations. Another alternative is to devise a set of direct operations on boundary representations that must preserve the validity of boundary representations at all times. Both approaches have been employed and we will briefly discuss them below in section 20.4.

20.3.4. Unification of representation schemes

Based on the assumed mathematical properties, we know that all of the above representation schemes are different methods for capturing complete geometric information about the *same* class of objects: semi-analytic subsets of Euclidean space. Therefore, it must be possible to convert such representations into each other, as may be required for different applications. This in turn suggests that all representation schemes are simply different ways to organize the same geometric and topological data. What is this data?

All representation schemes are organized in terms of a finite number of operations on a given set of *primitives*. The primitives are halfspaces in constructive representations, geometric carriers in cellular structures, and geometry of the cells in groupings. In all cases, the primitives are defined by analytic equalities and inequalities. The operations on primitives either produce new primitives (via interpolation, motion, deformation, etc.) or produce semi-analytic sets using set operations (\cap , \cup , $-$), closure \mathbf{k} , and selecting a connected component. All other operations and queries are simply compositions of these basic operations. In other words, a *fixed* finite set of primitives H gives rise to a representational space $M(H, O)$ consisting of the transitive closure of the primitives under the selected operations O [78]. It turns out that this representational space is *finite* (because only finitely many semi-analytic sets may be constructed using a finite set of primitives) and can be completely characterized by a particular stratification of space determined by the primitives in H [104,105].

Given a finite collection of primitives, H , consider the stratification of the whole Euclidean space E^3 such that every stratum is a maximal connected k -manifold that

¹⁵Strictly speaking, the resulting structure is not a proper cell complex, because it contains topologically distinct but geometrically coincident cells.

is also sign-invariant¹⁶ with respect to all primitives in H . Such a stratification exists for every collection of semi-analytic primitives, is unique, and is sometimes called *Whitney regular*. It has a reasonable low-degree polynomial size, and its strata satisfy the frontier conditions. Figure 20.3 shows some examples. The practical significance of this stratification lies in the fact that *every* representation (constructive and/or combinatorial) in the modeling space $M(H, O)$ is essentially an optimized union of some strata. For example, boundary representations consists of merged 0-, 1-, and 2-dimensional strata (vertices, edges, and faces respectively), CSG representations are regularized Boolean representations of unions of three-dimensional strata, and so on. Thus, the Whitney regular sign-invariant stratification of space serves as the lowest common denominator for all representation schemes and allows systematic development of algorithms and queries. See [104] for additional details.

20.4. ALGORITHMS

20.4.1. Fundamental computations

The unified view of solid representations allows to identify a small set of ‘fundamental’ operations: *primitive selection, stratification, point generation, point membership classification, set comparison, and ordering*. These operations are fundamental in the sense that most other algorithms can be defined through their composition. As with our classification of mathematical models, these operations are not entirely independent: for example, point membership may require stratification, and almost all operations require some form of comparison. Specific representation schemes are often optimized for several (but rarely all) of these fundamental operations. In practice, all fundamental operations except ordering can be implemented only approximately; see section 20.6.3 on standards and section 20.7.1 on unsolved problems for discussion of the ensuing difficulties.

Selection of geometric primitives

Before attempting any solid modeling computations, we must make sure that the set of known primitives and/or carriers is sufficient to represent the result of computation. Oversimplifying, the results of geometric computations are made up from portions of the candidate sets. In some cases, this is a trivial step: it is clear that in order to compute the intersection of a line with a solid’s boundary, we must have a representation of the line and representations of the surfaces bounding the solid. It may be less obvious (but true) that the boundary representation of a solid may be constructed from boundaries of the corresponding CSG primitives. But in many other situations, the candidate primitives are not obvious, must be determined as part of the computation, and are usually not unique. For example, it is well known that carriers found in a typical boundary representation are insufficient for the CSG representation of the same solid [109], and it is far from obvious which geometric primitives are needed to describe the boundary of a blend or a sweep [128].

¹⁶In other words, every primitive function in H has the same sign on all points of the stratum.

Stratification

Perhaps the most difficult of solid modeling computations is the process of identifying all cells in the Whitney regular sign-invariant stratification for a given set of primitives. The theoretical stratification process decomposes any semi-analytic set into a collection of connected smooth k -submanifolds by recursively extracting k -dimensional solid portions of the given set [64,130,131] (see Figure 20.3). But consider the simplest case of two primitives h_i and h_j . There are at most nine non-empty sign-invariant sets corresponding to the pairwise *intersections* of the sets defined by the signs of each primitive. Recall that intersection is also required for computing and representing the geometric carriers of lower-dimensional cells in a cell complex where only the geometry of the higher-dimensional cells is specified directly. In particular, boundary representations routinely require computation of the intersection curves between incident faces. Point classification against a boundary representation requires computing intersections of bounding surfaces with a line, and one of the most popular methods for visualizing implicit sets relies on ray-casting (intersecting the set with a grid of parallel lines). Furthermore, such intersection sets may be also disconnected, heterogeneous in dimension, or contain singularities and self-intersections. Thus, at the very least, stratification requires computing connected smooth submanifolds of the intersecting primitives satisfying the frontier condition, and it is not surprising that one of the chapters in this handbook (see the chapter on intersection problems by N. Patrikalakis and T. Maekawa) is devoted to the study of the intersection problems. While in theory any semi-analytic set may be stratified *exactly*, it appears that general and practical algorithms almost always resort to numerical methods.

Point membership classification (PMC)

We already discussed point membership queries in the context of individual representations schemes (constructive, combinatorial, and boundary). A more general PMC operates on an arbitrary point p and a representation of a set S , and returns *in*, *on*, or *out* depending on whether p is respectively in the interior, boundary, or outside of the solid S [119]. Since solidity, interior, and boundary are all topological concepts defined relative to some universal set X (for example E^3 , surface, curve), it should not come as a surprise that PMC of a point p usually requires computing the neighborhood of p relative to X . Neighborhoods of smooth and regular points on ∂S are adequately represented by the tangent (or normal) information at p . Other points may require non-trivial analysis and computations depending on the representation of S .

Point generation

Point generation is required to produce a single representative point from a set which may or may not have some special properties (such as center of mass), or to generate many such samples throughout the represented set with some (regular or irregular) intervals. Point generation is straightforward for sets represented parametrically or enumeratively, but may be difficult for other representations. For combinatorial representation, point generation is performed cell-by-cell serially or hierarchically, depending on how the cells are organized. For implicitly defined sets, points may be generated by sampling (and classifying using PMC) with desired resolution; for constructive representations, points may be generated for each of the primitives and then filtered using PMC test as described

above.

Comparison

Comparison of two points represented by their coordinates is relatively straightforward. But if the numbers are not exactly the same – and they rarely are – one must compute the *distance* between the two points using some appropriately defined metric. The most common type of distance is the usual Euclidean distance, but other metrics, such L_p and Hausdorff, are often useful and necessary. Comparing sets of points is a substantially more difficult task that depends on how the sets are represented, and what kind of metric is used. Notice that distance computation is a perfectly valid method for answering PMC queries, and comparison is usually required in order to decide whether two sets are incident on each other. Because neither implicit nor parametric representations for a set of points are unique in general, set comparison usually requires point generation, classification, and point comparison. Comparisons of different combinatorial representations may be formulated in terms of comparisons of individual cells. Finally, comparison of two solids may require development of metrics that take into account not only geometric distance, but also their topological form [14].

Ordering

A process of combining the results of the primitive computations into the representation of the result is difficult to describe generically, because the ordering process itself depends on the way the representation is organized. For constructive representations, ordering produces a tree of constructions; ordering of groupings produces either serial or hierarchical structures; and ordering of cell complexes involves creating and maintaining incidence and adjacency information.

A particularly elegant approach to ordering 2-manifold boundary representations takes advantage of the familiar Euler characteristic χ which is defined as an alternating sum:

$$\chi = V - E + F, \quad (20.9)$$

where V , E , and F are the numbers of vertices, edges, and faces respectively in the boundary representation. If boundary representation is *known* to be a connected manifold with every k -cell homeomorphic to a k -dimensional disk, then an even number χ provides complete classification of all such surfaces up to a homeomorphism. For a fixed χ , equation (20.9) can be viewed as a two-dimensional linear subspace of the three dimensional space defined by coordinates V , E , and F ; valid operations on the boundary representation may change the numbers of vertices, edges, and faces, but all resulting boundary representations must be confined to the same plane. Such valid operations are termed *Euler operators* (because they preserve the Euler characteristics) and can be viewed as vectors in the two dimensional linear subspace [54]. Because a two-dimensional linear space can be spanned by two linear independent vectors, only two (independent but not unique) Euler operators are required to build a boundary representation for any polyhedron homeomorphic to the sphere.

This counting principle can be extended to more complex boundary representations with more general cells and multiple connected components (shells) [16]. For example, simple counting arguments show that $\chi = R + 2(S - n)$ where R is the number of interior

face rings (counted by internal loops on the faces), S is the number of connected shells, and n is the genus. Eliminating Euler characteristic, χ , from equation (20.9), we obtain a linear constraint

$$V - E + F - R - 2(S - n) = 0, \quad (20.10)$$

representing a five-dimensional hyperplane in the six-dimensional space and implying that at least five distinct Euler operators are necessary and sufficient to span the space of all such cell complexes. Euler operators conveniently enforce the required combinatorial conditions on such cell complexes,¹⁷ but they are not unique, and other approaches to constructing and maintaining ordering are possible [29].

20.4.2. Enabling algorithms

All other geometric queries and algorithms in solid modeling may be constructed using sequences of the above fundamental computations. It would be impractical to describe here all algorithms important in solid modeling, but it is instructive to consider how some of the more common and critical computations can be cast in terms of the fundamental operations. Specifically, we focus on those algorithms that can be broadly classified as *representation conversions* because they tend to re-represent the solid in a manner that makes desired computations simpler; these algorithms shaped the solid modeling technology, enabled specific applications, and defined the architecture of the commercial systems. The following description is optimized for clarity; efficient algorithms involve essentially the same steps but are structured to take advantage of locality, proximity, coherence, symmetry, and hierarchy, as well as specific assumptions and known properties of particular geometric representations.

Ray casting

Ray-casting and ray-tracing are popular techniques for rendering solids and their boundaries [98], for representing a solid as a grouping of line segments, for performing PMC against a boundary representation of a solid, as well as for performing other types of analysis [26]. Such algorithms require computing the intersection of a given (possibly unbounded) line segment with a representation of the solid. For all representations, the ray-casting algorithm reduces to computing the intersection of the unbounded candidate line with each of the geometric primitives (leaves of a CSG tree or carriers of the highest dimensional cells) yielding an unsorted list of points along the line. Those points that classify *in* or *out* the solid are discarded, as well as those points that classify *out* with respect to the line segment. The remaining points are *on* the solid and bound one or more linear segments – the result of the intersection. These points are sorted along the line in order to induce in/out classification for the segments of the line they bound. The result must be regularized in the topology of the line (because we only want solid line segments), which may require constructing one dimensional neighborhoods of points and/or additional PMC tests. Ray casting is particularly popular with the CSG representations due to the elegant divide-and-conquer algorithm (similar to the merge sort) that merges ordered lists of intersection points for every interior node of the tree [119].

¹⁷Note Euler operators do *not* guarantee the validity of the results unless additional geometric conditions are satisfied as well.

Sampling and polygonization

Many of the application algorithms require sampling and/or approximating the represented solid. These include rendering, path generation (for motion planning or machining), computation of integral properties, and finite element meshing. Broadly, all such algorithms produce a k -dimensional grouping or a simplified cell complex from a given solid's representation.

To generate points (0-dimensional grouping) on the solid's boundary, one generates the points on the boundaries of candidate primitives (geometric carriers in boundary representation or boundary of primitives in the CSG tree), and classifies these candidate points against the given representation of the solid. Similar processes may be used for generating points in the interior of the solid, except that the candidate points must be generated throughout the three-dimensional space, for example on the regularly-spaced grid throughout the space containing the solid. Ray-casting (see above) can be used to generate 1-dimensional groupings.

Two-dimensional groupings are usually made up from triangles and polygons constructed from points generated on the solid's boundary. The construction may require that the vertices of every edge and/or polygons must have the same classification with respect to solid's faces or edges, if such information is available. The constructed edges and polygons may be further ordered into a valid cell complex satisfying the usual combinatorial conditions.

Popular three-dimensional sampling include octrees and tetrahedralization. Octrees are constructed by recursively classifying orthogonal boxes with respect to the solid as *in*, *out*, and those intersecting the boundary of the solid. The latter are subdivided further until the desired level of resolution is reached. The box/solid classification is in turn reduced to classifying the vertices of the box, intersecting edges or faces of the box with the solid's boundary, and ordering of the result. The octree cells may be further subdivided into tetrahedra; tetrahedra may be also constructed directly from a 0-dimensional grouping of points sampled or generated in the interior of the solid, for example using the Delaunay constraint [102].

Set membership classification

This is a more general computation that subsumes PMC, ray-casting, cell sampling, and many other computations in the following sense [119]. Given representations of two sets: X is a *candidate* set, S is a *reference* set; both are solids but do not have to be of the same dimension. The SMC procedure $M(X, S)$ partitions the candidate set with respect to the reference set

$$M(X, S) = \langle X_{inS}, X_{onS}, X_{outS} \rangle \quad (20.11)$$

into the three solid portions of X . In this context, solidity is defined with respect to X . When X is a single point, SMC reduces to PMC described under the fundamental computations; when X is a curve segment and S is a solid, SMC is the curved ray casting procedure outlined above. More generally, SMC subsumes many other geometric computations in solid modeling. For example, when both X and S are solids, X_{inS} is their regularized intersection; when S is a two-dimensional face and X is a curve lying in the same surface, X_{inS} is the portion of the curve contained in the region bounded by the

face; and so on. SMC could be considered itself as one of the fundamental computations, except that it is not implemented directly but must be reduced to some sequence of the other fundamental computations, as illustrated by the examples above.

Boundary evaluation and merging

The queen of all representation conversion procedures – both in complexity and its importance in solid modeling – is the so called *boundary evaluation* procedure that produces a valid boundary representation of a solid given its constructive representation [89]. For CSG representations, the procedure conceptually is straightforward. If the solid S is represented by a sequence of regularized set operations on collection of primitives $\{h_1, h_2, \dots, h_n\}$, then it is not difficult to show that

$$\partial S \subset (\partial h_1 \cup \partial h_2 \cup \dots \cup \partial h_n) \quad (20.12)$$

In other words, the boundary representation of the same solid may be stitched together from the boundary pieces of the CSG primitives. Which pieces? Those pieces that classify *on* with respect to the given CSG representation; they are also called *faces* in the boundary representation [112]. In generic terms, the boundary evaluation reduces to performing $SMC(\partial h_i, S)$ for every primitive, and representing the union of the result. This in turn requires partitioning ∂h_i into the candidate pieces that may (or may not) lie on the solid's boundary. For efficiency, we want the pieces to be as big as possible, but they need to be small enough so that we do not miss any portion of ∂S . It can be shown that a sufficient (but not necessary) partition is obtained by intersecting ∂h_i with the boundaries of all other primitives in the given CSG representation. Each portion of ∂h_i bounded (or *trimmed*) by the intersection curves becomes a potential candidate face, and a single PMC test for any point in the interior of the candidate face is sufficient to determine if the face belongs to the boundary representation or not. Each face that passes the *on* test must be represented in the resulting boundary representation, typically (but not necessarily) by *its* boundary which consists of the *segments* (i.e., connected 1-dimensional manifold strata) of the intersection curves, called *edges*. A sufficient set of candidate edges is obtained by intersecting each intersection curve with all other surfaces, but only those with points classifying *on* with respect to the face belong to the boundary representation.

Thus, a typical boundary evaluation algorithm involves computing intersection curves between the primitive surfaces, computing intersection between the curves and the surfaces, generating points in the tentative curves and faces, followed by PMC testing these points with respect to solids and faces, and ordering the passing edges and faces into a valid cell complex. Consider now a very special case when there are only two solids h_1 and h_2 – both given by their boundary representations – and combined using either regularized union \cup^* or regularized intersection \cap^* operation. Following the steps in the above procedure, we would have to compute intersection between the two sets $\partial h_1 \cap \partial h_2$, which involves trimming the faces and edges in the two boundary representations against each other, and merge the resulting pieces into a new boundary representation. This special but important case of boundary evaluation is often called *boundary merging*.

The conceptual structure of the CSG-to-boundary evaluation procedure provides a recipe for all other types of boundary evaluations. For example, suppose one wants

to perform boundary evaluation for a constructive representation containing the sweep operation defined by (20.7). The general procedure involves exactly the same steps as before: generate a sufficient set of candidate surfaces, trim the surfaces against each other to produce a set of candidate faces and edges, generate a point in each candidate cell, perform PMC against the sweep representation, and order the results into a cell complex. It is intuitive and can be shown formally that

$$\partial(\text{sweep}(S, M)) \subset \text{sweep}(\partial S, M), \quad (20.13)$$

which means that the candidate surfaces must be obtained by sweeping the edges and faces in the boundary of the given solid. This could be challenging for general boundary representations and motions, and various approximations for such sweep surfaces have been proposed using sampling and polygonization techniques. Boundary evaluation for other constructive representations requires similar sequences of fundamental computations.¹⁸ Practical implementations are usually optimized to generate the smallest possible number of candidate faces and edges, never repeat the same computations, and employ coarse spatial tests to localize computations whenever possible.

Other representation conversions

It can be shown that all other computations in this category can be designed systematically using sequences of the same fundamental steps [105]. For example, CSG representations may be computed from a given boundary representation roughly as follows: select the sufficient set of primitives; intersect all primitives to produce the usual stratification of the whole space; generate at least one point in every sign-invariant three-dimensional component in the stratification; classify the generated points (and therefore the corresponding components) against the given boundary representation; the regularized union of the components classifying *in* correspond to the disjunctive canonical ‘sum-of-products’ CSG representation which can be further optimized using Boolean optimization techniques. The last step relies on repeated point membership tests and comparison. The most difficult step in this procedure is the selection of a sufficient set of primitives, which has been solved for restricted but common types of boundary representations. More details on boundary to CSG conversion may be found in [108,109].

Other representation maintenance utilities – from comparing and optimizing constructive representations and approximating solids by groupings, to computing wireframes, silhouettes, and meshes – all may be systematically designed following similar procedures composed from the same fundamental computations [105].

20.5. APPLICATIONS

An inexhaustible variety of applications may be developed with the help of fundamental computations and enabling algorithms. Below we briefly survey those popular engineering applications that helped to shape solid modeling as we know it today.

¹⁸This assumes that a construction is properly defined and supports an unambiguous point membership test; unfortunately, such definitions may not always be available even for very common constructions, such as blending

20.5.1. Geometric design

Historically, the activity of geometric design is largely devoid of physical analysis, but seeks efficient methods for creating, modifying, visualizing, and annotating geometric representations of solid shapes. By definition, constructive representations are well suited to constructing and modifying the models using high-level engineering parameters: distances, angles, radii, coordinate systems, etc – all can be encoded as parameters of the constructions. The resulting solid shape (although implicit) may be controlled by modifying these parameters. At the same time, constructive representations provide *no* explicit information about the boundaries, which makes them difficult to visualize, modify locally, or annotate. By contrast, combinatorial representations, and boundary representation in particular, are ideal for visualization and annotation tasks that require direct access and traversal of a solid's boundary. But because combinatorial representations are difficult to construct and manipulate directly, they appear to be ill-suited for design activities.

To facilitate subsequent editing of solid models (a critical issue, since the vast majority of designs are in fact modifications of earlier designs), both constructive and combinatorial representations are often supplemented by constraints, expressed as equations and inequalities on the parameters in the constructive representations and/or on the coordinates of the carriers on the combinatorial representations [49,92,114]. Typical constraints include tangency, incidence, perpendicularity, distances, angles, and so on. More recent approaches apply such constraints directly on the cells in a stratification underlying the constructive parametric model [9]. To effect a desired modification, the user typically changes the values of constraints and/or some parameters; these changes are reflected in the updated systems of equations that are solved for the new values of all other parameters and coordinates, leading to a new instance of the solid model [66]. See the chapter on parametric modeling by Christoph Hoffmann and Robert Joan-Arinyo for more details.

20.5.2. Analysis and simulation

Most of the solid modeling computations in this category may be abstracted by single-valued functions of one or more solids (and possibly other variables) and have recognizably correct answers. Some of the more popular applications include rendering, computation of integral properties [47], assembly modeling, interference detection, simulation of mechanisms, and NC (numerical control) machining simulation.

Rendering and computation of *mass properties* are in fact quite similar in the sense that both require some form of finite enumeration of the points in – or rather the cells in a grouping associated with – the solid's interior or its boundary. In the case of integral property computations (which include volume, surface, inertial properties, and other integrals of functions defined over the solid's interior or boundary), the contribution of each individual cell is added to the result. In the case of rendering, the contribution of every cell is displayed on the screen. Two main principles used for such computations are *sampling* and *dimensional reduction*. Dimensional reduction relies on the generalized Stokes theorem to reformulate the computation over solid S in terms of another computation over the boundary ∂S [53]. Thus, the volume integral over S may be computed directly or reformulated as a surface integral over ∂S , and the integrals over individual faces may sometimes be reformulated in terms of the path integrals over the edges bounding the face. Similarly, rendering may be performed by visualizing solid cells drawn on the screen

in the depth-first order, boundary cells using the surface normal information, or by drawing the edges and silhouette curves generated from the boundary representation. Sooner or later, the computation reduces to evaluating some function over a relatively simple constituent cell: a line or curve segment, a triangle, a polygon, a tetrahedron, a cube, etc. Evaluation of polynomial functions over linear polyhedral cells may be performed exactly (within the machine precision) [51], but for more general functions and/or cells evaluation is performed only approximately based on the function's values at carefully chosen (e.g. Gauss) points in the cell.

Assembly is a collection of solids that are positioned and oriented by some some rigid motions in space, subject to mating and non-interference conditions [45]. At the very least, the mating conditions identify pairs of contacting surfaces and thus require explicit boundary information. Non-interference between two solids A and B requires that their regularized intersection $A \cap^* B$ is empty (the solids interiors are disjoint); empty non-regularized set intersection $A \cap B$ implies that there is no contact between the boundaries ∂A and ∂B . Non-interference conditions may be defined and computed with any unambiguous representation, but recall that deciding 'emptiness' of constructive representation is a non-trivial matter.

A *moving solid* is easily represented by applying the rigid motion to the coordinate system in which the solid is designed. A *mechanism* with two solids A and B is an assembly where A moves *relative* to (the coordinate system of) B . Thus the *static* mating and non-interference conditions must be enforced at all times, resulting in more complex *dynamic* conditions. Dynamic non-interference between two moving solids A and B may be formulated as a static non-interference between stationary B and $sweep(A, M)$, where M is the motion of A relative to B . The dual operation of $unsweep(A, M)$ can be used to formulate and compute queries about largest/smallest non-interfering objects [39]. The continuous motion may be also simulated discretely by evaluating the static solid configuration in small time increments. See [50] for a recent survey. Maintaining proper assembly conditions at all time steps may require substantial computing resources.

The simple pairs of solids may be chained together into graph structures to model *mechanism linkages*, such as robot arms. The mechanism motion is instantiated by combining the individual relative motions according to the graph by a procedure called *forward kinematics*, which involves multiplication of the matrices representing the individual relative motions. The inverse kinematics algorithms require solving the systems of non-linear equations to determine the individual relative motions, given the motion of some point or coordinate system on the mechanism. Mechanism modeling may be viewed as a natural extension of the constructive representation that allows using continuous motions (as opposed to instances of motions used to position a solid in space) [121].

Simulating *numerically controlled (NC) machining* is an application similar to mechanism modeling: solid cutter A is moving relative to solid stock B which is fixed in some solid fixture C . Motion of the cutter A is determined by the NC program, and the purpose of the simulation is to determine whether the moving cutter A removes the desired amount of material from B without interference with the fixture C . The material removal operation is modeled by the regularized difference operation: at every time step, the material removed by the solid cutter A is subtracted from the stock B . A more accurate approximation of the NC machining process requires computing $sweep(A, M)$, estimating

volume removal rates, and other integral properties. See [59] for discussion of these and related issues.

20.5.3. Dynamic analysis and lumped-parameter systems

Simulation of a rigid body motion under the externally applied forces and moments requires computation of a solid's mass properties (mass, moments of inertia, center of mass), solving for the acceleration of the solid at that time instant, and integrating it through time to modify the solid's velocity. This in effect approximates the solution of the ordinary differential equation of solid's motion. Repeating this computation at small discrete time intervals produces realistic simulation of motion. A more sophisticated simulation applies the same procedure to the *system* of rigid bodies interconnected at joints, solving the constrained system of ordinary differential equations [22]. In this case, the physical object under consideration is better represented combinatorially as a *graph* of solids, whose links specify the types of the motions allowed at the joints, and no interference is allowed between the individual solids at any time during motion. The latter task requires dynamically tracking the distances and identifying the collisions between all moving solids, which can be handled, albeit at a high computational cost, by the standard solid modeling methods [50]. However, the contact geometry of joints in such a mechanism structure is usually *assumed*; computing it would require full power of solid modeling discussed above plus adequate models of contact and friction mechanics [7].

The next logical step is to enhance the simulation model with a proper model of impact mechanics, and predict the motion after an impact takes place. In addition to the need to identify precisely the time and type of all possible contacts, it is apparent that the contacting solids usually constitute a non-manifold geometric model, and can be also considered as a mechanism with a "contact" joint at the time of impact. Commercial implementations of such simulations are already available, but they are all limited by the lack of good models of impact mechanics – that must include mechanical deformations and/or experimentally-determined coefficients of restitution.

A one-dimensional graph structure of interacting solids can be employed with more general systems of ordinary differential equations arising in other branches of physics. Such graph structures have been known for a long time [71] and are now used by commercial software systems. The graphs are composed from nodes that represent 'lumped'¹⁹ constitutive properties, such as masses, inertia, dampers, resistances, inductances, spring constants, and so on, plus a finite number of transducers that convert and couple different physical models in a single structure. In this context, spatial information serves three purposes: computation of the lumped properties associated with each solid, visualizing the response of the system in terms of the solids, and managing the spatial incidence and adjacency of the lumped elements based on interaction between the solids.

20.5.4. Planning and generation

In contrast to analysis and simulation applications, planning and generation tasks do not typically have a unique answer but must produce one or more acceptable solutions from

¹⁹'Integrated' may be a more proper way to described 'lumped' [107].

some usually large spaces of feasible answers.²⁰ An application in this category is usually constructed as a heuristic search procedure based on repeated querying of known solid models – typically using the fundamental operations described in section 20.4.1.

Motion planning requires finding a collision-free path for a moving object with respect to one or more solids [44]. In addition to the numerous popular robot motion planning tasks, other applications in this category include generation of tool paths for NC machining and automatic inspection plans by coordinate measurement machines. *Feature recognition* is a process of matching the portions of a represented shape to previously known parameterized forms, shapes, or processes, e.g., for the purpose of constructing a manufacturing process plan or identifying parts for inspection purposes [40]. The vast majority of such features appear to be defined by relationships between portions of a solid’s boundary: faces, edges, special points, incidence, convexity, symmetry, specific sizes, and so on. Another important example of an application in this category is *mesh generation*, which is a process of constructing a cell complex representation for a given solid (called a mesh), satisfying specific requirements on the size, shape, number, and topology of cells. These requirements are defined by some analysis application that seeks to approximate the answer to a boundary-value problem using a particular type of spatial discretization of the domain (finite element, finite differences, hexahedral, simplicial, etc.) All meshing procedures require generating points in the solid’s interior and boundary and ordering them to form a complex that has the same topology as, and conforms geometrically (as closely as possible) to the solid’s boundary [68].

It has been clear from the outset that the guaranteed validity of solid models also holds a great promise for *automatic design* of engineering artifacts [125]. The constructive representations appear to be particularly well suited for the task, if a way can be found to relate the construction parameters to the desired design and/or functional characteristics. The constructive representation may then be viewed as a procedural definition – a computer program that implements the design algorithm and outputs a valid solid-represented design. Indeed a number of languages and grammars for generating such solid-represented designs have been proposed, but all faced challenges because evaluation of the generated solids required not only explicit geometric and combinatorial information, but also physical and functional information that is normally not present in a geometric modeling system [107].

20.5.5. Manufacturing

Effective application of solid modeling techniques is possible for many traditional manufacturing processes that are idealized as *unit processes* [63] with well-characterized input and output geometry. Simulation, planning, and verification of unit processes, particularly machining and assembly, were some of the main catalysts for the development of solid modeling [86]. More recently, the range of supported applications has been greatly expanded to include sheet metal manufacturing, injection molding, stamping, pipe routing, and so on. Such applications are supported within the solid modeling framework through specialized user interfaces that force designers to create models in terms of a

²⁰The two types of problems are not entirely independent of course; since many of the analysis and simulation computations are approximate, the ‘correct’ answers are not unique, and in fact may depend on the more difficult planning and generation tasks.

limited application-specific lexicon. For example, machined parts would be designed in terms of pockets, holes, and other machining features; sheet metal parts would be designed by sequences of bending and stamping operations; draft angles are automatically added to injection-molded parts; and so on. Usually, such operations are translated into application-specific constructive representations that are then evaluated into combinatorial (usually boundary) representations. In addition, heuristic knowledge-based systems are emerging that are capable of identifying common geometric features that may (or may not) be supported by a particular manufacturing process.

Beyond traditional manufacturing, if every solid can be approximately represented on a computer by a grouping of some k -dimensional cells, why not manufacture it by a computer-controlled process that will grow the solid by depositing the individual k -cells? This basic idea underlies a host of new manufacturing techniques that are often referred to as layered-manufacturing (because they build the solid layer by layer), solid imaging or printing (because they employ deposition processes that are reminiscent of the paper printing methods), or solid free-form fabrication (because the cell-by-cell deposition process removes many of the manufacturability constraints on the solid's shape [55]). For a layered process to work correctly, it must work with valid solid representations accepted from different sources. A *defacto* standard representation for this purpose is called STL: a boundary representation constructed as a collection (a 2-grouping) of triangular oriented 2-cells. The main virtue of this representation is its simplicity; but as with most groupings, STL representation is only approximate, lacks information on incidence between the cells, is sensitive to round-off and precision errors, and as such, is quite unreliable.

The ability to rapidly manufacture solids of arbitrary shape without specialized tooling naturally leads to an attractive idea of object copying. This involves sampling points on an existing object and constructing its solid representation (a process often called *reverse engineering*), followed by a layered reproduction process [124].²¹ Detailed discussion of reverse engineering can be found in the chapter by Martin of this handbook. Briefly, reverse engineering is a process of constructing a solid's representation from an unorganized cloud of points that are sampled on the boundary of an existing physical object with some accuracy. In terms of fundamental solid modeling operations, the process involves comparing and ordering the points into strata of dimensions 0 (points), 1 (edges), and 2 (faces) whose union represents the boundary of the object being reconstructed.

20.6. SYSTEMS

20.6.1. Classical systems

We saw that each constructive and combinatorial approach to solid modeling offers specific practical advantages when it comes to implementing particular engineering applications. Theoretically, the two approaches are equivalent, because both are informationally complete with respect to the postulated mathematical models, and therefore are capable of supporting all of the fundamental and enabling computations discussed above. But the constructive representations are parameterized, concise, and robust representations that come with a built-in point-membership test and are a natural choice for creating,

²¹A more general notion of reverse engineering should include to reproducing the functionality, as well as the shape, of the artifacts.

editing, and programming solid models; they are a poor choice for applications requiring point generation, boundary traversal, and persistent spatial addressing. In contrast, the combinatorial representations provide explicit, persistent, and spatially addressable enumeration of the solid's boundary (and possibly interior) that is perfectly suited for applications requiring point generation and sampling; they also appear to have a greater geometric coverage, but combinatorial representations are verbose, more sensitive to errors and inconsistencies, and are difficult to create and manipulate directly.

The early solid modeling systems were designed around single carefully chosen representation schemes, usually either the CSG or the boundary representation, but by the mid-1980's virtually all single representation systems were enhanced and extended with auxiliary representation and features, making them in fact hybrid *dual-representation* systems [87,88]. The emerging architecture for such a dual-representation solid modeling systems is shown in Figure 20.8(a). The constructive representation facilitates model creation and editing, often in terms of application specific constructions that reflect individual user's concepts and mentality, while the associated combinatorial representation supports other applications requiring generation, traversal, and spatial referencing. Computations may be performed against either representation. The combinatorial representation usually takes the form of a boundary representation, an approximate polygonal model, or an octree that is produced automatically from the constructive representation. Note that the representation conversion is one way: always from the constructive representation to the combinatorial one – partly because the inverse conversion problem is more difficult, but mostly because the conversion process does not usually capture the constructive semantics of particular applications. This implies that the combinatorial representations should not be allowed to be edited or modified directly, because this could lead to inconsistency between the two representations.

20.6.2. Parametric interaction

But by the late 80's, parametric modeling acquired one new and decisively critical ingredient: the constructions and constraints were applied interactively and incrementally by making direct references to the *previously constructed* cellular representation of geometry [21,110]. The resulting construction method was strongly reminiscent of the datum-based dimensioning system used by engineers, and the resulting graphical user interface resonated well with designers whose primary job was to produce engineering drawings fast and without mistakes.

The new parametric solid modeling systems have a complex multi-layered architecture (shown schematically in Figure 20.8(b)) that combines constructive and combinatorial representations with constraint-solving and heuristic algorithms [35,110]. As in the earlier systems, direct modifications of cells in the boundary representation is not allowed in this architecture, because this may lead to a loss of consistency with the corresponding construction; however, evaluation of every parametric edit does modify the cells in the boundary representation thereby affecting all future constructions that refer to this cell. Recall also that all combinatorial representations (including the boundary representations) support persistent spatial addressing, which means that every cell in the representation has a unique name that identifies the set of points associated with this cell. By contrast, the constructive representations define sets of points that may be indistinguishable from

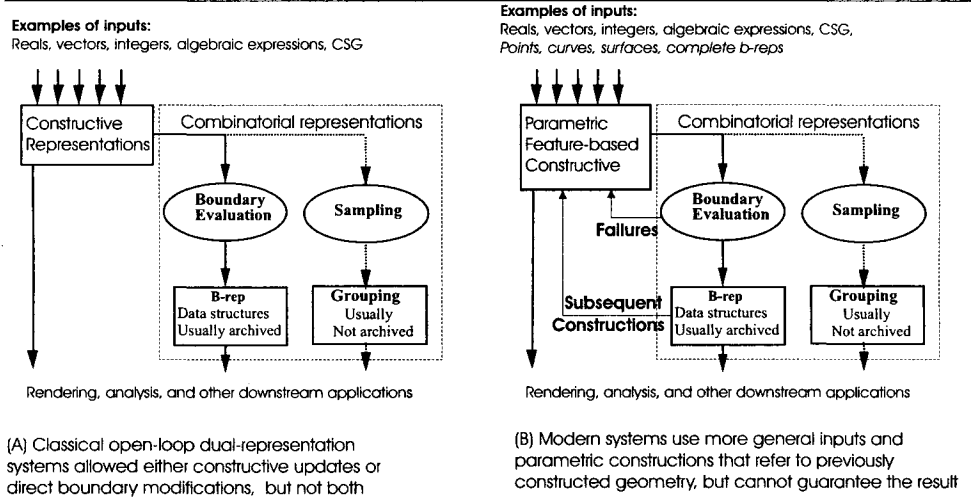


Figure 20.8. The architecture of the classical dual-representation solid modeling systems is defined and limited by the representation conversion technology; the architecture of the new parametric systems is shown side-by-side.

each other, as is the case with the connected components of a disconnected set defined by intersection of simpler sets.²² When a solid modeling construction refers to a particular cell in the boundary representation (for example, a new hole is positioned with respect to an existing reference face), it assumes that the cell is persistent. But the cell itself was evaluated from an earlier construction, and may have a different name should the boundary model be regenerated at a different time with modified parameters or conditions. If the name of the reference cell changes, all future constructions change their semantics, resulting in a drastically different and unpredictable behavior(see example in Figure 20.9). The problem of assigning unique names to the cells in the boundary representation as they are generated from a parametric definition came to be known as *persistent naming*. It has been characterized formally and solved under certain conditions in [76]; several heuristic approaches have also been proposed [19,41], but no general solution is known at this time. Thus, it should come as no surprise that the architecture of parametric modeling system as illustrated in Figure 20.8(b) allows for the failed operation feedback loop. The parametric solid modeling systems do produce solid models, but not all constructions have well-defined semantics or are guaranteed to succeed, and however solid the results may be, they are often not predictable or repeatable.

But let us suppose for a moment that we completely solve the two difficult problems of persistent naming and of the representation conversion. Would that eliminate the appar-

²²The underlying mathematical problem is much more difficult than it may appear: ordering connected components of an implicitly semi-analytic set without computing its boundary or combinatorics is as difficult as enumerating the multiple roots for a system of equations before it is solved[105].

ent difficulties in the parametric modeling? Hardly. Consider the solid model generated in response to the parametric change shown in Figure 20.6.2. Is the solid produced by the system correct? Maybe or maybe not, depending on your *definition* of correctness. In retrospect, it should not be surprising that parametric modeling does not guarantee the properties of the results beyond solidity. The classical mathematical models assumed in the scenario of Figure 20.1 correspond to instances, and do not reflect the parametric nature of the modeling systems.

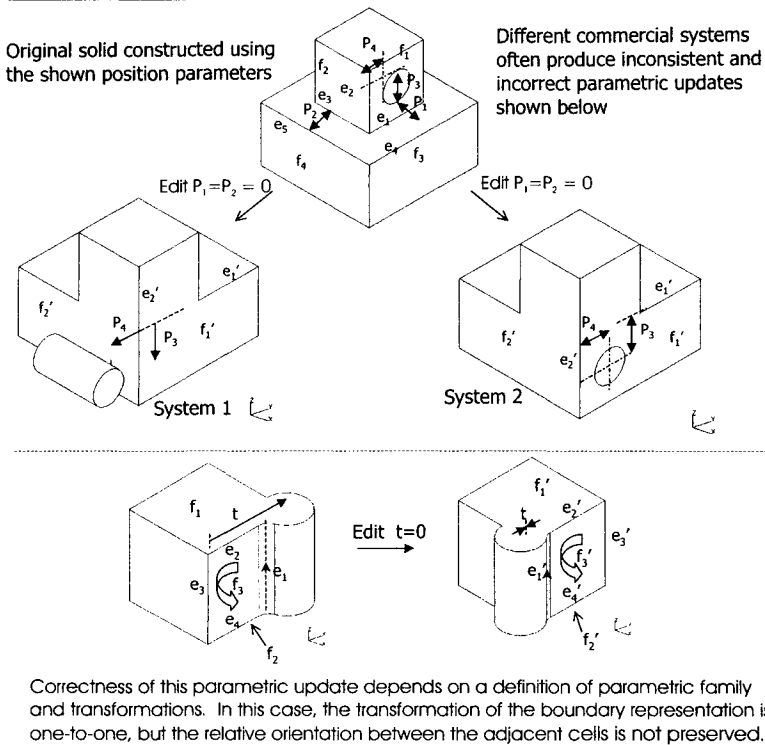


Figure 20.9. In the absence of standard mathematical models, parametric solid modeling systems often produce incorrect or inconsistent results.

20.6.3. Standards and interfaces

Different users and applications often rely on different systems and representation schemes. The common approaches to the critical problem of data exchange and transfer include native translation, neutral standard file format, and standardized programming interfaces. The most general and ubiquitous approach is the neutral format because it does not require knowing *a priori* which systems are involved and does not lead to proliferation

of special-purpose translators. Such formats were designed for the common CSG and boundary representations in early 80's and were later adopted as part of the international standards for product data description [67]. Work is currently underway to extend such formats for representation of parametric representations by including specification of common parametric constraints and constructions. This should allow systems to exchange representation of a nominal solid and the syntax for an associated parametric family, but cannot guarantee that this family is valid or is the same in all systems (see discussion above) [74].

Most commercial solid modeling systems provide an application programming interface (API) that allows to access and compute with models in a given system. Standardizing on such an API is another way to alleviate the problem of data exchange. A number of limited proposals for such standard APIs have been made recently as surveyed in [74]. Since the stratification of a solid model into cells provides a common theoretical framework for unifying all representations, it also provides a basis for designing a representation-free API that is both formal and general. The initial effort towards that goal is described in [6].

All existing approaches to data exchange and standardization are further undermined by the accuracy and robustness problems in solid modeling. Recall that the fundamental operations, in particular stratification and point membership classification, cannot be computed exactly and must be approximated in *all* systems. Different systems make widely varying and often incompatible assumptions about accuracy and precision of these and other approximations, at times making data exchange and standardization difficult or even impossible.

20.7. CONCLUSIONS

20.7.1. Unsolved problems and promising directions

Continuing improvements in computer representations, algorithms, and general computing technology have led to mature industrial strength implementation of the modeling scenario shown in Figure 20.1. It is safe to predict that the improving data structures for increasingly complex geometric shapes, relatively efficient and tested algorithms for fundamental and enabling computations taking advantage of the latest techniques in hardware, and the increasing computing power will result in further evolutionary progress in specialized situations and the ever-growing array of engineering and consumer applications. But these advances alone are not likely to solve the major outstanding problems in solid modeling, because they require substantial revision of the fundamental premises and the assumed mathematical models. A number of such issues and promising directions are discussed below in a logical order that does not necessarily correspond to their importance or priority.

Robustness of geometric modeling computations and systems has remained a challenge, despite numerous advances in accuracy and consistency (see [27] for a sample of recent work). The fundamental issue is that the theory of geometric and solid modeling is based on the classical model of exact geometry, but engineering data and computations are almost always approximate. Strictly speaking, this means that no theorem of exact geometry can be assumed to hold in the approximate model, and it must be proved again

in the presence of errors. It is not enough to compute the results very accurately and consider all special cases, because introduction of *any* errors may lead to inconsistent results and contradictions. An ultimate solution of the robustness problem requires recognizing the imprecise nature of the mathematical models and reformulation of the key concepts and computations. For example, what is the meaning of the approximate stratification (or intersection) of imprecise primitives and how is it related to the approximate point membership test?

Persistent naming of cells in a combinatorial representation in terms of the primitives and operations in the corresponding constructive representation is required in order to support regeneration, editing, and exchange of parametric models. Mathematically, the problem reduces to the difficult problem of indexing the connected components of an implicitly represented set. The most promising approach is to devise a new class of constructive representations, where every such component is named persistently by construction, for example using interactive datums and references.

Parametric families of solids is a widely accepted, but poorly understood, notion. Informally, since we are dealing with families of physical objects, it is reasonable to expect that small changes in the values of parameters of a given solid should result in another solid that is similar or is “near” the original solid. This should in principle eliminate the jumps, sudden changes, and other unpredictable behaviors observable in current systems. Formally, this assumption corresponds to the notion of *continuity* of the mapping from the parameter space into the space of subsets of E^3 and/or their representations [76]. And therein lies the major difficulty of parametric modeling: for a given representation scheme, there is more than one way to define the notion of continuity, and furthermore, different solid representations schemes imply different parametric families. Specifically, there are several distinct methods for defining continuous parametric families based on a boundary representation [76], and they vary widely in their computational properties.²³ As expected, CSG and boundary representations of the same solid normally imply very different families [110] that may be related using tools from category theory [75]. Since most parametric systems rely simultaneously on the constructive and the boundary representations, generating and classifying solids in the combined family, as well as maintaining consistency between the distinct parametric families has emerged as a major technical challenge [34,77] that must be met if there is any hope for computer interpretation of parametric models, such as those required in shape optimization or for standardization of parametric definitions.

Tolerancing and metrology includes issues of accuracy, variability, measurement, and reconstruction, that are important in all manufacturing applications. One of the pillars of modern mass manufacturing, the doctrine of *interchangeability* of mechanical parts in an assembly, calls for precise geometric specification of when a part may replace another part in an assembly without affecting the functionality [36]. This suggests strongly that proper models for mechanical parts must include the notion of mechanical variation or *tolerance* and the procedure for deciding when two such parts are interchangeable. A

²³The main technical task is that of deciding whether or not two boundary representations belong to the same continuous family requires constructing explicit maps that may be difficult or impossible with the existing data structures, because they were designed to model solid instances and do not always properly record changes in orientation and dimension [76].

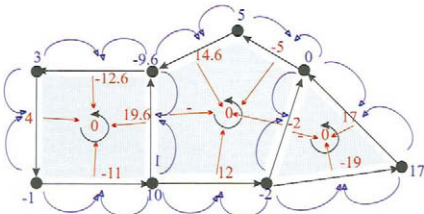
simple minded approach to mechanical tolerancing is to endow some or all parameters of solid's representation with variations in $\pm\epsilon$ range. While it is easy to implement, this approach suffers from several severe limitations, including restriction to perfect-form and ambiguous semantics. A *geometric dimensioning and tolerancing* (GD&T) approach that is more consistent with modern mechanical engineering practices is based on the notions of datums and tolerance zones and gives the basis for various national and international standards. Several researchers proposed mathematical extensions of solid modeling theory to accommodate GD&T [14,81], it is now clear that the formal semantics of the national standards are not completely defined. Despite massive efforts to mathematize the standards [116], it will be some time before a standard approach to computational modeling of tolerances can be defined [126], because this may require fundamentally reformulating the notions of variational control [117]. In the absence of a common mathematical standard, tolerances are being treated syntactically as attributes that are attached as labels to solid's features and surfaces. Nominal solid models can be used to plan sampling of the points on the surface of the corresponding actual parts, and numerous commercial packages will numerically fit the sampled data to curves and surfaces (typically using proprietary algorithms), but no such fitting algorithms are standard today.

Interoperability refers to the ability of solid modeling systems to exchange and compute on the models constructed by other systems or applications[90]. It plays the role of virtual interchangeability for computer models of mechanical parts. As such, full interoperability subsumes the issues of standardization, representation conversions, robustness, and well-defined semantics for parametric models and tolerances. Furthermore, it is clear that the very notion of interoperability depends on the application and the type of queries. Two solid models may be considered equivalent for the purpose of space packaging studies and point membership testing, but may not be interchangeable when it comes to performing engineering analysis, parametric studies, or manufacturing process planning. Systematic formulation and solution of such problems is a critical and fruitful area of research.

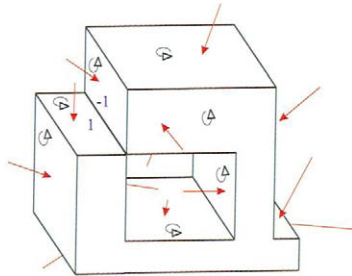
Physical field modeling is a natural generalization of solid modeling aimed at modeling spatially and temporally distributed physical properties. In a nutshell, the problem amounts to developing methods for representing, constructing, and manipulating discrete and continuous variations of physical quantities defined over a given geometric domain, subject to the postulated physical laws. For example, classical solid models presume material homogeneity, but new modeling techniques are needed to support design, analysis, and manufacturing of objects with materials that vary heterogeneously and anisotropically, resulting in products with superior structural properties, compliant mechanisms, and embedded sensors [72]. Similarly, engineering analysis requires representing and computing assumed physical properties (displacement, energy, temperature, stress, flux, etc.) over the solid's interior and boundary. The predominant approach to such problems requires conforming approximation of a given solid by a particular type of spatial discretization that supports numerical computations for the problem at hand. Finite-element and finite-difference meshing have emerged as a major research area and a substantial bottleneck to advances in engineering analysis [68]. From a practical point of view, such representation conversions are computationally intensive and numerically sensitive procedures that are difficult to automate; proliferation of different and often incompatible techniques fur-

ther undermines the standardization, robustness, integration, and interoperability efforts described above.

More fundamentally, modeling of physical fields should generalize and subsume continuum and combinatorial solid modeling, as well as constructive and combinatorial representations. A suitable continuum generalization of an r -set is based on the concept of a *fiber bundle* [42,135] with the solid model playing the role of the base space and the fiber space corresponding to the field defined over the solid. The corresponding generalization of the combinatorial model may be defined using algebraic topological *cochains* over finite cell complexes [69,123]. Just as the notions of boundary and cycles are needed to define valid boundary representations, the dual notions of coboundary and cocycles formally capture the combinatorial physical balance laws that must be satisfied by all valid field models (see Figure 20.10). It is reasonable to expect that the two models of the field problems – the continuum using fiber bundles and the combinatorial using cochains, provide different characterizations of the field that will lead to computationally complementary representations. For example, the fiber bundle model naturally leads to new methods for field description and analysis that are not limited by the traditional mesh-based methods and difficulties [100,106,135], while the cochain models of physical distributions have been instrumental in developing new geometric languages for describing physical phenomena [20] and improved numerical techniques that preserve the indicated physical laws [56].



Coboundary operation on k -cochain transfers the coefficients from every k -cell to all incident $(k+1)$ -cells with +/- sign depending on relative orientation. Addition of coefficients yields the total quantities that enter the $(k+1)$ cells through their boundary. Repeating the coboundary operation produces a $(k+2)$ -chain with all 0-coefficients.



A k -cochain associates a coefficient with every k -cell to represent a distribution of a physical quantity over a cell complex – in this case, a 2-cochain for a vector quantity distributed over the boundary of the solid. The cochain is a cocycle if its coboundary is 0, corresponding to the model of physical equilibrium with respect to the physical quantity.

Figure 20.10. Cochains, coboundaries, and cocycles as combinatorial models for physical fields. Compare to Figure 20.4.

Applications and systems will continue to get faster, more robust, and sophisticated, but the breakthroughs in solid modeling technology require solving many of the above problems and introducing new mathematical models and paradigms. Thus, application of solid modeling to engineering problems involving complex physics, deformation, and

phase changes (for example, those arising in etching, micromachining, compression molding, metal casting, and so on) requires either complete geometric characterization of the controlled transformations, or merging solid and physical field modeling techniques. By contrast, the current trend towards function- and criterion-driven design requires rethinking the role of geometric models in synthesis of mechanical artifacts, which over centuries has been reduced to catalogue search and *posteriori* numerical analysis. Each engineering function usually influences only *some* of the design shape, while the same geometry usually serves multiple functions [38]. Somewhat paradoxically these observations suggest that synthesis tools may be more effective with *partial* geometric information at different stages; this in turn demands updating the classical notions of informational completeness, validity, and membership classification in solid modeling to include more general physical and process characteristics.

20.7.2. Summary

Solid modeling was conceived as a universal informationally complete geometric language for describing physical artifacts in support of industrial automation. It has had a dramatic effect on those areas where the classical notions are sufficient. The evolutionary progress in solid modeling during the last decade led to dramatic improvements in speed, reliability, domain coverage, and widespread use of commercial solid modeling systems.

At the same time, solid modeling grew and expanded to the point where it is no longer adequately supported by the original theoretical foundations. Many emerging applications cannot guarantee the correctness of results, because they are often based on heuristic and/or idealized geometric analysis of the physical problems. Since the guarantee is not always possible or is too expensive to compute, heuristic and sometimes wrong answers seem to be tolerated, as long as they can be generated quickly and checked by a human user or another system for correctness and then used for the downstream applications, such as producing annotated engineering drawings, manufacturing process planning, or engineering analysis. Validity and guarantee appear to have been replaced by an effective and iterative paradigm that demands speed and interactivity, while supporting and encouraging incremental improvements in the solid modeling technology.

Taking a longer term view of the field, further progress in solid modeling requires development of new mathematical models to support computer representations of increasingly complex physical artifacts. A pragmatic approach is to assume that such models may be application dependent. In particular, design, analysis and manufacture of engineering systems is driven by manufacturing processes, physical criteria, variability and incomplete information. This suggests that the corresponding notion of the “informational completeness” of such solid models must be modified to recognize that these and other attributes are at least as important as the associated nominal geometry.

Acknowledgments

This work was supported in part by the National Science Foundation grants DMI-9502728, DMI-9900171, DMI-0115133, and CCR-0112758. This survey was written during my sabbatical stay at the Dip. Informatica e Automazione, Universita' Roma Tre, Italy. I am grateful to Alberto Paoluzzi for numerous stimulating discussions over the course of writing this survey, to Jeff Chard, Horea Ilies, Earlin Lutz, Malcolm Sabin, Nick Sapidis

and Herb Voelcker for careful reading and criticism of the preliminary versions of this paper, and to Myung-Soo Kim for his patience and encouragement.

REFERENCES

1. *ACM Symposium on Solid Modeling and Applications*, ACM Press, 1991, 1993, 1995, 1997, 1999, 2001.
2. K. Abdel-Malek, D. Blackmore, and K. Joy. Swept volumes: foundations, perspectives, and applications. *International Journal of Shape Modeling*, 2001. submitted.
3. M.K. Agoston. *Algebraic Topology*. Marcel Dekker Inc, New York, 1976.
4. P.S. Aleksandrov. *Combinatorial Topology, Volume 1*. Graylock Press, Rochester, New York, 1956.
5. F. Arbab. Set models and boolean operations for solids and assemblies. Technical report, University of Southern California, 1985.
6. C. Armstrong, A.A. Bowyer, S. Cameron, J. Corney, G. Jared, R. Martin, A. Middleditch, M. Sabin M, and J. Salmon. *Djinn: A Geometric Interface for Solid Modeling*. Information Geometers, Winchester, UK, 1998.
7. D. Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10(2-4):292-352, 1993.
8. B.G. Baumgart. *Geometric Modeling for Computer Vision*. Computer Science, Stanford University, 1974.
9. R. Bidarra, K.J. de Kraker, and W. Bronsvort. Representation and management of feature information in a cellular model. *Computer-Aided Design*, 30(4):301-313, 1998.
10. J. Bloomenthal and K. Shoemake. Convolution surfaces. *Computer Graphics*, 25(4):251-256, 1991.
11. J. Bloomenthal and B. Wyvill. Interactive techniques for implicit modeling. *Computer Graphics*, 24(2):109-116, March 1990.
12. P.J. Booker. *A History of Engineering Drawing*. Northgate Publishing, London, 1979.
13. A. Bowyer. *SVLIS - Introduction and User Manual*. Information Geometers, Winchester, UK, 1994.
14. M. Boyer and N. Stewart. Modelling spaces for toleranced objects. *International Journal of Robotics Research*, 10(5):570-582, 1991.
15. I. Braid. The synthesis of solids bounded by many faces. *Communications of the ACM*, pages 209-216, 1975.
16. I.C. Braid, R.C. Hillyard, and I.A. Stroud. Stepwise construction of polyhedra in geometric modeling. In K. W. Brodlie, editor, *Mathematical Methods in Computer Graphics and Design*. Academic Press, 1980.
17. E. Brisson. Representing geometric structures in d dimensions: Topology and order. *Discrete and Computational Geometry*, 9:387-426, 1993.
18. P. Brunet and I. Navazo. Solid representation and operation using extended octrees, 1990.
19. V. Capoyleas, X. Chen, and C.M. Hoffmann. Generic naming in generative, constraint-based design. *Computer-Aided Design*, 28(1):17-26, 1996.

20. J.A. Chard and V. Shapiro. A multivector data structure for differential forms and equations. *IMACS Transactions Journal, Mathematics and Computers in Simulation*, 54:33–64, 2000.
21. X. Chen and C.M. Hoffmann. Towards feature attachment. *Computer-Aided Design*, 27:675–702, 1995.
22. J. Cremer and A. Steward. The architecture of Newton, a general-purpose dynamics simulator. In *IEEE International Conference on Robotics and Automation*, pages 1806–1811. IEEE Press, 1989.
23. D.P. Dobkin and M.J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Proc. Third Symp. on Computational Geometry*, pages 86–99, 1987.
24. R. Egli and N.F. Stewart. A framework for system specification using chains on cell complexes. *Computer-Aided Design*, 32:447–459, 2000.
25. G. Elber and M.-S. Kim, editors. Special issue on sweeps and minkowski sums. *Computer-Aided Design*, 1999.
26. J. Ellis, G. Kedem, T. Lyerly, D. Thielman, R. Marisa, J. Menon, and H. Voelcker. The raycasting engine and ray representations. In *Proceedings of ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 255–267. ACM Press, 1991.
27. S. Fortune, editor. Special issue on implementation of geometric algorithms. *Algorithmica*, 27(1), 2000.
28. J.E. Goodman and J. O'Rourke, editors. *CRC Handbook of Discrete and Computational Geometry*, Chapter on Solid Modeling, pages 863–880. CRC Press LLC, Boca Raton, FL, 1997.
29. L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4:74–123, 1985.
30. E.L. Gursoz, Y. Choi, and F.B. Prinz. Vertex-based representation of non-manifold boundaries. In M. Wozny, J. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 107–130. North Holland, 1990.
31. H. Hironaka. Triangulation of algebraic sets. In *Proceedings of Symposia in Pure Mathematics, Algebraic Geometry, ARCATA 1974, Vol. 29*, pages 165–185, 1975.
32. J.G. Hocking and G.S. Young. *Topology*. Dover Publications, New York, 1961.
33. C.M. Hoffmann. *Geometric and Solid modeling*. Morgan Kaufman, USA, 1989.
34. C.M. Hoffmann and K.-J. Kim. Towards valid parametric CAD models. *Computer-Aided Design*, 33(1):81–90, 2001.
35. C.M. Hoffmann and J.R. Rossignac. A road map to solid modeling. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):3–10, March 1996.
36. D. Hounshell. *From the American System to Mass Production, 1800-1932*. The Johns Hopkins University Press, 1984.
37. K.C. Hui. Solid modelling with sweep-CSG representation. *Proceedings of CSG 94 Set Theoretic Solid Modelling: Techniques and Applications*, pages 119–131, 1994.
38. H. Ilies and V. Shapiro. An approach to systematic part design. In *Proceedings of the 5th IFIP WG5.2 Workshop on Geometric Modeling in CAD*, pages 383–392, may 1996.
39. H. Ilies and V. Shapiro. The dual of sweep. *Computer-Aided Design*, 31(3):185–201,

March 1999.

40. Q. Ji and M.M. Marefat. Machine interpretation of CAD data for manufacturing applications. *ACM Computing Surveys*, 29(3):264–311, 1997.
41. J. Kripac. A mechanism for persistently naming topological entities in history-based parametric solid models. *Computer-Aided Design*, 29(2):113–122, 1997.
42. V. Kumar, D. Burns, D. Dutta, and C. Hoffmann. A framework for object modeling. *Computer-Aided Design*, 31:541–556, 1999.
43. K. Kuratowski and A. Mostowski. *Set Theory*. North-Holland Publishing Co., USA, 1976.
44. J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.
45. K. Lee and D.C. Gossard. A hierarchical data structure for representing assemblies: Part 1. *Computer-Aided Design*, 17, 1, 15 – 19, 1985.
46. S.H. Lee and K. Lee. Partial entity structure: A compact non-manifold boundary representation based on partial topological entities. In *Sixth ACM Symposium on Solid Modeling and Applications*, Ann Arbor, MI, pages 159–170, 2001.
47. Y. Lee and A. Requicha. Algorithms for computing the volume and other integral properties of solids II: a family of algorithms based on representation conversion and cellular approximation, 1982.
48. P. Lienhardt. Subdivisions of n -dimensional spaces and n -dimensional generalized maps. In *Proc. Fifth ACM Annual Symposium on Computational Geometry*, pages 218–227. 1989.
49. R. Light and D.C. Gossard. Modification of geometric models through variational geometry. *Computer-Aided Design*, 14(4):209–214, 1982.
50. M. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In *IMA Conference on Mathematics of Surfaces*, pages 602–608. San Diego, CA, 1998.
51. S.L. Lien and J.T. Kajiya. Symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra. *IEEE Computer Graphics and Applications*, 4(10):35–41, 1984.
52. S. Lojasiewicz. *Triangulations of Semi-Algebraic Sets*, volume 18 of 3. Annali della Scuola Normale Superiore di Pisa, 1964.
53. E.D. Lutz. *Numerical Methods for Hypersingular and Near-Singular Boundary Integrals in Fracture Mechanics*. PhD thesis, Cornell University, Computer Science Department, May 1991.
54. M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, Maryland, USA, 1988.
55. A. Marsan and D. Dutta. A survey of process planning techniques for layered manufacturing. In *Proc. 1997 ASME Design Technical Conferences*, Sacramento, CA, 1997.
56. C. Mattiussi. An analysis of finite volume, finite element, and finite difference methods using some concepts from algebraic topology. *J. Comput. Phys.*, 133:289–309, 1997.
57. J.C.C. McKinsey and A. Tarski. On closed elements in closure algebras. *Annals of Mathematics*, 47(1):122–162, 1946.
58. D. Meagher. Geometric modelling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.

59. J. Menon and H. Voelcker. Toward a comprehensive formulation of nc verification as a mathematical and computational problem. In *Proceedings of the 1992 Winter Annual Meeting of ASME*, Volume 59, pages 147–164. Anaheim, CA, 1992.
60. J. Menon and H. Voelcker. The completeness and conversion of ray representations of arbitrary solids. In *Proc. of ACM Symposium on Solid Modeling and Applications*, pages 175–186. May 1995.
61. A.E. Middleditch. Applications of vector sum operator. *Computer-Aided Design*, 20(4):183–188, 1988.
62. J.R. Munkres. *Elements of Algebraic Topology*. Addison-Wesley, Reading, Massachusetts, 1984.
63. National Research Council Unit Manufacturing Process Research Committee. *Unit Manufacturing Processes: Issues and Opportunities in Research*. National Academy Press, 1995.
64. M.A. O'Connor and J.R. Rossignac. SGC: A dimension independent model for pointsets with internal structures and incomplete boundaries. In *IFIP/NSF Workshop on Geometric Modeling, Rensselaerville, NY, 1988*. North-Holland, September 1990.
65. N. Okino, Y. Kakazu, and H. Kubo. TIPS-1: technical information processing system for computer-aided design, drawing, and manufacturing. In J. Hatvany, editor, *Computer Languages for Numerical Control*, pages 141–150. North-Holland, Amsterdam, 1973.
66. J.C. Owen. Algebraic solution for geometry from dimensional constraints. In J. Rossignac and J. Turner, editors, *Proc of the First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 397–408, 1991.
67. J. Owen. *STEP: An Introduction*. Information Geometers, Winchester, UK, 1993.
68. S. Owen. A survey of unstructured mesh generation technology. In *7th International Meshing Roundtable*. Dearborn, Michigan, Oct. 26–28 1998.
69. R.S. Palmer and V. Shapiro. Chain models of physical behavior for engineering analysis and design. *Research in Engineering Design*, 5:161–184, 1993.
70. A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics*, 12(1):56–102, 1993.
71. H.M. Paynter. *Analysis and Design of Engineering Systems*. The MIT Press, Cambridge, MassFachusetts, 1961.
72. M.J. Pratt. Modelling of material property variation for layered manufacturing. In *Mathematics of Surfaces IX*, Cambridge, UK, 2000.
73. M.J. Pratt. Solid modeling. In *Encyclopedia of Computer Science and Technology*, Volume 42 (Supplement 27), pages 295–333. Marcel Dekker, New York, NY, 2000.
74. M.J. Pratt and B.D. Anderson. A shape modelling application programming interface for the step standard. *Computer-Aided Design*, 33:531–543, 2001.
75. S. Raghothama. *Models and Representations for Parametric Family of Parts*. PhD thesis, Department of Mechanical Engineering, Univeristy of Wisconsin-Madison, September 2000. Spatial Automation Laboratory Technical Report, SAL-2000-5.
76. S. Raghothama and V. Shapiro. Boundary representation deformation in parametric solid modeling. *ACM Transactions on Graphics*, 17(4):259–286, October 1998.

77. S. Raghoothama and V. Shapiro. Consistent updates in dual representation systems. *Computer-Aided Design*, 32(8–9):463–477, 2000.
78. A. Rappoport. Geometric modeling: A new fundamental framework and its practical implications. In *SMA '95: Proceedings of the Third Symposium on Solid Modeling and Applications*, pages 31–41, 1995.
79. A.A.G. Requicha. Mathematical models of rigid solid objects. Technical report, TM-28, PAP, University of Rochester, Rochester, NY, November 1977.
80. A.A.G. Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Computing Surveys*, 12(4):437–464, December 1980.
81. A.A.G. Requicha. Towards a theory of geometric tolerancing. *International Journal of Robotics Research*, 2(4):45–60, 1983.
82. A.A.G. Requicha. Solid modeling: a 1988 update. In B. Ravani, editor, *CAD Based Programming for Sensory Robots*, pages 3–22. Springer-Verlag, 1988.
83. A.A.G. Requicha and J.R. Rossignac. Constructive non-regularized geometry. *Computer-Aided Design*, 23(1):21–32, January 1991.
84. A.A.G. Requicha and R.B. Tilove. Mathematical foundations of constructive solid geometry: General topology of closed regular sets. Technical report, TM-27a, PAP, University of Rochester, Rochester, NY, June 1978.
85. A.A.G. Requicha and H.B. Voelcker. Constructive Solid Geometry. Technical report, TM-25, PAP, University of Rochester, Rochester, NY, November 1977.
86. A.A.G. Requicha and H.B. Voelcker. An introduction to geometric modeling and its applications in mechanical design and production. In J.T. Tou, editor, *Advances in Information Systems Science*, Vol. 8. Plenum Publishing, 1981.
87. A.A.G. Requicha and H.B. Voelcker. Solid modeling: A historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, Volume 9–24, March 1982.
88. A.A.G. Requicha and H.B. Voelcker. Solid modeling: Current status and research directions. *IEEE Computer Graphics and Applications*, 3(7):25–37, October 1983.
89. A.A.G. Requicha and H.B. Voelcker. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 73(2):30–44, January 1985.
90. Research Triangle Institute. Interoperability cost analysis of the U. S. automotive supply chain. Technical Report 99-1, NIST, 1999.
91. A. Ricci. A constructive geometry for computer graphics. *Computer Journal*, 16(3):157–160, May 1973.
92. J.R. Rossignac. Constraints in constructive solid geometry. In F. Crow and S.M. Pizer, editors, *Proc. of 1986 Workshop on Interactive 3D Graphics*, pages 93–110. ACM Press, 1986.
93. J.R. Rossignac. Through the cracks of the solid modeling. In S. Coquillart, W. Strasser, and P. Stucki, editors, *From Object Modelling to Advanced Visualization*. Springer-Verlag, 1994.
94. J.R. Rossignac and A.A.G. Requicha. Offsetting operations in solid modeling. *Computer Aided Geometric Design*, 3(2):129–148, 1986.
95. J.R. Rossignac and A.A.G. Requicha. Solid modeling and beyond. *IEEE Computer Graphics and Applications*, 12(5):31–44, September 1992.

96. J.R. Rossignac and A.A.G. Requicha. Solid modeling. In J. Webster, editor, *Encyclopedia of Electrical and Electronics Engineering*. Webster, John Wiley & Sons, 1999.
97. J.R. Rossignac and H.B. Voelcker. Active zones in CSG for accelerating boundary evaluation, redundancy elimination, interference detection, and shading algorithms. *ACM Transactions on Graphics*, 8(1):51–87, 1989.
98. S. Roth. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18(2), 1982.
99. V.L. Rvachev. *Theory of R-functions and Some Applications*. Naukova Dumka, Kiev, 1982. in Russian.
100. V.L. Rvachev, T.I. Sheiko, V. Shapiro, and I. Tsukanov. Transfinite interpolation over implicitly defined sets. *Computer Aided Geometric Design*, 18:195–220, 2001.
101. M. Sabin and V. Shapiro. Modifications in cellular models. Technical report, NA-09, DAMTP, University of Cambridge, UK, September 1998.
102. N. Sapidis and R. Perucchio. Advanced techniques for automatic finite element meshing from solid models. *Computer-Aided Design*, 21:248–253, 1989.
103. V. Shapiro. Theory of R -functions and applications: A primer. Technical report, TR91-1219, Computer Science Department, Cornell University, Ithaca, NY, 1991.
104. V. Shapiro. *Representations of Semi-Algebraic Sets in Finite Algebras Generated by Space Decompositions*. PhD thesis, Cornell University, Cornell Programmable Automation, Ithaca, NY, 14853, February 1991.
105. V. Shapiro. Maintenance of geometric representations through space decompositions. *International Journal of Computational Geometry and Applications*, 7(4):383–418, 1997.
106. V. Shapiro and I. Tsukanov. Meshfree simulation of deforming domains. *Computer-Aided Design*, 31(7):459–471, 1999.
107. V. Shapiro and H. Voelcker. The role of geometry in mechanical design. *Research in Engineering Design*, 1(1), 1989.
108. V. Shapiro and D.L. Vossler. Construction and optimization of CSG representations. *Computer-Aided Design*, 23(1):4–20, January 1991.
109. V. Shapiro and D.L. Vossler. Separation for boundary to CSG conversion. *ACM Transactions on Graphics*, 12(1):35–55, January 1993.
110. V. Shapiro and D.L. Vossler. What is a parametric family of solids? In *Proc. of the Third ACM Symposium on Solid Modeling and Applications, Salt Lake City, Utah, May, 1995*.
111. N. Shareef and R. Yagel. Rapid previewing via volume-based solid modeling. In *Proceedings of the Third Symposium on Solid Modeling and Applications*, pages 281–292. 1995.
112. C.E. Silva. Alternative definitions of faces in boundary representations of solid objects. Technical report, TM-36, PAP, University of Rochester, Rochester, NY, November 1981.
113. J.M. Snyder and J.T. Kajiya. Generative modeling: A symbolic system for geometric modeling. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 369–378, 1992.
114. L. Solano and P. Brunet. A system for constructive constraint-based modelling. In B.

- Falcidieno and T. L. Kunii, editors, *Modeling in Computer Graphics*, pages 61–83. Springer-Verlag, New York, 1993.
115. A.I. Sourin and A.A. Pasko. Function representation for sweeping by a moving solid. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):11–18, 1996.
 116. V. Srinivasan. Recent efforts in mathematization of asme/ansi y14.5m standard. In *Proc. Third CIRP Seminars on Computer Aided Tolerancing*, pages 223–232. Editions Eyrolles, Paris, 1993.
 117. V. Srinivasan. A geometric product specification language based on a classification of symmetry groups. *Computer-Aided Design*, 31(11):659–668, 1999.
 118. W.C. Thibault and B.F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Computer Graphics*, 21(4):153–162, 1987.
 119. R.B. Tilove. Set membership classification: A unified approach to geometric intersection problems. *IEEE Transactions on Computer*, C-29(10), October 1980.
 120. R.B. Tilove. Exploiting spatial and structural locality in geometric modeling. Technical report, TM-38, PAP, University of Rochester, Rochester, NY, October 1981.
 121. R.B. Tilove. Extending solid modeling systems for mechanism design and kinematic simulation. *IEEE Computer Graphics and Applications*, 3(3):9–19, 1983.
 122. R.B. Tilove and A.A.G. Requicha. Closure of boolean operations on geometric entities. *Computer-Aided Design*, 12(5):219–220, September 1980.
 123. E. Tonti. *On the Formal Structure of Physical Theories*. Istituto Di Matematica Del Politecnico Di Milano, Milan, 1975.
 124. T. Varady, R. Martin, and J. Cox. Reverse engineering of geometric models — an introduction. *Computer-Aided Design*, 29:255–268, 1997.
 125. H. Voelcker. Modeling in the design process. In W. Dale Compton, editor, *Design and Analysis of Integrated Manufacturing Systems*. National Academy Press, Washington, DC, 1988.
 126. H.B. Voelcker. The current state of affairs in dimensional tolerancing: 1997. *Integrated Manufacturing Systems*, 9(4):205–217, 1998.
 127. H.B. Voelcker and A.A.G. Requicha. Research in solid modeling at the University of Rochester: 1972–1987. In L. Piegl, editor, *Fundamental Developments in Computer-Aided Modeling*. Academic Press, London, 1993.
 128. W. Wang and K. Wang. Geometric modeling for swept volume of moving solids. *IEEE Computer Graphics and Applications*, pages 8–17, 1986.
 129. K.J. Weiler. *Topological Structures For Geometric Modeling*. PhD thesis, Rensselaer Polytechnic Institute, 1986.
 130. H. Whitney. Elementary structure of real algebraic varieties. *Annals of Mathematics*, 66(3):545–556, November 1957.
 131. H. Whitney. Local properties of analytic varieties. In S.S. Cairns, editor, *Differential and Combinatorial Topology, A Symposium in Honor of Marston Morse*, pages 205–244. Princeton University Press, 1965.
 132. T.C. Woo. A Combinatorial Analysis of Boundary Data Structure Schemata. *IEEE Computer Graphics and Applications*, 5(3):21–40, 1985.
 133. J. Woodwark. Blends in geometric modeling. In R. Martin, editor, *The Mathematics of Surfaces II*, pages 255–297, 1987.
 134. Y. Yamaguchi and F. Kimura. Nonmanifold topology based on coupling entities.

- IEEE Computer Graphics and Applications*, 15(1):42–50, January 1995.
135. J. Zagajac. *Engineering Analysis Over Subsets*. PhD thesis, Cornell University, Ithaca, NY, May 1997.

Chapter 21

Parametric Modeling

Christoph M. Hoffmann and Robert Joan-Arinyo

Parametric solid modeling is a key technology to define and manipulate solid models through high-level, parameterized steps. These steps can be modified by users and instantiated to specific parameter values and constraint configurations. More than that, the design paradigm supported allows the shape designer to define entire families of shapes, not just specific instances. We review the core techniques of parametric modeling, describe new trends, and sketch a number of open problems that must be resolved to take full advantage of the potential of parametric modeling.

21.1. INTRODUCTION

A parametric solid can be defined as a solid whose actual shape is a function of a given set of parameters and constraints upon them. The objective of *parametric solid modeling*, hereafter also referred to as *parametrics*, is to represent, manipulate, and reason in a computer about the three-dimensional shape of parameterized solid objects.

Prior to the development of parametrics, designers of solid models created a particular shape. Once created, editing and altering the shape was not specifically supported. To accomplish that, the designer had to import the shape and modify it by additional design steps. In contrast, parametric design focuses on the steps creating a shape and parameterizes them. This allows the designer to define an entire class of shapes that later on can be simply instantiated. The added flexibility can be exploited in many ways, and constitutes an important advance in solid modeling and its applications in, e.g., product design.

This overview of parametric solid models covers the two main components, constraints in Section 21.4 and in features in Section 21.5. While constraints comprise a well-defined set of tools and techniques, features are a more loosely-knit vocabulary. Feature semantics evolves with applications that seek to conceptualize, in a high-level vocabulary, major design steps and components. The multiplicity of application requirements and agendas makes features a less precisely cast subject that continues to be debated.

We also explore trends we perceive in parametric modeling. Those trends bifurcate into issues especially of interest to academics and issues of immediate interest to industry. There is overlap, of course, and we will work out key aspects in Section 21.6. Naturally, the trends throw up open problems, to be described more fully in Section 21.7.

21.2. PARAMETRIC MODELS

The foundations of solid modeling were laid by the pioneering work of Requicha and Voelcker in the late 1970s for constructive solid geometry (CSG), in which solid shapes are composed from instantiated primitives using set-theoretic operations. Their careful investigation of the topological and geometric foundations of the representation of rigid solids applies to rigid solid models in general, including the boundary representation models that arose around the same time from the work by Braid and Eastman. A survey of the state of the art in 1982 is found in [61].

The framework that originates with the CSG work captures models that have a geometric and a topological structure. The geometric structure relates to the actual shape of the solid surface, and the topology to adjacency and connectedness of the solid interior and its boundary surface. Such models can be characterized as semi-algebraic point sets, and we refer to them as *specific* solid models.

In contrast, a parametric solid model is more than a specific solid because it includes a metastructure from which specific solid models can be derived as instances. Thus, it is more appropriate to think of a parametric solid as a class of specific solid models, and so very different representational schemes have been proposed for them. See for example [60]. The representational proposals divide into procedural representations and mathematical ones. In procedural representations a specific solid shape is constructed by elaborating a sequence of construction steps. In mathematical representations an attempt is made to characterize variational classes of solids by postulating properties such as, e.g., that all members of the class have the same topological genus. To add to the diversity, note that the procedural representations may include nonprocedural substeps. For instance, a cross section to be extruded may be defined by a set of geometric constraints, with more than one solution, and the selection of a particular solution may depend on the constraint solver employed.

The procedural approach is unsatisfactory to some because it does not explicitly characterize a class of solids that can be derived from a common procedural representation. However, the mathematical approach is unsatisfactory as well to some because it has difficulty capturing properties accepted in practice. Those properties are based on a veiled intuition grounded in application requirements or in the particulars of an evaluation mechanism. At this point in time, there is no satisfactory definition of the term *variational class* of solids that has broad acceptance. The field thus moves through territory whose foundations are not fully understood, propelled by technological advances that arise from needs of applications. In view of this incomplete state of knowledge, we offer the following working definition for parametric models:

A parametric solid model is an information structure that permits deriving specific solid models, in the sense of Requicha and Voelcker, using a deterministic algorithm. Moreover, the specific shape derived depends on parameters that

are explicit in the information structure and must be valued for obtaining a specific solid shape.

Our commitment to the procedural school, apparent in this definition, reflects the current state of technology and practice. Note also that we understand a parametric solid model to comprise all specific solid shapes that are derivable from the representation. Some authors have called this a *variational family*, [70].

The bulk of tools incorporated into parametric models and their evaluation are geometric constraints and feature operations. Variant modeling is a precursor to this concept and has closer ties to specifics of the model representation or creation. We explain those concepts in separate sections. In addition, operations such as deformations of solid shapes have been considered, but are found predominantly in experimental solid modeling systems. We do not discuss them further.

21.3. VARIANT MODELING

If the objective is to shift from an instance design to a generic one, a simple technique is to prepare a variant design. Using the representation as a symbolic system, parameters can be identified and valued in different ways to generate variant designs. For example, consider the CSG expression $BLO(W, H, D)$ that evaluates to a block, in standard position, of width W , height H and depth D . Understanding the quantities W , H , and D as parameters, we can instantiate many blocks. This paradigm can be broadened by parameterizing complex expressions built from parametric solid primitives and embedding the expressions into a programming language that permits computing parameter values procedurally. Clearly, we can parameterize the transformation expressions that place the primitives in relation to each other, form conditional branches that may or may not evaluate component shapes based on specific parameter valuations, and abstract a design by encapsulating dependent parameters and exposing independent ones. We call this design approach, first demonstrated by the PADL-2 system, [8], *variant design*.

A slightly different variant design approach [45], implemented by Joan-Arinyo at the Universitat Politècnica de Catalunya in 1993, derives parameters from a symbolic abstraction of design gestures. Ultimately, a program is derived that generates design instances based on pre-defined parameters observed from a visual design gesture. For example, in ducting and pipe design, we may work with a repertoire of standard shapes, to be parameterized in a predefined way and placed sequentially in a way the user defines. Here, the design system can derive the design structure from the user interface gestures and create the variant design.

The variant methodology is especially well suited for applications that deal with those product families that are composed of standard basic shapes with simple parameterization. Moreover, the variation in net shape should be small. See for example [77]. Variant designs survive in libraries of standard parts. For example, there are libraries of fasteners, brackets, and so on, that are essentially derived from a few variant designs and indexed by a catalogue. Some limitations of variant design are explained in, e.g., [54].

Recent developments aim to improve the methodology by providing full support for retrieval of an existing design specification for the purpose of adapting it to design a new but similar product, [3,23,55,57]

21.4. CONSTRAINT-BASED MODELING

Variant design depends on a fixed script that has been defined manually. Although the script could be very complex, as in the case of embedding a design language into a general programming language, design as programming is less desirable than giving the designer visual tools and deriving from a visual design process a flexible and intuitive parametric design. With the arrival of the geometric constraint solving technology was at hand to do that. Finally, it was possible to prepare a rough sketch and, by adding specific dimensional and relational constraints, transform it into a precise drawing. Coupled with operations such as extrusions and cuts, it became possible to create designs intuitively and with ease. Furthermore, by valuating the dimensional constraints differently, variants could be obtained automatically by means of a general purpose *constraint solver*.

21.4.1. Constraints

A constraint specifies a relation on or between entities in a model that must be maintained. The following classes of constraints arise naturally:

- Geometric relationships such as concentricity, perpendicularity, etc., as well as metric dimensional constraints such as distance or angle.
- Equational constraints that express relations between dimensional parameters and/or technological variables such as torque.
- Semantic constraints that define validity conditions on a shape.
- Topological relations between entities in a model, such as incidence or connectivity.

To date, constraint systems of varying competence deal with some or all of these types of constraints. We distinguish between variational and parametric constraint problems.

A *parametric* geometric constraint problem is one in which a sequence of steps can be identified or derived that solves the problem. In each step, a single geometric element is placed in relation to elements already placed.

In contrast, a *variational* geometric constraint problem includes steps in which several geometric entities must be placed simultaneously in relation to each other.

For planar geometric constraint problems competent and efficient solvers are readily available. For spatial constraint problems the technology is not nearly as mature, likely owing to a greater intrinsic difficulty of spatial problems. This difference is manifest in design systems available today.

21.4.2. Modeling with constraints

When defining a model initially, sketches are prepared and annotated with constraints. *Sketching* can be done with a mouse or more specialized devices. *Constraining* the sketch often is through menu dialogs. The sketches are then converted into precise shapes, by solving the constraints. Finally, the solid shape is defined from the sketches, using operations such as cuts or protrusions generated from revolving or extruding cross sections.

Most systems allow interleaving sketching and constraining. Figure 21.1 left shows the sketch of a constraint problem input to the constraint solver. Here the arc should be tangent to the adjacent segments, and the two other segments should be perpendicular. Output of the constraint solver is shown on Figure 21.1 right.

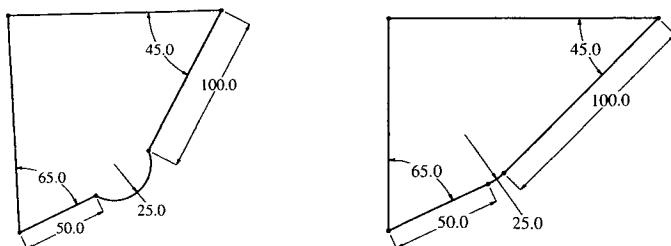


Figure 21.1. Sketch of a constraint problem and a solution generated by the solver.

When a sketch is *solved*, the underlying constraint solver expects in some cases a well-constrained problem, that is, one in which no additional constraints can be added without creating redundancies. Overconstrained sketches are usually rejected. In the case of underconstrained sketches the system will infer additional constraints that, when added, make the problem well-constrained.

When an already defined model is edited, the user changes some parameters or constraint values in the simplest case. The system constructs the new instance automatically solving for the changed values. More complex editing may change the parametric model itself, adding or deleting features, or changing the definition of some of them.

21.4.3. Solving geometric and equational constraints

Some constraint problems permit a sequential solution, in which the geometric elements are placed one-by-one, in accordance with the constraints. Such problems correspond to triangular, nonlinear equation systems. For planar cross section definitions, only a few modelers restrict to sequential problems. Most systems allow variational constraint problems in 2D, and therefore free the designer from the burden of having to understand whether the constraint schema is a constructive, sequential one. Note that sequential problems may also entail multiple solutions. For instance, assume that we are given two fixed circles and seek a common tangent of them. Then we would have to select one of up to four possible tangents.

A variational constraint problem is equivalent to a nonlinear system of equations. Moreover, a mathematically well-constrained problem will have more than one solution in general. There are general algorithms to convert a nontriangular system of nonlinear equations into a triangular system, [12]. Therefore, the distinction between parametric and variational constraint solving is artificial in theory. However, triangularization of systems of nonlinear equations is not tractable even for problems with a relatively small number of variables and equations.

Ideally, differentiating between the possible solutions and selecting the appropriate one would be accomplished by adding other, nongeometric constraints. Unfortunately, to-date no convincing approach to this problem has been discovered, and solvers rely on proprietary, sometimes rather complicated heuristics to select a solution that hopefully matches the intent of the designer. Simple “metaconstraints” can be entertained that might assist solution selection. For example, when designing a cross section, we might require that the bounding contour is not self-intersecting. Unfortunately, such simple rules cannot be efficiently implemented; [25].

Owing to the difficulty of variational constraint solving in three-space, spatial constraint problems are typically sequential. This imposes limitations on the designer that manifest themselves very clearly when designing mechanical assemblies.

Many approaches to solve geometric constraint problems have been reported in the literature. They can be categorized roughly as equational, constructive and degree of freedom analysis. We give a brief sketch of these techniques. For a thorough review see, e.g., reference [25].

Equational methods

An equational solver translates the geometric constraint problem into a system of algebraic equations which are then solved using a collection of techniques.

The numerical approach

A numerical solver applies iterative techniques to solve the equation system. Such solvers can be quite general, and many constraint solvers switch to numerical methods as an alternative to another method. However, most numerical methods have trouble handling overconstrained and underconstrained problems. Only overconstrained problems which consistently define an object may be solved using this techniques.

Early systems such as those reported in [5,73] used *relaxation* methods to solve the system of equations. Relaxation methods work by perturbing the values assigned to variables in such a way that the total error is minimized. The main problem is that convergence is slow.

A widely used numerical technique is the Newton-Raphson iterative method. Its main drawback is that the iteration requires a good initial value. If, as is usual, the initial values are taken from a rough sketch defined by the user, the sketch must almost satisfy all the constraints. Nonlinear systems have an exponential number of solutions and the Newton-Raphson iteration will find only the solution closest to the initial guess. Since the approach is unable to find alternative solutions, it is inappropriate when the initial sketch leads the solver to a solution which does not fit the users needs. Solvers in [30,31,53,56] are based on Newton iteration.

Hel-Or *et al.*, [28], report on a paradigm called *relaxed parametric design*, to guide the solver in the selection of a solution amongst a set of candidates, which satisfy all the constraints. The designer may provide soft constraints weighted by a user-defined certainty. Soft constraints are represented by a measurement and a tolerance and do not have to be satisfied exactly. A probabilistic constraint schema is used and an estimate of the model is computed using the Kalman filter technique developed in control theory.

Kin *et al.*, [46], solve geometric constraint problems using an extended Boltzmann ma-

chine, an artificial neural network. An energy function associated with the constraint network is defined to include terms of higher order than quadratic with respect to the binary states of the units that constitute the network. The extended Boltzman machine minimizes the polynomial energy function.

Recently developed methods in numerical continuation known as *homotopy* methods, are able to compute all solutions to polynomial systems [22,50,75]. The solution of a system of nonlinear equations by numerical continuation is motivated by the idea that small changes in the parameters of the system usually produce small changes in the solutions.

The symbolic approach

The symbolic approach translates the system of equations into another set of polynomials with the same roots. The resulting system is solved with symbolic algebraic methods, such as Buchberger's Gröbner Bases, [11], or the Wu-Ritt method, [16]. Both methods can solve general nonlinear systems of algebraic equations, but they require exponential running times. The transformed system is triangular, so the problem of simultaneously solving n polynomials with n variables is reduced to repeated univariate polynomial solving. The approach finds in principle all solutions. Solvers in references [10] and [47], use Buchberger's algorithm.

Propagation methods

The method generates an undirected graph whose nodes are the variables and constants in the system of equations and whose edges represent equations relating these variables and constants. The propagation method attempts to direct the graph edges so that every equation can be solved incrementally. The technique thus tries to discover a sequential strategy for solving the constraint system.

Various propagation techniques have been reported in the literature, [24,65,51], but none of them guarantees a solution when one exists, and most fail when a cyclic dependence is found. Propagation is sometimes used in conjunction with a numerical technique. For example, in [5,73], when the propagation of degrees of freedom fails, a relaxation method is used. For a review of these methods, see [52].

Constructive methods

Constraint solvers based on a constructive approach take advantage of the fact that many geometric constraint problems can be seen as engineering drawings which are usually solvable by ruler, compass and protractor. The two main approaches commonly classified as constructive are the rule-based and the graph-based approach.

Rule-based approach

In a rule-based approach, constraints are expressed by predicates, and geometric construction operations by functional expressions. These constructive solvers compute a symbolic solution of the constraint problem using a rewriting system to find a sequence of geometric operations that build the object which satisfies all the constraints. If the constraints consistently describe the position and orientation of the object, then the constraint problem can be solved.

The earliest rule-constructive solvers did not consider the problem of nonunique solutions, [1,72]. However, later approaches, [6,9,43,71,74,78], compute all possible solutions when constraint problems are well-defined.

Hoffmann and Joan-Arinyo, in [33], combine a rule-constructive solver with an equational solver based on graphs. When no more constructive rules apply, a bigraph is used to analyze the structure of the system of equations. Using matching theory techniques, a set of equations is isolated and solved in a general purpose equational solver. Joan-Arinyo and Soto generalized this approach in [44].

Graph-based approach

Graph-constructive solvers derive a sequence of construction steps using graph analysis techniques. DCM, a commercial constraint solver described in [58], uses this method: a graph is broken up into a set of subgraphs such that an algebraic solution for each class of the resulting subgraphs exists. Then, the subgraphs are positioned applying rigid body transformations to all geometric elements that belong to the subgraph.

Fudos and Hoffmann in [26] report on a graph-constructive approach to solve systems of geometric constraints capable of efficiently handling well-constrained, overconstrained, and underconstrained configurations.

Although this approach is faster and more methodical than the rule-constructive approach, the graph analysis algorithm needs to be modified when new types of constraints have to be considered.

21.4.4. Degrees of freedom analysis

In this approach, the notion of degrees of freedom is associated to primitive geometric objects and constraints. Any geometric object (point, line, circle, etc.) has a number of degrees of freedom in its embedding space. Constraints (coincidence, distance, angle, etc.) reduce the degrees of freedom of an object.

Kramer, [48], solves geometric constraint problems by symbolically reasoning about the geometric entities themselves using a technique called *degrees of freedom analysis*. In this approach, the configuration variables of a geometric object are defined as the minimum number of real-valued parameters required to specify the object in space unambiguously. The configuration variables parameterize an object's translational, rotational and dimensional degrees of freedom with one variable required for each degree of freedom. A constraint solver for three dimensional constraints is described in [48], in which constraints on rigid bodies are satisfied incrementally by a sequence of rigid-body motions. A plan of measurements and actions is devised to satisfy each constraint incrementally, thus monotonically decreasing the system's remaining degrees of freedom. This plan is used to solve, in a maximally decoupled form, the equations resulting from an algebraic representation of the problem. Kramer's solver is restricted to kinematic loops of length 4. For more complex interdependence his solver has to resort to numerical methods.

Using a graph-based technique, Hsu derives a plan of evaluation by examining and updating the degrees of freedom and dependencies between objects, [41]. First the method generates a connected subgraph and a dependency graph. Then the dependency graph is solved by a hybrid solver which generates the solution in the form of direct constructions and iterative constructions.

A flow-based method for decomposing the graph of a geometric constraint problem is described by Hoffmann *et al.* in [38]. The method fully generalizes the degree-of-freedom approach. The method iterates to obtain a decomposition of the system of equations underlying the constraint graph, into small subsystems.

21.5. FEATURE-BASED MODELING

Features have become an integral part of parametric modeling. Features provide a higher level vocabulary for specifying operations to create shapes by providing parametrized geometry, attributes and geometric constraints. Moreover, parameters, attributes and constraints can be encapsulated.

In a good design, features capture explicit engineering attributes and relationships for product definition and provide essential information for various design tasks and performance analyses. In manufacturing, features can be linked to manufacturing knowledge, thereby facilitating manufacturing and process planning. Features also provide a framework for organizing design and manufacturing information in a data repository for reuse in new product design, [68].

21.5.1. Features and the feature model

Features have been defined in a number of different ways in the literature. A good definition that captures the current trends in features development is due to Shah, [66], who defines a feature as a generic shape with which engineers associate certain properties or attributes and knowledge useful in reasoning about a product.

In order to be useful, a feature should embody at least three different concepts: Generic shape, behavior, and engineering significance, [68]. The generic shape is parametrically defined as a boundary representation, a CSG tree or another geometric representation, including procedural representations.

Behavior and engineering significance are defined by means of attributes and domain-specific rules. Attributes can be classified into several groups. Geometric attributes refer to the feature's shape and examples are dimension attributes, default and feasible values for parameters, tolerances, location parameters and so on. Technological attributes give information useful to downstream applications, such as material properties, heat treatments, tool and fixture information, etc. Some attributes can take the form of rules to define the behavior of the feature. The rules state what conditions should or must be imposed on a feature within a given process in order to perform a particular activity. Attachment validation and symbolic or skeletal representation derivations are examples of such rules.

A feature model is a data structure that represents a part or an assembly in terms of its constituent features. Feature models are created by organizing the constituent features in a suitable structure that expresses the required relationships between the various features.

There is a continuing debate on what a precise definition of feature should be. In part, the debate is fueled by conflicting part and assembly conceptualizations arising from different categories of design, analysis, and manufacture. For example, the burner casing of a jet engine may have a set of features relevant to thermal analysis, yet a different set of features may be relevant to structural analysis. A third set of features may be important to the casting process by which the casing is manufactured, and a fourth set

of features may be important to analyzing tolerances in the context of the assembly with other engine parts. These different categories can be considered views, and each view of the product will focus on its particular set of features.

This divergence of feature sets, on the same product, would be less onerous were it not that the design process of the geometric shape forces the designer to distinguish a particular set of features for the purpose of geometry creation. This set of design features often is not useful to the manufacturing engineer or the performance analyst. Owing to limited technology, switching between different feature sets during product design and manufacture is difficult or not supported by many CAD systems, leading to privileged views and continuing interest in developing techniques to switch effectively between different views without losing the flexibility of parametric design.

21.5.2. A brief feature taxonomy

We distinguish between geometric features and nongeometric features. Geometric features are closely related to the geometry of a model and can be further differentiated into

- Form features: portions of nominal geometry defining a feature's shape.
- Tolerance features: Deviation from nominal definitions of shape, size or location.
- Assembly features: Grouping of various features to define assembly relations such as mating conditions, position and orientation, kinematic relations, etc.

Viewed from an application perspective, geometric features can also be classified into design features, manufacturing features, process planning features, etc.

Nongeometric features are generally related to technological information. Examples of this type of features are:

- Functional features: Sets of features related to a specific function like design intent, parameters related to function and performance, etc.
- Material features: Material composition, treatment, surface finish, etc.

21.5.3. Feature model construction

Three basic techniques for constructing parametric feature models have been identified, [66,68]: Interactive feature definition, automatic feature recognition and design by features.

Interactive feature definition

In the interactive feature definition technique the user interacts with a model that has already been defined, possibly using another design methodology. Interactively, the user selects on this displayed model entities to be grouped into a feature. A feature so defined can then be annotated with attributes such as surface finish and tolerances. In some cases, the feature can be parameterized by defining parameters and constraints on the entities.

Groupings and annotations are easily implemented. Moreover, the entire model need not be featurized, only those features need to be defined that are of particular use in the application the user has in mind. Feature validation is usually the task of the user.

If many features have to be defined, this process is error-prone and tedious. Moreover, the persistence of annotation is usually not guaranteed, although technology exists to make persistent annotations of parametric models. The definitional task can be assisted by feature recognition coupled with a feature library that contains generic feature definitions and can propose completions of a partially defined feature.

A more complicated issue arises when the original model does not have any parametric information and the user adds parameters to the features defined on the model. Re-evaluation of the model with changed parameter values is needed in this case, and there are commercial modelers that can accomplish this task within certain limits. The problem increases in difficulty if the original model is parametric, and we wish to preserve the original parameterization structure in addition to editing the model from user-defined features and their parameters. This problem is technically related to reconciling different views when editing, and requiring that different views can edit from within their perspective.

Automatic feature recognition

In automatic feature recognition, a previously defined geometric model is processed algorithmically to detect features defined in terms of rules or subgraphs or other kind of generic feature knowledge. This approach has received considerable attention in the literature. However, several research challenges still need to be addressed and solved in order to ground the approach in a robust and solid framework. See [2] for a review of automatic feature recognition systems.

A major difficulty common to all known approaches to automated feature recognition is the recognition of intersecting features. When several features intersect, their topology can change dramatically. Since most of the proposed feature recognition techniques seek to identify among the model's edges and faces groups that exhibit a specific topological and geometric character, the fragmentation entailed by intersecting features can foil the recognition algorithm. Moreover, it can lead to interpretation ambiguities that must be resolved by adopting certain heuristics. Work has been reported that attempts to recognize features not only individually but also feature interrelations like containment and intersection; see, e.g., [63]. Unfortunately, the computational complexity explodes even for very simple objects.

Most of the existing feature recognizers work in batch mode. They accept as input a completed design and produce as output a feature model. For the reasons explained before, the feature model built represents one of several possible interpretations. Even a small change in the initial model can force feature recognizers to discard the previous work and start an expensive geometric reasoning process from scratch. Since batch operation is undesirable in interactive environments, some efforts have been devoted to incremental feature recognition, [27].

Another drawback arises from the fact that automated feature recognition techniques process final functional forms. Therefore, it is not adequate for certain types of downstream applications. For example, in process planning in the automotive industry, the intermediate geometry must also be identified, [29]. Since intermediate shapes, sometimes also called *in-process* shapes, cannot reliably be reconstructed from the final net shape, important information has been lost that reduces the applicability of automated feature recognition.

Design by features

Design by features is the most widely used technique in feature-based modeling. Here the model is built directly by the user by instantiating generic feature definitions, which are used as templates, and locating them in space or attaching them on existing features. The features can be instantiated from a library of either standard generic features or from application-oriented, user-defined features, [64].

Feature-based systems usually provide standard generic feature libraries as collections of predefined features such as slots, pockets, and holes, and operations for defining sketched features where geometry is created by sweeping a planar cross section or lofting between two or more planar cross sections. Standard operations provided by systems allow the user to create complex shape designs that could not be built using standard features only.

A plausible design process of a pocket with a bridge across the bottom is illustrated in Figure 21.2. The designer first creates a pocket of maximum depth, and then adds the rib as a protrusion with fillets at the edges.

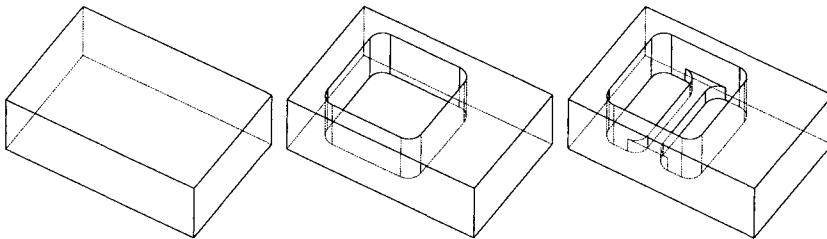


Figure 21.2. Design sequence: make a cut, then add a protrusion.

The design by features technique provides a set of operations to edit the model. Broadly speaking, these operations are: Inserting or deleting an entire feature, changing feature attributes, modifying dimension values that define the feature or place it in the model, changing the set of constraints associated with the feature, and changing the feature shape definition. Procedural steps common to these editing operations are given in [13].

21.5.4. Feature representation

The shape of a feature may be expressed in terms of construction steps that produce the geometry corresponding to the feature or in terms of an enumeration of geometric and topological entities and relations along with dimension parameters, [35]. The first approach is a procedural approach while the second is a declarative approach.

Procedural representation

In the procedural approach, generic features are predefined in terms of a collection of procedures which may include methods for managing a feature as a whole, like instantiating, copying and deleting a feature, and methods for specific operations on a given feature like generating the geometry, deriving values for parameters, and validating feature opera-

tions. The procedures may be encoded in either a general or special-purpose programming language.

In procedural feature representation, feature definitions are explicitly expressed in terms of a computation. So, a feature is always instantiated from a given function with a given set of parameters whose values the user sets. If the given parameter values are changed, the entire procedure must be run again to compute the new instance.

Declarative representation

The declarative approach, features and their properties are first described in a declaratively; i.e., the definition declares what the feature is rather than how it is built. Then, a general algorithm constructs the actual detailed feature model. One of the main tools of the declarative approach is the use of constraints to define the particulars of features.

In [37], the authors proposed the Erep declarative framework that achieves a clean separation between definition and construction. In this framework, geometry and properties are represented in a neutral form while the actual construction is performed by algorithms committed to a clear, underlying semantics. The framework naturally provides tools for representing constraints and attributes of features.

In declarative feature representation, constraints play a central role because they provide a natural way to describe spatial relations between geometric entities within a feature and between features. Furthermore, constraints provide a mechanism to define relationships between geometric and technological parameters. Therefore, all the constraint solving machinery can be applied effectively.

21.5.5. Features and constraints

Increasingly, solid modeling systems use both features and constraints in the design interface. Typically, feature-based design systems deploy a design paradigm in which the designer may use a set of predefined features and operations for defining *sketched features*. The geometry of sketched features is created by sweeping a planar cross section or lofting between two or more planar cross sections.

Cross sections are defined as sketches with declarative constraints. A variational constraint solver instantiates the cross section based on the sketch and the constraints, and places it with respect to the geometry constructed so far from prior features. Parameters and constraints then define the sweep extent.

21.6. TRENDS

Trends in parametric solid modeling are fostered mainly by two different requirements: integration with product data management and downstream applications, and support for concurrent distributed design environments.

To fulfill these requirements, solid modeling should provide an efficient and direct communication between engineering processes which, in turn, requires advanced modeling tools and methods to provide users with facilities to capture geometry sufficiently enriched with engineering semantics. These requirements affect parametrics, and therefore features and constraints, in a number of ways.

21.6.1. Feature libraries

Devising a universal set of features would improve the interoperability between different applications in an integrated environment. But seeking to devise such a set of features would lead to an unmanageable number of features. For this reason, research has begun to investigate generic mechanisms to give the user the option of building custom feature libraries that might satisfy specific application needs. This approach has been advocated in the following work.

Shah *et al.* reported on the ASU shell in [69], a testbed for rapid prototyping of feature based applications. The library of generic features is organized in the form of a list of properties. Each feature has a feature type identifier, a name, a list of generic, compatible features, and a solid representation. A CSG tree provide the solid representation for form features. Recent developments of the test-bed are reported in [67].

Laakko and Mäntylä, [49], describe an extension of the programming language Common Lisp to represent features procedurally. The feature models are organized as a structure of Lisp frames. Such frames model two different types of features: *features classes* which are templates that store generic information, and *feature instances* that store specific information belonging to individual features. Feature classes are organized by a taxonomy and use the inheritance mechanism of Common Lisp. A feature model is a list of feature instances.

De Kraker *et al.* in [19] and Dohmen *et al.* in [21] report on the specification of a feature language developed at Delft University of Technology. Features are specified using predefined types in the object-oriented, imperative programming language LOOKS. Therefore, the feature library is a library of LOOKS procedures and defining a new feature means to write new code for it.

Hoffmann and Joan-Arinyo in [37] proposed the Erep framework for expressing form features and constraints. The representation of a part design is a generative feature description. It is textual and neutral, in the sense that it is not committed to a particular core solid modeling system. In [35], the Erep was extended with a procedural mechanism for generating and deploying user-defined features through standard graphic operations provided by the underlying modeling system.

21.6.2. Multiple views

In an integrated, concurrent and distributed design environment, the data in the product model is contributed by different applications. Each application has its own *view* of the data. For example, from a machining point of view, the feature structure shown in Figure 21.3 is one of many possible interpretation for the design view in Figure 21.2. Therefore, a persistent association between data contributed by each application must be establish and maintained. Creating and maintaining such a persistent association is a key problem.

It is natural to argue that in concurrent engineering, a modification required by a specific application should be made in the view of that application. Moreover, all modifications introduced in one view should be propagated automatically to all other views. Some work based on this assumption has approached the problem in a setting far too general. In the absence of specifics, such work has not proposed credible mechanisms to address the view consistency problem. The work by Bronsvort *et al.* is a notable proposal in that

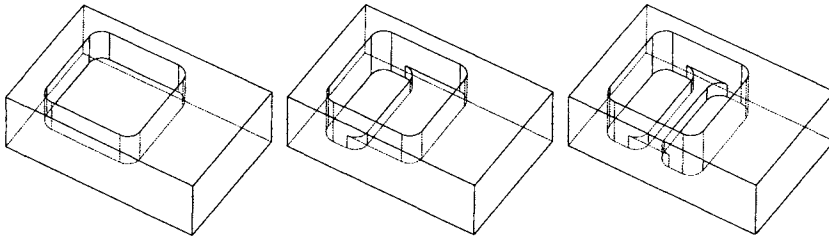


Figure 21.3. Machining view sequence: make a shallow pocket, then deepen the pockets to the left and right of the bridge.

respect, [7,18,19,21]. The work addresses formally the problem of different form feature views editing a common net shape. Briefly, the net shape is modeled by a cell complex where the cells are subdivided as necessary such that every feature of an application view is composed of entire cells. That property permits to edit shape mechanically from any feature view and achieves consistency across all views.

A different approach is developed by Hoffmann and Joan-Arinyo in [34,36]. The approach is based on the concept of the *master model*, an object-oriented repository that provides essential mechanisms for maintaining the integrity and consistency of the deposited information structures. The clients of the master model are the modeling system and domain-specific applications. An analysis of the needs of different views allows a simpler solution of consistent edits that uses persistent information associations.

21.6.3. Semantic features

The concept of semantic feature was developed in response to the need of capturing engineering information that connect form with functional intent. Such features also support integration with downstream applications. Semantic features provide tools for using features consistently.

Procedural semantics for attaching features to a model in generative constrained-based modeling systems have been defined by Chen and Hoffmann in [15]. The work considers generated features that are based on a planar profile swept into a three dimensional shape. This work was extended in [14] where a set of techniques needed for editing generative feature-based models is discussed, including persistent naming techniques. Persistent naming is needed to re-evaluate edited features.

Effectively maintaining the validity of a feature, attached to a model, entails developing mechanisms to detect invalid situations, mechanisms to properly report improper feature use, and mechanisms to provide the user with good choices to recover validity. In the work reported in [4,17,20] and [21], the validity of a feature is specified as a set of geometric, topological and functional constraints. Whenever a modeling operation is completed, a validity check is performed. If a violation of some validity criterion is detected, the operation branches into a reaction loop where a validity recovery process is triggered. A valid state is achieved again, either by reestablishing the previous valid state or through

a dialog with the user.

21.6.4. Persistent naming

As mentioned before, applications of solid modeling in manufacturing and other application arenas seek to enrich the semantic content of a shape model. Geometric constraints and parameters can be considered semantic information pertaining to shape, sometimes in an implicit form. Other information, such as annotations or feature definitions relevant to different views, is needed as well. When a parametric model is edited, it is crucial to preserve and re-attach semantic information. The technology to do this has been called *persistent naming*.

The problem of persistent naming arises as follows. Consider a particular shape instance on which we annotate a selected face with some material property information. This is done easily for the instance data structure, usually a boundary representation. However, the parametric model itself need not have any faces, so it is not clear how to record the information such that it persists under a change of parameters or constraints. More than that, different model instances of the same parametric model may have zero, one, or several faces that might correspond to the original annotated face. This problem has been considered in [14,32,70] and [59], and several approaches have been proposed in the literature.

Ultimately, a solution to the persistent naming problem is a semantic interpretation of the instantiation of a parametric model. There remain basic questions on what constitutes a good semantics that may never be resolved by the community. Therefore, the persistent naming solutions proposed in the literature can be considered to be specific proposals for such a semantics. Because of this larger context, work on persistent naming should be considered evolutionary.

We postulate that a successful semantics for parametric shape design has to allow construction algorithms that exhibit two key properties:

1. The parametric instantiation algorithm has to be *continuous*.
2. The parametric instantiation algorithm has to be *persistent*.

Continuity requires that a geometric configuration, derived by instantiating for a set of parameter values (p_1, \dots, p_k) should change by a small amount when altering the values of the parameters by a small amount. Persistence means that after changing the parameter values and reinstantiating the configuration, a return to the original parameter values should result in the original configuration being recovered. Clearly, those are minimal requirements any user of would expect from parametric design.

Geometric software is often hard-pressed to make good on continuity which is easily violated when the configuration passes a degenerate configuration. Such degenerate configurations are not easily recognized because parameter changes are typically discrete. Cinderella [62] is an example of geometry software whose design pays particular attention to achieving continuity. Cinderella allows sequential constructions of geometric configurations of points, lines and conic sections. The user can then drag elements and the system updates the configuration in a continuous way. It accomplishes continuity by tracking solutions through complex configurations. Since the paths for changing the configuration

avoid singularities by randomization, but do not remember the chosen paths, Cinderella does not exhibit persistence.

Many constraint solvers exhibit persistence without continuity, including the solvers we have designed. Here, persistence is accomplished by finding coordinate-independent characterizations of constraint solutions so that a solution can be designated by a short certificate that is derived from the initial or the current configuration.

21.7. OPEN PROBLEMS

Parametric design has achieved great accomplishments. However, to take full advantage of its potential, a number of open problems need to be solved. We outline some of the key problems.

21.7.1. Constraint solving

Two-dimensional constraint solving has been studied extensively, and although there is no single best technique, successful approaches have been developed that are both efficient and sufficiently general. Geometric constraint solving in three dimensions has been much less explored, except for the purely numerical techniques, whose drawbacks have been discussed in Section 21.4.

The problem in three dimensions grows in difficulty, not only because the number of variables is larger, but also due to the fact that some of the results valid in two dimensions do not extend to three dimensions. Some recent developments in placing points, lines and spheres with fixed radius in three dimensional space with reasonable computational cost and reliability are reported by Hoffmann and Vermeer, [39,40] and by Durand [22]. Currently, active research seeks better techniques. A general problem decomposition algorithm has been given in [38].

21.7.2. Features

A key obstacle is the lack of techniques to support multiple views effectively. Mapping algorithms are needed that can connect different feature schemata with each other and allow designers to edit in different views. This would greatly expand the flexibility of parametric design, and permit designers to increase the semantic content of parametric models.

21.7.3. Semantics of parametric design

Advantages of parametrics lie in the ability of the system to allow easy subsequent edits of the design by changing input parameters that were initially specified when the design was created. To date, all such changes are described in terms of the procedures that actually perform the change. For simple objects, those procedures provide highly productive tools.

When the ranges of parameter values widen or the complexity of the designed objects increases, today's algorithms no longer guarantee that the parametric models are valid and unambiguous, and the results of modeling operations are not always predictable.

As an example, consider the solid shown in Figure 21.4 left. It was constructed as follows: First, a rectangle was drawn and extruded into a block. On the front face of the block, a circle was drawn as a profile of a slot across the top of the block. Then the left edge of the slot was visually identified for rounding. The result is shown in Figure 21.4

left. This design is now edited by altering the position of the circular slot profile. The correct result is shown in Figure 21.4 middle, but some systems may construct instead the shape in Figure 21.4 right, clearly an error.

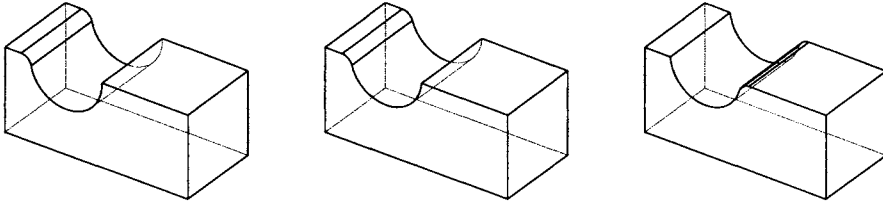


Figure 21.4. A block with a slot and a round on an edge.

The problem is how to describe in the generic design the edge to be rounded. An abstract definition of shape change under constraint changes would be needed, as a step towards a rigorous semantic definition of generic design and constraint-based editing.

21.7.4. Assembly-centric design

Shape design has traditionally been conceptualized as part-centric design, and CAD systems are very good at designing detailed shapes of individual parts. However, in many applications the interactions of the parts create the primary view of a mechanism, and engineering design often begins from this vantage point. In part-centric design, the focus of the design activity is on the geometry creation, typically of the net shape, and annotations of the shape with features relevant to various views.

To create an assembly-centric parametric design process, it is minimally required to interrelate the parameters and constraints of the various parts of the mechanism to each other. As with views, this raises the question of updating all parts of an assembly when changing a parameter of a single component part. Here, design methodology might guide how to conceptualize key parameters and their functional relevance. Assembly-centric design would find immediate application in areas such as tolerancing and process planning, and in the functional design of abstract mechanisms.

Assembly-centric design would span two levels. On the basic level, parts are interrelated to each other in the assembly, and design parameters would be correlated between parts or derived from assembly-level parameters in an algorithmic way.

A more advanced conceptualization of assembly design would give the designer the capability to differentiate subassemblies and re-use them as parametric elements of the overall design. For instance, we might consider the sensor assembly of an automobile cooling system such a subassembly. Depending on dashboard configurations, this subassembly could be connected to different electric leads, in one combination only lighting up a light indicating operating range temperature and a warning light if the temperature is too high. In another combination the leads from the subassembly could drive a temperature gauge in the dashboard; e.g., [42,76].

Finally, specific assembly-level parameters could result in the instantiation of differently configured subassemblies that would be responsive to different functional characteristics and ranges.

REFERENCES

1. B. Aldelfeld. Variation of geometric based on a geometric-reasoning method. *Computer-Aided Design*, 20(3):117–126, April 1988.
2. V. Allada and S. Anand. Feature-based modelling approaches for integrated manufacturing: state-of-the-art survey and future research directions. *International Journal of Computer Integrated Manufacturing*, 8(6):411–440, 1995.
3. P. Andrews, T. Shahin, and S. Sivaloganathan. Design reuse in CAD environment - Four case studies. *Computers & Industrial Engineering*, 37(1):105–109, 1999.
4. R. Bidarra and J. Bronsvoort. Validity maintenance of semantic feature models. In W. Bronsvoort and D. Anderson, editors, *Fifth Symposium on Solid Modeling and Applications*, pages 85–96, ACM Press, Ann Arbor, MI, June 9–11 1999.
5. A. Borning. The programming language aspects of ThingLab, a constrained oriented simulation laboratory. *ACM Trans. on Prog. Lang. and Systems*, 3(4):353–387, October 1981.
6. W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. Geometric constraint solver. *Computer-Aided Design*, 27(6):487–501, June 1995.
7. W. Bronsvoort and F. Jansen. Multi-view feature modelling for design and assembly. In J. Shah, M. Mäntylä, and D. Nau, editors, *Advances in Feature Based Manufacturing*, Manufacturing Research and Technology, 20. Elsevier Science B.V., Ch. 14, pages 315–330, 1994.
8. C. Brown. PADL-2: A technical summary. *IEEE Computer Graphics and Applications*, 2(2):69–84, March 1982.
9. B. Brüderlin. Using geometric rewrite rules for solving geometric problems symbolically. In *Theoretical Computer Science 116*, Elsevier Science Publishers B.V., pages 291–303, 1993.
10. S. Buchanan and A. de Penington. Constraint definition system: a computer-algebra based approach to solving geometric-constraint problems. *Computer-Aided Design*, 25(12):741–750, December 1993.
11. B. Buchberger. *Multidimensional Systems Theory*. D. Reidel Publishing Theory, Ch. Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, pages 184–232, 1985.
12. B. Buchberger, G. Collins, and B. Kutzler. Algebraic methods for geometric reasoning. *Annual Review of Computer Science* 3:85–120, 1988.
13. X. Chen. *Representation, Evaluation and Editing of Feature-Based and Constraint-Based Design*. PhD thesis, Department of Computer Sciences, Purdue University, May 1995.
14. X. Chen and C. Hoffmann. On editability of feature-based design. *Computer-Aided Design*, 27(12):905–914, 1995.
15. X. Chen and C. Hoffmann. Towards feature attachment. *Computer-Aided Design*, 27(9):695–702, 1995.

16. S.-C. Chou. An introduction to Wu's method for mechanical theorem proving in geometry. *Journal of Automated Reasoning* , 4:237–267, 1988.
17. J. de Kraker, M. Dohmen, and W. Bronsvoort. Maintaining multiple views in feature modeling. In C. Hoffmann and W. Bronsvoort, editors, *Fourth Symposium on Solid Modeling and Applications* , pages 123–130, ACM Press, Atlanta, GA, May 14–16 1997.
18. K. de Kraker, M. Dohem, and W. Bronsvoort. Multiple-way feature conversion. Opening a view. In M. Pratt, R. Siriram, and M. Wozny, editors, *Product Modeling for Computer Integrated Design and Manufacture*, pages 203–212, Chapman and Hall, London, UK, 1997.
19. K. de Kraker, M. Dohmen, and W. Bronsvoort. Feature validation and conversion. In D. Roller and P. Brunet, editors, *CAD Systems Development*. Springer-Verlag, Heidelberg, 1997.
20. M. Dohmen. *Constrained-Based Feature Validation*. PhD thesis, Delf University of Technology, 1997.
21. M. Dohmen, K. de Kraker, and W. Bronsvoort. Feature validation in a multiple-view modeling system. In *16th ASME International Computers in Engineering Conference*, ASME, Irvin, NY, USA, 19–22 August, 1996.
22. C. Durand. *Symbolic and Numerical Techniques for Constraint Solving*. PhD thesis, Computer Science, Purdue University, December 1998.
23. J. Fowler. Variant design for mechanical artifacts: A state-of-the-art survey. *Engineering with Computers* ,12(1):1–15, 1996.
24. B. Freeman-Benson, J. Maloney, and A. Borning. An incremental constraint solver. *Communications of the ACM* , 33(1):54–63, 1990.
25. I. Fudos. *Constraint Solving for Computer Aided Design*. PhD thesis, Purdue University, Department of Computer Sciences, 1995.
26. I. Fudos and C. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics* , 16(2):179–216, April 1997.
27. J. Han and A. Requicha. Incremental recognition of machining features. In *Proceedings of the ASME Computers in Engineering Conference*, pages 587–598, Minneapolis, MN, 1994.
28. Y. Hel-Or, A. Rapoport, and M. Werman. Relaxed parametric design with probabilistic constraints. In J. Rossignac, J. Turner, and G. Allen, editors, *Second Symposium on Solid Modeling and Applications*, pages 261–270, ACM Press, Montreal, Canada, May 19–21 1993.
29. V. Hetem. Communication: computer aided engineering in the next millennium. *Computer-Aided Design* , 32(5-6):389–394, May–June 2000.
30. A. Heydon and G. Nelson. The Juno-2 constraint-based drawing editor. Research Report 131a, Digital Systems Research Center, December 1994.
31. R. Hillyard and I. Braid. Characterizing non-ideal shapes in terms of dimensions and tolerances. In *ACM Computer Graphics*, pages 234–238, 1978.
32. C. Hoffmann. On the semantics of generative geometry representations. In *Nineteenth ASME Design Automation Conference*, pages 411–420, ASME Press, New York, NY, 1993.
33. C. Hoffmann and R. Joan-Arinyo. Symbolic constraints in constructive geometric

- constraint solving. *Journal of Symbolic Computation* , 23:287–300, 1997.
34. C. Hoffmann and R. Joan-Arinyo. CAD and the product master model. *Computer-Aided Design* , 30(11):905–918, September 1998.
 35. C. Hoffmann and R. Joan-Arinyo. On user-defined features. *Computer-Aided Design* , 30(5):321–332, April 1998.
 36. C. Hoffmann and R. Joan-Arinyo. Distributed maintenance of multiple product views. *Computer-Aided Design* , 32(7):421–431, June 2000.
 37. C. Hoffmann and R. Juan. Erep – An editable high-level representation for geometric design and analysis. In P. Wilson, M. Wozny, and M. Pratt, editors, *Geometric Modeling for Product Realization*, pages 129–164, North Holland, 1993.
 38. C. Hoffmann, A. Lomonosov and M. Sitharam. Geometric constraint decomposition. In B. Brüderlin and D. Roller, editors, *Geometric Constraint Solving and Applications*, pages 171–195. Springer, Berlin, 1998.
 39. C. Hoffmann and P. Vermeer. Geometric constraint solving in \mathbb{R}^2 and \mathbb{R}^3 . In D.-Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, pages 266–298. World Scientific Publishing, 1995.
 40. C. Hoffmann and P. Vermeer. A spatial constraint problem. In J. Merlet and B. Ravani, editors, *Computational Kinematics'95*, pages 83–92. Kluwer Academic Publ., 1995.
 41. C.-Y. Hsu. *Graph-Based Approach for Solving Geometric Constraint Problems*. PhD thesis, Department of Computer Science, The University of Utah, June 1996.
 42. ISATP'99. *Proceedings of the 1999 Third IEEE International Symposium on Assembly and Task Planning*, Porto, Portugal, 1999.
 43. R. Joan-Arinyo and A. Soto. A ruler-and-compass geometric constraint solver. In M. Pratt, R. Sriram, and M. Wozny, editors, *Product Modeling for Computer Integrated Design and Manufacture*, pages 384–393. Chapman and Hall, London, 1997.
 44. R. Joan-Arinyo and A. Soto-Riera. Combining constructive and equational geometric constraint solving techniques. *ACM Transactions on Graphics* , 18(1):35–55, January 1999.
 45. R. Juan, L. Solano, E. Torres, and J. Vega. Sistema dmi modelado geométrico de tuberías. In *III Congreso Español de Informatica Grafica*, Granada, Spain, 1993.
 46. N. Kin, Y. Takai, and T. Kunii. A connectionist approach to geometrical constraint-solving. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics*, Springer-Verlag, 1993.
 47. K. Kondo. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer-Aided Design* , 24(3):141–147, March 1992.
 48. G. Kramer. *Solving Geometric Constraints Systems*. MIT Press, 1992.
 49. T. Laakko and M. Mäntylä. Incremental constraint modelling in a feature modelling system. *Computer Graphics Forum* , 15(3):367–376, 1996.
 50. H. Lamure and D. Michelucci. Solving geometric constraints by homotopy. In C. Hoffmann and J. Rossignac, editors, *Third Symposium on Solid Modeling and Applications*, pages 263–269, ACM Press, Salt Lake City, Utah USA, May 17–19, 1995.
 51. R. Latham and A. Middleditch. Connectivity analysis: a tool for processing geometric constraints. *Computer-Aided Design* , 28(11):917–928, November 1996.
 52. W. Leler. *Constraint Programming Languages: Their Specification and Generation*.

- Addison-Wesley, 1988.
53. R. Light and D. Gossard. Modification of geometric models through variational geometry. *Computer-Aided Design* , 14:209–214, July 1982.
 54. C. McMahon, K. Lehane, J. S. Williams, and G. Webber. Observations on the application and development of parametric-programming techniques. *Computer-Aided Design* , 24(10):541–546, October 1992.
 55. M. Mitchell, T. Jiao, and J. Jiao. A variant approach to product definition by recognizing functional requirements patterns. *Computers & Industrial Engineering* , 33(3-4):629–633, 1997.
 56. G. Nelson. Juno, a constraint-based graphics system. *SIGGRAPH* , 235–243, San Francisco, July 22–26 1985.
 57. K. Olsen and P. Sætre. Describing products as executable programs: Variant specification in a customer-oriented environment. *International Journal of Production Economics* 56-57, 20:495–502, September 1998.
 58. J. Owen. Algebraic solution for geometry from dimensional constraints. In R. Rossignac and J. Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 397–407, ACM Press, Austin, TX, June 5–7, 1991.
 59. S. Raghobhama and V. Shapiro. Consistent updates in dual representation systems. *Computer-Aided Design* , 32(8/9):463–477, July/August 2000.
 60. A. Rapoport. Geometric modeling: a new fundamental framework and its practical implications. In C. Hoffmann and J. Rossignac, editors, *Third Symposium on Solid Modeling and Applications*, pages 31–41, ACM Press, Salt Lake City, Utah, May 17–19 1995.
 61. A. Requicha and H. Voelcker. Solid modeling. A historical summary and contemporary assessment. *IEEE Computer Graphics and Applications* , 2(2):9–24, March 1982.
 62. J. Richter-Gebert and U. H. Kortenkamp. *The Interactive Geometry Software Cinderella*. Springer, New York, 1999.
 63. H. Sakurai and C.-W. Chin. Definition and recognition of volume features for process planning. In J. Shah, M. Mäntylä, and D. Nau, editors, *Advances in Feature Based Manufacturing*, Manufacturing Research and Technology, 20. Elsevier Science B.V., Ch. 4, pages 65–80, 1994.
 64. O. Salomons, F. van Houten, and H. Kals. Review of research in feature based design. *Journal of Manufacturing Systems* , 12(2):113–132, 1993.
 65. M. Sannella. The SkyBlue constraint solver. Technical Report 92-07-02, University of Washington, Dept of Computer Science and Engineering, 1993.
 66. J. Shah. Conceptual development of form features and feature models. *Research in Engineering Design* , 2:93–108, 1991.
 67. J. Shah, H. Dedhia, V. Pherwani, and S. Solkhan. Dynamic interfacing of applications to geometric modeling services via modeler neutral protocol. *Computer-Aided Design* , 29(12):811–824, December 1997.
 68. J. Shah and M. Mäntylä. *Parametric and Feature-Based CAD/CAM*. John Wiley and Sons, Inc, New York, 1995.
 69. J. Shah and M. Rogers. A testbed for rapid prototyping of feature based applications. In J. Shah, M. Mäntylä, and D. Nau, editors, *Advances in Feature Based Manufac-*

- turing, Manufacturing Research and Technology, 20. Elsevier Science B.V., Ch. 18, pages 423–453, 1994.
70. V. Shapiro and D. Vossler. What is a parametric family of solids? In C. Hoffmann and J. Rossignac, editors, *Third Symposium on Solid Modeling and Applications*, pages 43–54, ACM Press, Salt Lake City, UT, May 1995.
 71. W. Sohr. *Interaction with Constraints in Three-Dimensional Modeling*. Master's thesis, Dept of Computer Science, The University of Utah, March 1991.
 72. G. Sunde. A CAD system with declarative specification of shape. *Eurographics Workshop on Intelligent CAD Systems*, 90–105, April 1987.
 73. I. Sutherland. Sketchpad, a man-machine graphical communication system. In *Proc. of the Spring Joint Comp. Conference*, pages 329–345, IFIPS, 1963.
 74. A. Verroust, F. Schonek, and D. Roller. Rule-oriented method for parameterized computer-aided design. *Computer-Aided Design*, 24(10):531–540, October 1992.
 75. C. Wampler, A. Morgan, and A. Sommese. Numerical continuation methods for solving systems arising in kinematics. *Journal of Mechanical Design*, 112:69–68, 1986.
 76. D. Whitney. The potential for assembly modeling in product development and manufacturing. Tech. Rep., MIT, 1996.
 77. J. Wilkes and R. Leonard. Variant design as a method of automating the design process. *Computer-Aided Engineering Journal*, 97–102, June 1988.
 78. Y. Yamaguchi and F. Kimura. A constraint modeling system for variational geometry. In J. T. M.J. Wozny, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 221–233, Elsevier North Holland, 1990.

Chapter 22

Sculptured Surface NC Machining

Byoung K. Choi, Bo H. Kim, and Robert B. Jerard

Many products are designed with sculptured surfaces to enhance their aesthetic appeal; it is an important factor in customer satisfaction, especially in the automotive and consumer-electronics industries. Other products have sculptured shapes to meet functional requirements, such as:

- Aerodynamic: airfoils (jet engines), impellers (compressors), marine propellers, etc.
- Optical: lamp reflectors (automobiles), shadow masks (TV monitors), radar dishes, etc.
- Medical: parts for anatomical reproduction.
- Structural: structural frames (aircraft), sporting goods, etc.
- Manufacturing: parting surfaces (molding dies), die faces (stamping dies), etc.

While these aesthetic and functional surfaces are created using CAGD (computer-aided geometric design) techniques, it is the role of SSM (sculptured surface machining) to realize them in physical form. As an ever-increasing variety of products are being designed with sculptured surfaces, efficient machining of these surfaces is becoming increasingly important in many areas of manufacturing including the automobile, consumer-electronics, aerospace, ship-building, die-making, sporting-equipment and toy-making industries. For many companies, sculptured surface machining has become a strategic technology.

22.1. INTRODUCTION

22.1.1. Overview of the sculptured surface machining process

Examples of various sculptured surface machining (SSM) processes are provided in a number of articles: airfoil machining in Mason [15], impeller machining in Takeuchi et al. [22], marine propeller machining in Choi et al. [6], die and mold machining in Altan et al. [1] and Fallbohmer et al. [12], medical parts machining in Duncan and Mair [10], etc. However, the lack of a unified framework for describing SSM processes makes it difficult to compare one approach to another.

The first step in providing such a framework is a set of basic definitions as follows:

- **UMO** (unit machining operation): a basic unit of an SSM operation carried out by a single cutting tool, which has a distinguishable pattern with a well-defined machining boundary.
- **Machining stage**: a group of UMOs employed to achieve a certain operational goal, such as roughing, finishing, clean-up, etc.
- **SSM process**: a set of machining stages employed in making a sculptured part.

It should be noted that the actual definitions of UMOs and machining stages may vary between different applications, but they provide a framework for representing an SSM process.

The purpose of an SSM process is to produce a sculptured part by applying a series of metal-removal processes to a workpiece. The term **workpiece** is used to denote the current state of the object at any given stage of the SSM process.

Most engineering disciplines are concerned with modeling. A modeling process involves abstraction and generalization, and the resulting model should serve as a general framework as well as a useful tool for investigating or solving the problem at hand. SSM process models are classified into sequential models and hierarchical models.

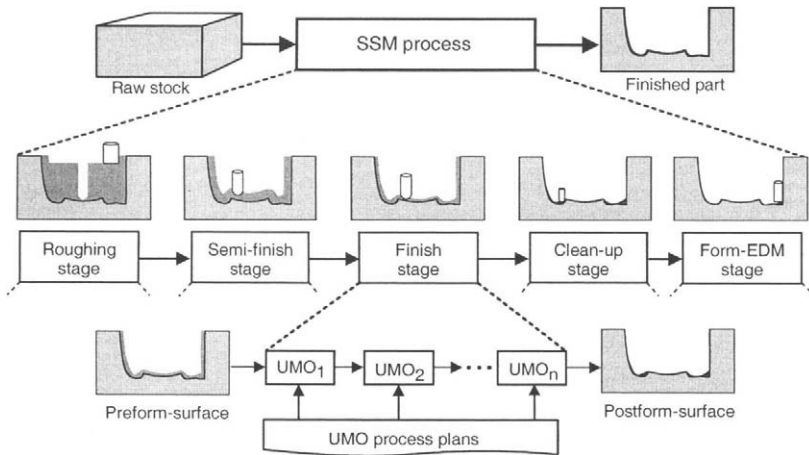


Figure 22.1. Hierarchical model of an SSM process.

In the abstract, an SSM process may be viewed as a sequence of **material removal functions** (MRF). In the most simplistic view, the SSM process can be regarded as a ‘sculpting process’ in which a ball-endmill cutter will form the machined-surface by a series of ‘touches’, just as a sculptor forms his massif by the classical process called ‘pointing’ (Duncan and Mair [10]). Taking this view, the entire surface is treated as

a single sculptured surface, and all the SSM operations are treated as a single material removal function called SCULPT. Of course, this model is too simple to be useful in metal cutting.

A natural extension of the sculptural model is to decompose the SCULPT operation into a number of specialized operations. If individual UMOs are used as transform functions, we will have a 'UMO model'. If machining stages are used, a 'machining-stage model' may be obtained. Based on the observation that each machining stage consists of a sequence of UMOs, one may obtain a model of the SSM process, as shown in Figure 22.1, in which it is modeled as a sequence of machining stages, and each machining stage is decomposed into a sequence of UMOs.

22.1.2. Information processing issues

The SSM problem is to generate: 1) a sequence of UMOs for machining the sculptured part, 2) a sequence of NC code blocks for each UMO, and 3) cutting conditions for each NC code block. We need a separate information processing stage for each of these three steps. Given below are the three information processing stages along with their functional requirements:

1. The **feature-based processing** stage generates UMOs with a minimum P/M (NC programming / NC machining) ratio.
2. The **geometric processing** stage obtains NC blocks with the minimum cutter-failure rate and the minimum cutter-gouge rate.
3. The **technological processing** stage obtains cutting conditions by which maximum cutting efficiency and minimum cutting-failure rate are achieved.

Feature-based information processing

Generating an SSM process (i.e. a sequence of UMOs) from input data requires a high-level decision-making function, which in turn requires feature-based information processing. A list of machining features is extracted from the geometric definition of the design surface, and then these are converted into a sequence of UMOs (unit machining operations). The first process is called feature extraction, and the second is called computer-automated process planning (CAPP).

The main issues at this stage are 1) how to define and extract the machining features and 2) how to define and obtain the UMOs. In SSM, these two areas, feature extraction and CAPP, have not yet been fully investigated.

Geometric information processing

Geometric information processing involves the generation and verification of NC data. As shown in Figure 22.2, the generation function consists of tool path planning and cutter location data computations, while the verification function involves cutting simulation and gouge detection. To describe the geometric information processing stage, the following terms are now introduced: a **CC-path** is a series of cutter-contact (CC) points where the cutter is tangent to the surface being machined; and a **CL-path** is defined as the locus of cutter-location (CL) points, typically at the center or tip of the tool. Brief descriptions of those operations are now given:

1. Tool path planning: for a UMO, CC-paths are obtained from the design surface.

2. CL-data computation: CC-paths are converted to CL-paths.
3. Cutting simulation: the workpiece is 'virtually machined'.
4. Gouge detection: the simulated machined surface is compared against the design surface.

The cutting simulation operation also involves computing metal-removal volumes (MRV) and checking for collisions.

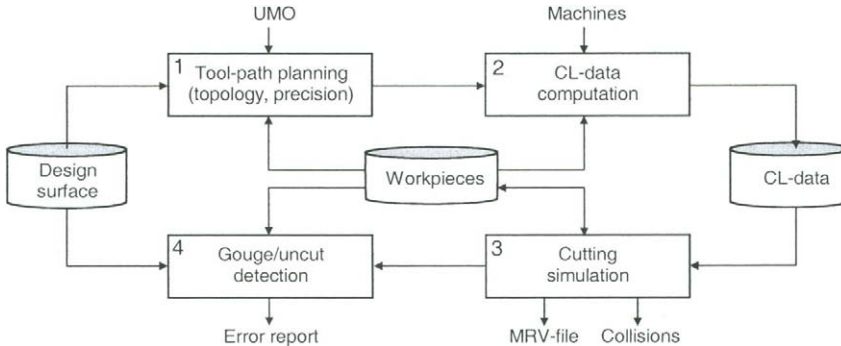


Figure 22.2. Geometric information processing.

The key issue at this stage is how to generate dependable NC data to minimize both the cutter failure rate and cutter gouge rates, while also achieving tool path economy (i.e. minimizing the effective length of the total cutter path). Another issue is how to automate the tool path generation process by using generative NC (GNC), in order to minimize the P/M ratio.

Technological information processing

Technological information processing is mainly concerned with cutting conditions, but it also involves selecting cutting tools and choosing milling strategy options etc. Once the tool path pattern has been determined, during the geometric information processing stage, cutting efficiency is dependent on the spindle speed and feedrate for each NC block. Ideally, the feedrate would be adaptively varied according to the changes in metal removal volume. In general, the machining process conditions are affected by non-geometric factors such as:

1. Tolerance requirements.
2. Surface-finish requirements.
3. Properties of the workpiece material such as hardness, strength, ductility, etc.
4. Cutting tool material (HSS, WC, CBN, ...), type, shape, etc.
5. Machine tool characteristics.
6. Milling strategy options (e.g. down-milling or up-milling, reverse-cutting or plunge-cutting, etc.)

The difficulty of technological information processing lies in the fact that there are so many variables to consider, because modern SSM operations have become so complicated. Thus, it is essential to have a cutting-condition DBMS (database management system) that is constantly updated, based on feedback from the shop floor.

22.2. UNIT MACHINING OPERATIONS

This section presents tool path topology, milling strategy options, and a comprehensive list of UMOs for 3-axis machining. Commonly used cutter types for SSM operations are the ball-endmill, flat-endmill, and round-endmill.

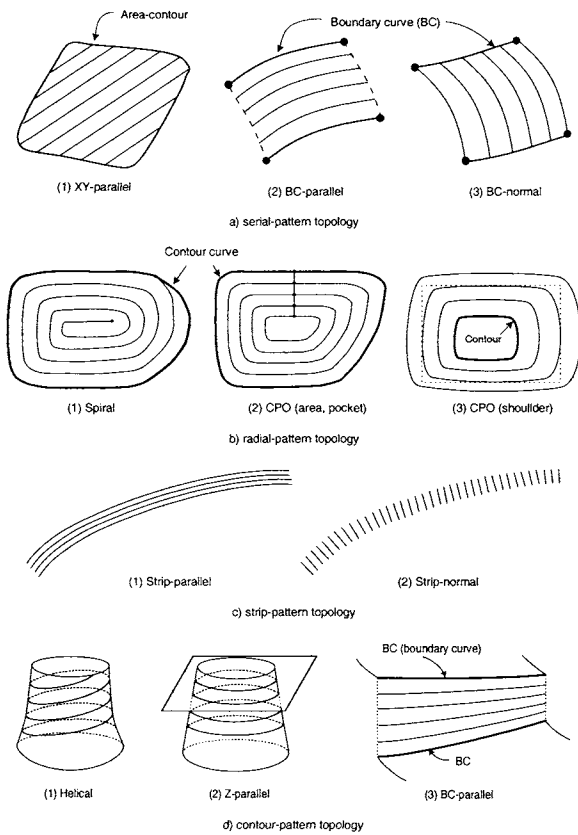


Figure 22.3. Tool path topology.

22.2.1. Tool path topology and milling-strategy options

Sculptured surface machining is a ‘point’ milling process in which a sequence of CC-points is traced by milling cutters. When a region is machined by the point milling method, the process is often called **regional milling**, and the pattern of ‘tracing’ or scanning is called the tool path topology (Marshall and Griffiths [14]). The traced region could be an area, a strip of fillets or a ‘wall’. There are four types of **tool path topology** pattern, as summarized below (where BC stands for boundary-curves and CPO stands for contour-parallel offset):

- Serial-pattern: xy -parallel, BC-parallel and BC-normal (Figure 22.3-a).
- Radial-pattern: spiral and CPO (Figure 22.3-b).
- Strip-pattern: strip-parallel and strip-normal (Figure 22.3-c).
- Contour-pattern: helical, z -constant, and BC-parallel (Figure 22.3-d).

Both the serial and radial types may be used for machining an area, and the contour type is appropriate for cutting a vertical or slanting wall. The spiral and helical topologies (Figure 3-b-1 and Figure 3-d-1) are widely used in **high-speed machining**. It should be noted that the so-called isoparametric tool path is a special case of the BC-parallel topology, which also includes the so-called isocurvature tool path proposed by Jensen and Anderson [13] and Suresh and Yang [21].

When planning for regional milling, it is also necessary to consider milling strategy options (Schulz and Hock [19]) and parameters related to the SSM-process such as:

- Milling mode: **up-milling** or **down-milling**.
- Vertical moves: **upward** or **downward** milling.
- Effective cutting edge: acceptable range of inclination angle (α) etc.
- Tool path linking: zigzag, one-way, etc.

22.2.2. Ball-endmill UMOs

The ball-endmill is by far the most popular cutting tool for sculptured surface machining. As shown in Figure 22.4, there are seven types of ball-endmill UMOs widely found in sculptured surface machining. Listed below are these ball-endmill UMOs together with their tool path topologies. (There are a large number of possible combinations of tool path topologies, but only the most common ones are considered here):

1. Area-cut: serial-pattern or radial-pattern topologies (Figure 22.4-a: xy -parallel).
2. Fillet-cut: strip-pattern topologies (Figure 22.4-b: strip-parallel).
3. Pencil-cut (Figure 22.4-c).
4. Contour-cut: contour-pattern topologies (Figure 22.4-d: helical).
5. Pocketing (hollow only): radial-pattern topologies (Figure 22.4-e).
6. Shouldering: radial-pattern or xy -parallel topologies (Figure 22.4-f).
7. Plane-step roughing (or plane-stepping): xy -parallel topologies (Figure 22.4-g).

The area-cut UMO is mainly used in generating a smooth surface by tracing the surface area. If a large ball-endmill is used during finish-machining, strips of uncut region may appear along the sharp concave fillets. These uncut regions in the concave fillets can be cleaned up by the fillet-cut UMO using a smaller ball-endmill. Since the radius of the

concave fillet is smaller than the radius of the ball-endmill, the ball-endmill will make multiple-point contact with the part surface. The trajectory of the center of the ball-endmill in this case is called a ‘pencil curve’, and the progress of the ball-endmill along the pencil-curve is called a **pencil-cut** UMO.

As shown in Figure 22.4-d, a ‘core-wall’ surface can be effectively machined by employing the contour-cut UMO. The remaining three UMOs in Figure 22.4 are for roughing operations: pocketing, shouldering and plane-stepping. They are based on the concept of **cutting layers** in which the volumes to be removed are sliced into layers by a number of (equally spaced) horizontal cutting planes. The thickness of the cutting-layer is often called the ‘plane step’, and so this type of roughing method is often called the plane-step method. Especially in pocketing, if the cavity-volume to be removed already has a cavity, as shown in Figure 22.4-e, the process is called **hollow pocketing**; if the cavity is to be machined from a solid stock, it is called (solid) pocketing.

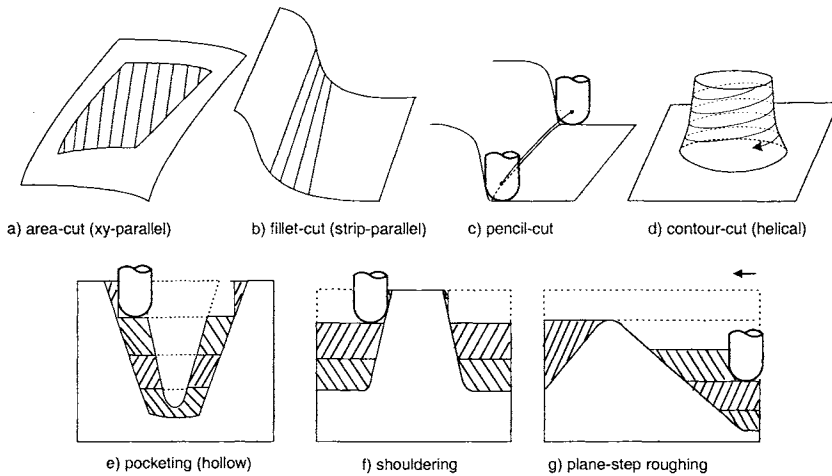


Figure 22.4. Ball-endmill UMOs.

22.2.3. Flat-endmill UMOs

There are seven types of 3-axis flat-endmill UMOs widely used in sculptured surface machining. They are listed below, together with their tool path topologies (but note that only the most common topologies are considered):

1. Fillet-cut (upward only): strip-normal topology (Figure 22.5-a).
2. Area-cut (downward only): BC-normal topology (Figure 22.5-b).
3. Slotting.
4. 2D-contouring.
5. Pocketing: radial-pattern topologies.

6. Shouldering: radial-pattern or xy -parallel topology.
7. Plane-stepping (upward only): xy -parallel topology.

With 3-axis NC machine tools, flat-endmills are rarely used for finish-machining, except in the two cases shown in Figure 22.5: that is, the ‘strip-normal topology’ shown in Figure 22.5-a and the ‘BC (boundary contour)-normal topology’ shown in Figure 22.5-b. The former is often used in finish-machining of injection molding dies, and the latter is mainly employed in clean-up machining. The two ‘simple’ UMOs, slotting and 2D-contouring, are used in the roughing stage as well as in the form-cutting stage (i.e. the form-EDM stage for concave sharp edges). The other three roughing UMOs—pocketing, shouldering and plane-stepping—are defined in almost the same way as their ball-endmill counterparts.

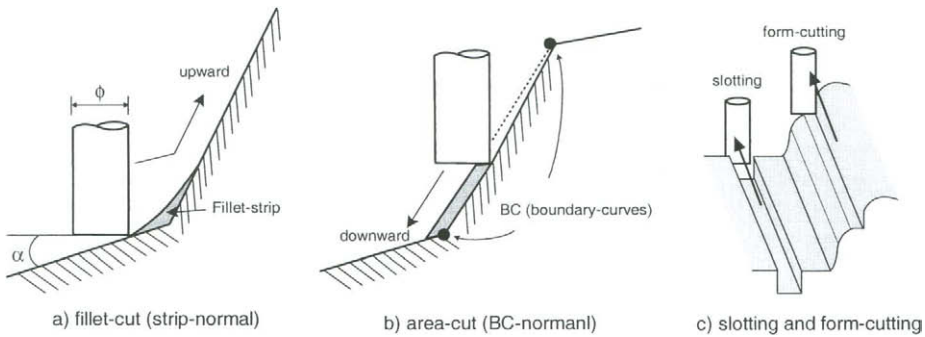


Figure 22.5. Flat-endmill UMOs.

22.3. INTERFERENCE HANDLING

Even with the most robust algorithms, there are many opportunities for making errors when generating the tool paths for sculptured surface machining (SSM). The kinds of errors considered in this section are:

1. **Gouging** of the part in excess of the in-tolerance value t_i .
2. **Collision** between a non-cutting cutter element (e.g. the cutter holder) and the workpiece.

The purpose of this section is to understand the nature of cutter interference in SSM, which is essential for preventing, detecting, and correcting errors. In this section, commonly found types of cutter interference are categorized as:

1. CL-point interference: gouging occurs at a CL-point.
2. CL-line interference: gouging occurs on a CL-line.
3. Collision: collision occurs during cutting motion.

22.3.1. CL-point interference

Figure 22.6-a shows a typical CL-point interference; this type is known as a **concave-gouge**, which can occur at a CL-point located in a concave region when a cutter contacting a CC-point invades other portions of the part surface. When a smooth surface is machined with a ball-endmill, a sufficient condition for concave-gouging is given by

$$\kappa_n > 1/\rho, \tag{22.1}$$

where ρ is the ball-endmill radius and κ_n is the maximum normal curvature. Even though some methods for preventing concave-gouging have been proposed in the literature (Choi and Jun [4]), it is not easy to detect (and correct) this type of gouging from a given tool path. Moreover, ‘correction’ of the concave-gouge would lead to a **CL-point uncut** as depicted in Figure 22.6-b.

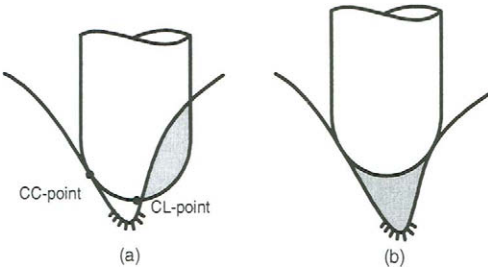


Figure 22.6. A concave gouge and an uncut.

22.3.2. CL-line interference

Even if there is no interference at each CL-point, a **convex-gouge** may occur during the movement of the tool along a CL-line, as shown in Figure 22.7-a. A convex-gouge may occur only at a CL-line (and a concave-gouge may occur only at a CL-point). Figure 22.7-a shows a typical convex-gouge associated with a ball-endmill. The ‘thickness’ of the convex-gouge (γ_i) is defined as the distance from the CC-point (\mathbf{r}_i) to the machined surface, and it is expressed (Choi and Jun [4]) as:

$$\gamma_i = \rho(1 - \sin \alpha_i), \quad \text{for } i = 1, 2, \tag{22.2}$$

where ρ is the cutter radius, $\alpha_1 = \angle \mathbf{r}_1 \mathbf{p}_1 \mathbf{p}_2$ and $\alpha_2 = \angle \mathbf{p}_1 \mathbf{p}_2 \mathbf{r}_2$ (see Figure 22.7-a).

In the extreme, the convex-gouge shown in Figure 22.7-a becomes the sharp-edge gouge shown in Figure 22.7-b. In both cases (Figures 22.7-a and 22.7-b), the thickness of the convex-gouge could be unacceptably high for a large value of α_i . The actual size of the convex-gouge is roughly equal to the sum of the gouge thickness γ given by (22.2) and the in-tolerance τ_i .

A simple way to correct (or remove) the convex gouge at \mathbf{p}_i is to insert additional CL-points at \mathbf{q}_i (for $i = 1, 2$), located (Choi and Jun [4]) at:

$$\mathbf{q}_i = \mathbf{p}_i + \rho \cdot (\mathbf{n}_i - \mathbf{c} / (\mathbf{n}_i \cdot \mathbf{c})) \cdot (1 / (1 - (\mathbf{n}_i \cdot \mathbf{c})^2)^{1/2} - 1), \quad \text{for } i = 1, 2, \quad (22.3)$$

where \mathbf{n}_i is the unit normal vector at \mathbf{r}_i and $\mathbf{c} = (\mathbf{r}_2 - \mathbf{r}_1) / |\mathbf{r}_2 - \mathbf{r}_1|$. Now the CL-path at the convex region is changed to the sequence

$$\rightarrow \mathbf{p}_1 \rightarrow \mathbf{q}_1 \rightarrow \mathbf{q}_2 \rightarrow \mathbf{p}_2 \rightarrow .$$

This correction is effective on both the convex-gouge of Figure 22.7-a and the sharp-edge gouge of Figure 22.7-b.

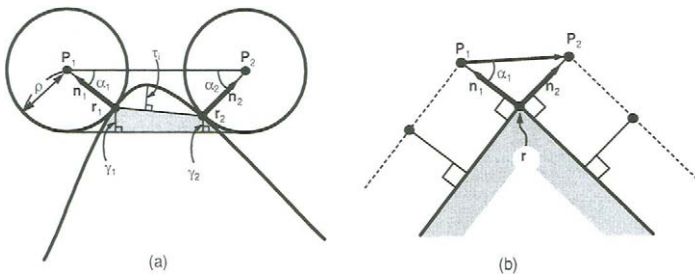


Figure 22.7. Convex gouging and sharp-edge gouging.

22.3.3. Collisions

A cutting tool is supposed to interact with the workpiece only through its ‘cutter-part’ (or cutting-edge), and if its non-cutting portion makes contact with the workpiece, there is a collision. In fact, there may be three types of collision: 1) holder-collisions, as shown in Figure 22.8-a, 2) shank-collisions as in Figure 22.8-b, and 3) dead-center collisions, as shown in Figure 22.8-c.

22.4. TOOL PATH GENERATION METHODS AND CONSEQUENT GEOMETRIC ISSUES

22.4.1. The conventional approach and the C-space approach

The conventional approach

In CC-based TPG (tool path generation) methods, tool paths are generated by sampling a sequence of CC-points from the part surface, and then each CC-point is converted to a CL-point. Here, the part surface is used as a path-generation surface on which tool paths are generated. These are often called conventional TPG-methods.

There are three types of path planning domain in which tool path patterns are planned: the parameter domain, the guide-plane and the drive-surface. Thus, depending on the type of path-planning domain, CC-based TPG-methods can be grouped into the three cases shown in Figure 22.9:

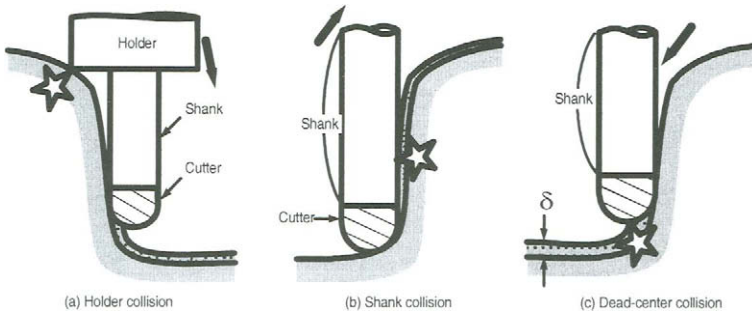


Figure 22.8. Collisions (holder and shank).

1. The isoparametric method: CC-paths are planned on the parameter domain of the part surface and then they are mapped on to the part surface (Figure 22.9-a).
2. The Cartesian method: tool paths are planned on a guide-plane and then tool positions are projected on to the part surface (Figure 22.9-b).
3. The APT (Automatically Programmed Tool) method: tool paths are defined by intersecting the part surface with a series of drive-surfaces (Figure 22.9-c).

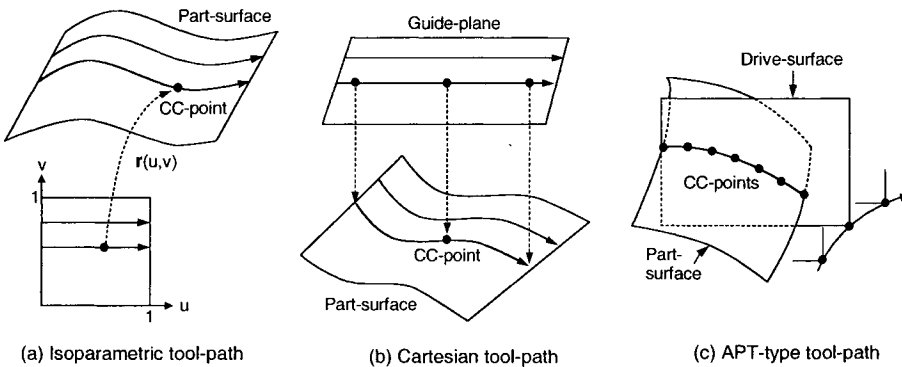


Figure 22.9. CC-based tool path generation methods.

The isoparametric method is the simplest TPG-method, but it may not be suitable for machining a compound surface consisting of a collection of surface patches. Moreover, this method is susceptible to concave gouging. Thus, the alternative Cartesian method is widely employed in modern CAD/CAM systems. The APT method has been employed

in the APT system. In all these three cases, however, the CL-data computation is carried out in the following three phases:

1. Mapping: computation of a CL-point for a given 'domain-point'.
2. Marching: finding the next domain-point from the current point on the path.
3. Side-stepping: finding the initial domain-point on the next path.

The C-space approach

The C-space (configuration space) idea has been widely used in spatial planning for robot manipulators [16] and may also be applied to NC tool path planning by treating the cutter assembly as the moving object and the workpiece and fixtures as the obstacles. The configuration of a 3-axis NC machine (i.e. its cutter) is the 3D position vector denoting a CL-point, and its C-space is given by the **NC-volume** V_{NC} within reach of the cutter. In tool path planning, however, there are two types of safe C-space: 'free-travel' C-space and 'machining' C-space.

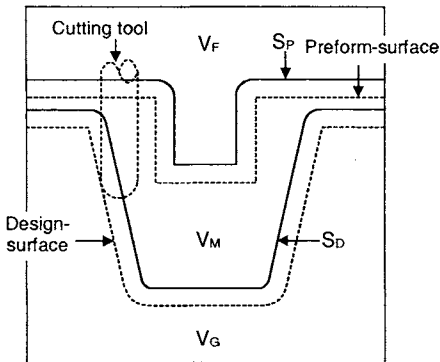


Figure 22.10. Construction of C-space for 3-axis machining.

Now we will consider two CL-surfaces (S_D and S_P), one from the design-surface and the other from the preform-surface, as depicted in Figure 22.10:

- S_D : CL-surface for the design-surface.
- S_P : CL-surface for the preform-surface (or raw-stock).

Then, as shown in Figure 22.10, the two CL-surfaces would divide the entire C-space V_{NC} into the following three disjoint C-space volumes:

- V_F : the 'free-travel' C-space volume (the region above S_P).
- V_M : the 'machining' C-space volume (the region between S_P and S_D and including both surfaces).
- V_G : the 'gouging' C-space volume (the region below S_D).

The geometric entities S_P , S_D , V_F , V_M , V_G are called **C-space elements**.

As will be discussed shortly, these C-space elements contain all the geometric information needed for generating tool paths. In summary, the C-space approach to tool path generation may be formalized as follows:

1. Compute S_P : CL-surface of the preform-surface.
2. Compute S_D : CL-surface of the design-surface (with an uncut-allowance).
3. Compute V_F , V_M , V_G : volume-type C-space elements:
4. Generate tool paths from the C-space elements.

Compared with conventional methods, the key feature of the C-space approach is that all the decisions, including global ones such as feature extraction and adaptive feed control, can be based solely on the C-space representation.

22.4.2. Geometric issues in conventional approach

The geometric information processing of SSM may be classified into a tool path generation (TPG) step and an interference handling step in which geometric issues are identified; this is the case both in the conventional approach and in the C-space approach. Table 22.1 summarizes the geometric aspects of the conventional approach, while the six key geometric procedures in the TPG step are listed below:

- *A sliding cutter trajectory between the part surface and a horizontal plane* is traced to generate a BP (boundary pocket)-curve or a z -constant contour curve. However, many researchers are still struggling to design and implement a robust tracing algorithm (see Figure 22.11-a).
- *A 2D PS(point sequence)-curve offsetting algorithm* is used to generate a series of contour-parallel offset (CPO) curves especially in pocket machining. A more detailed description of this issue is presented in Section 22.5.2.
- *PS-curves defined during tool path planning are projected* on to the part surface to generate CPO-paths or direction-parallel CC-paths. Vertical and horizontal projection are usually sensitive in near-vertical and near-plane regions of the part surface respectively. Depending on the shape of the part surface, we should select an appropriate projection method, such as the normal projection.
- *Sliding cutter trajectories in concave regions* are traced to detect pencil-curves on the part surface. The trajectory of the center of the ball-endmill in this case is the pencil-curve (see Figure 22.11-b).
- *Sliding cutter two-point contact curve pairs* are traced to extract clean-up machining regions. The sliding cutter in this case is the smallest used in finish machining (see Figure 22.11-b).
- *PS-curve fairing* is used in refining pencil-curves and correcting CL-paths to improve the quality of the machined surface.

The following four key geometric issues are also identified in the interference handling step:

- *Principal curvature calculations and Boolean operations between the cutter and the workpiece* are used to detect concave-gouging (CL-point gouging) in SSM.

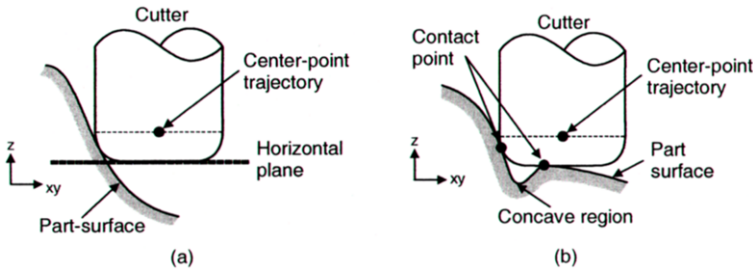


Figure 22.11. Sliding cutter trajectory.

- Boolean operations between the volume swept by the cutter and the workpiece are used to detect convex-gouging (CL-path gouging).
- Boolean operations between the volume swept by the tool assembly and the workpiece are used to check for collisions.

In these cases, effective performance of the Boolean operations requires a unified scheme for representing the cutter and the workpiece. There is no general algorithm to prevent the erosion of sharp edges in the conventional approach, so a special algorithm has to be designed and implemented for each case.

Table 22.1
Geometric issues in the conventional approach

Information Processing	Key Geometric Operations	Required Functions
Tool Path Generation	Sliding cutter trajectory tracing between the part surface and a horizontal plane	BP-curve generation Z-constant contour path generation
	2D PS-curve offsetting	Generation of CPO-paths
	PS-curve projection	Direction-parallel path generation
	Sliding cutter trajectory tracing in concave regions	Pencil-curve tracing
Interference Handling	PS-curve fairing	Pencil-curve refinement CL-path smoothing and correction
	Sliding cutter two-point contact curve pair tracing	Clean-up region detection
	Surface trimming	Clean-up path generation
	Surface principal curvature calculation	Concave-gouge avoidance
Interference Handling	Cutter/workpiece Boolean operation	Concave-gouge avoidance
	Cutter swept volume/workpiece Boolean operation	Convex-gouge avoidance
	Tool assembly swept volume/workpiece Boolean operation	Sharp-edge erosion prevention
	Tool assembly swept volume/workpiece Boolean operation	Collision avoidance

22.4.3. Geometric issues in the C-space approach

In the C-space approach, six geometric issues may be identified in the TPG step and two more become apparent in the interference handling step. PS-curve handling algorithms, discussed in the conventional approach, can also be applied to the C-space approach. The six key geometric issues in the TPG step can be summarized as follows.

- The concept of the *Minkowski sum* is used to construct the inverse tool offset surface, the CL-surface. A unified scheme for consistent representation the part surface and the CL-surface is required to construct the CL-surface effectively. In Section 22.5.1 an algorithm for constructing the CL-surface is described. The z-map model is used as a representation scheme for the CL-surface.
- A z-constant contour curve including a BP-curve are computed by *intersecting the CL-surface and a horizontal plane*. The z-constant contour curve becomes a path segment in the C-space approach.
- As in the case of the conventional approach, *2D PS-curve offsetting*, *PS-curve projection*, and *PS-curve fairing* are employed to handle PS-curves (in the generation of CPO-paths, direction-parallel tool path generation, and in pencil-curve refinement).
- Since pencil-curves are represented as sharp-edges in the CL-surface, we use *the algorithm for tracing sharp-edge curves* to detect pencil-curves and clean-up regions in the C-space approach.

Table 22.2
Geometric issues in the C-space approach

Information Processing	Key Geometric Operations	Required Functions
Tool Path Generation	Minkowski sum	CL-surface construction
	Intersection between CL-surface and a horizontal plane	BP-curve generation Z-constant contour curve generation
	2D PS-curve offsetting	Generation of CPO-paths
	PS-curve projection	Direction-parallel path generation
	Sharp-edge curve tracing	Pencil-curve tracing Clean-up region detection
	PS-curve fairing	Pencil-curve refinement CL-path smoothing and correction
Interference Handling	Height Boolean	Collision detection
	Non-circular offsetting	Sharp-edge erosion prevention

The C-space of the tool assembly is constructed from the part surface using the concept of the Minkowski sum. Collisions between the part surface and the tool assembly are easily detected using height Boolean operations, because the tool assembly is represented as a line segment in its C-space. To prevent sharp-edge erosion in the C-space approach, we must use a special method, such as a non-circular offsetting of the sharp edges. (If a cutter moves along a circular offset tool path which crosses a convex sharp edge, the cutter contacts only one point on the convex sharp edge. In that case, the convex sharp

edge should be eroded because of the interpolation characteristics of CNC controllers and spindle vibration.) Table 22.2 summarizes key geometric issues in the C-space approach.

22.5. GEOMETRIC ALGORITHMS

22.5.1. CL-surface construction

The merit of the C-space approach can only be realized through a reliable and efficient implementation scheme. To construct the CL-surface, we first of all select its representation, considering the following factors:

- The CL-surface (S_D) can be represented in a nonparametric form: $z = f(x, y)$.
- S_D can be constructed by computing its inverse tool-offset (ITO) surface.
- For chip-load leveling, it is necessary to trace the sharp edges of S_D . (An abrupt jump in chip load may result in cutter breakage)
- For cutting-load smoothing, it is necessary to compute normal curvatures of S_D .
- For tool path planning, it is necessary to extract machining features from S_D .
- For tool path generation, it is necessary to intersect S_D with planes.

The CL-surface may be represented as a triangulated facet model (Choi et al. [3] and Sheng and Hirsch [20]) or as a **Z-map model**. It turns out that the above requirements are quite effectively covered by the Z-map model. A Z-map, also known as a **G-buffer** (Saito and Takahashi [18]), is nothing but a 2D-array of z -values of the surface sampled at points on a regular grid. The x, y coordinates of the Z-map element $z[i, j]$ are computed as

$$x[i] = x[0] + \gamma_x \cdot i \quad \text{and} \quad y[j] = y[0] + \gamma_y \cdot j, \quad (22.4)$$

where $(x[0], y[0])$ is the grid-point at the bottom-left corner of the Z-map domain, and γ_x and γ_y denote the grid intervals.

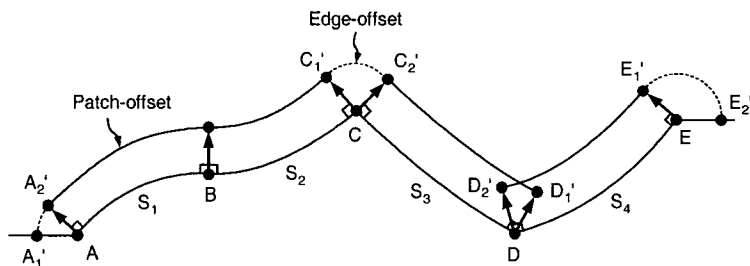


Figure 22.12. Patch-offsets and edge-offsets in a CL-surface.

This section presents a method for obtaining the Z-map model of a CL-surface called the **CL Z-map**. The method for constructing a CL Z-map from a CAD model having

a number of trimmed parametric surface patches is called the offset-surface digitizing method. Throughout the rest of this section, the term ‘master Z-map’ will be used to denote the Z-map model of the part surface to be produced or its preform-surface. The generation of a master Z-map from a CAD model is called **Z-map sampling** or **virtual digitizing**, which may be carried out by using the 2D Jacobian inversion algorithm (Choi [5]).

Once the Z-map structure is obtained, the CL-surface of the CAD model has to be ‘digitized’ again at each grid-point of the Z-map model. Since the 2D Jacobian inversion algorithm would become unstable for a CL-surface, we use the ‘offset-surface digitizing method’ in constructing a CL Z-map (Choi and Jerard [9]).

Figure 22.12 shows a two-dimensional cross-sectional view of a CAD model and its CL-surface. The CAD model consists of four trimmed parametric surface patches S_i together with five edges, A, B, C, D and E. Note that A and E are boundary edges, B is a smooth common edge, C is a convex edge and D is a concave edge. Also depicted in the figure are patch-offsets and edge-offsets: one patch-offset surface for each surface patch and an edge-offset surface for each convex-edge and boundary-edge. (Smooth-edges and concave-edges are simply neglected.)

As depicted in Figure 22.12, the CL-surface is a compound surface consisting of patch-offset surfaces and edge-offset surfaces. Thus, in the offset-surface digitizing method, the CL Z-map is constructed by digitizing the individual offset surfaces. (If more than one z -value is sampled for a given grid-point, the highest is selected.) Each patch-offset surface is digitized by performing Z-map sampling on its triangulated facet-model, while edge-offset surfaces are digitized by simulating the cutting of an ‘inverse tool’, as depicted in Figure 22.13. Thus, the overall procedure for constructing the CL EZ-map (extended Z-map) may be expressed as follows:

Input: a CAD model consisting of parametric surface patches, together with the cutter geometry.

Step 1. Construct a master Z-map model using the 2D Jacobian algorithm.

Step 2. Construct a triangulated facet model for each of the patch-offset surfaces.

Step 3. Digitize each individual offset triangle in the triangulated facet-models.

Step 4. Find all the convex and boundary edges.

Step 5. Perform ‘inverse tool’ cutting simulation along each of the edges found in Step 4.

Figure 22.14-a shows a parametric surface patch whose offset surface is to be digitized. First, the surface patch is discretized and triangulated as, depicted in Figure 22.14-b, and then the vertices of each triangle are offset along their ‘offset vectors’ to obtain an offset triangle as shown in Figure 22.14-c. For a ball-endmill of radius ρ , the offset vector may be given by $\rho \cdot \mathbf{n}$, where \mathbf{n} is the unit normal vector. Finally, as shown in Figure 22.14-d, each offset triangle is digitized.

A number of **surface-discretization** algorithms are available in the literature (see Austin et al. [2]). One of the key issues in discretization is the choice of an appropriate resolution, the step-length λ between two adjacent vertices, while keeping the discretization error ε within the input tolerance τ . A simple method is to approximate the curve joining the two vertices with a circular arc of radius ρ_k , so that the step length λ is given

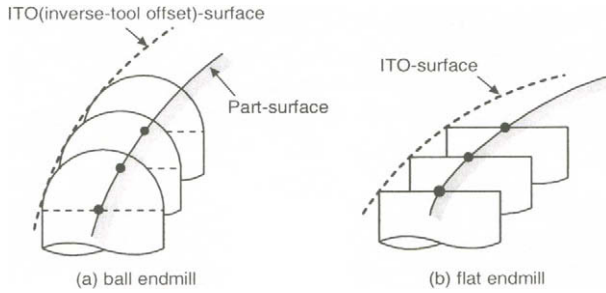


Figure 22.13. Cutting by a simulated ‘inverse tool’.

by

$$\lambda = 2 \cdot (2 \cdot \rho_k \cdot \varepsilon - \varepsilon^2)^{1/2}. \tag{22.5}$$

However, as shown in Figure 22.15, the offset surface discretization error, ε^o , is different from the base surface discretization error, ε , although they are related to each other as follows:

$$\varepsilon^o = \varepsilon \frac{(\rho_k \pm \rho_c)}{\rho_k}, \tag{22.6}$$

where ρ_c is the radius of the (ball-endmill) cutter. (The \pm becomes $+$ for a convex region and $-$ for a concave region.) Since $\varepsilon^o \leq \tau$, from the results of (22.5) and (22.6), the triangulation step-lengths λ for the convex case (Figure 22.15-a) and concave case (Figure 22.15-b) are expressed as follows:

$$\lambda_{\text{convex}} \leq 2 \cdot (2 \cdot \rho_k \cdot \tau_{\text{conv}} - (\tau_{\text{conv}})^2)^{1/2} \text{ and} \tag{22.7}$$

$$\lambda_{\text{concave}} \leq 2 \cdot (2 \cdot \rho_k \cdot \tau_{\text{conc}} - (\tau_{\text{conc}})^2)^{1/2}, \tag{22.8}$$

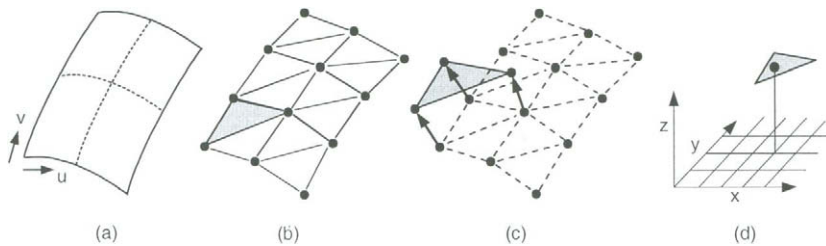


Figure 22.14. Virtual digitizing by means of offset-surface triangulation.

with

$$\tau_{\text{conv}} = \tau \cdot \rho_k / (\rho_k + \rho_c) \text{ and}$$

$$\tau_{\text{conc}} = \tau \cdot \rho_k / (\rho_k - \rho_c) \text{ for } \rho_k > \rho_c,$$

where ρ_k is the radius of the normal curvature in the direction of the ‘next’ vertex and ρ_c is the (effective) radius of the cutter (in the same direction).

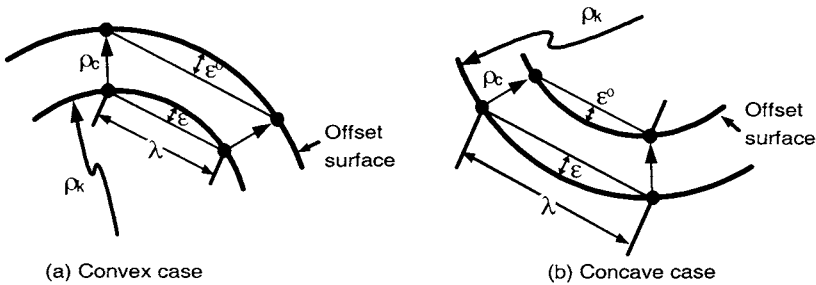


Figure 22.15. Triangulation step-lengths and discretization errors.

It should be noted that the inequalities (22.7) and (22.8) are undefined in ‘narrow concave regions’ where the radius of normal curvature is less than or equal to the cutter radius. (Such regions are simply excluded from digitizing.)

The above procedure for determining step-length may easily be implemented for a ball-endmill. However, the procedure would be quite involved for a round-endmill because:

1. The effective cutter radius ρ_c is not easily computed, and
2. Inequality (22.6) is no longer valid in horizontal regions where the normal vector \mathbf{n} of the surface is vertical. (The horizontal regions may be digitized by simulating inverse cutting.)

22.5.2. 2D PS-curve offsetting

The 2D-curve offsetting problem (Tiller and Hansen [23]) has been regarded as one of the key issues in generating tool paths for 2D pocketing. In general a 2D curve may be 1) in parametric form, for instance a NURBS-curve, 2) a curve consisting of lines and arcs, or 3) a curve defined by a sequence of points (PS-curve).

The parametric-curve offsetting problem was formulated mainly in terms of self-intersection while the main reason for offsetting a line/arc-curve was to obtain a non-self-intersecting offset curve. Line/arc-curve offset methods may further be classified into the curve-based approach and the area-based approach. The area-based methods make use of related concepts such as the bisector and Voronoi algorithms, which provide more stable means to obtain an offset curve. The subject of offsetting a PS-curve has received less attention,

perhaps because it can be approximated by a line/arc-curve. However it still remains a hot issue in developing commercial SSM software.

This section describes a robust pairwise offset algorithm that obtains the valid offset curves of an input boundary PS-curve. Initially, the PS-curve is given as a closed sequence of 2D points, where each point is regarded as a vertex. In a preprocessing step, collinear vertices may be removed from the initial PS-curve. This is not essential for the algorithm but it may help speed up subsequent processes. After that, the PS-curve is converted into a counter-clockwise linked-list of segments, during which a list of pointers (to the PS-curve) for the convex vertices is also constructed. The PS-curve offsetting algorithm is described as follows (Choi [5]):

2D PS-curve offsetting

Input: PS-curve, offset direction, offset distance (ρ).

Step 1. Remove all local interfering ranges (LIR):

While (there remains a convex vertex in the PS-curve) do {

1. Get the parameter value t_c of a convex vertex;
2. Set $f = t_c$ and $b = t_c$;
3. Perform pairwise interference detection (f, b);
4. Remove the LIR = $[b, f]$ from the PS-curve;

};

Step 2. Construct a raw offset curve from the remaining segments in the PS-curve.

Step 3. Find all self-intersections of the raw offset curve using a sweep-line algorithm.

Step 4. Remove all global interfering ranges (GIR):

GIR = Φ ;

While ('unmarked' self-intersections remain in the raw offset curve) do {

1. Select an unmarked self-intersection point $P = (f_s, b_s)$ and mark it;
2. Set $f = f_s$ and $b = b_s$;
3. Perform pairwise interference detection (f, b);
4. Mark the self-intersection point $Q = (f, b)$;
5. GIR = GIR $\cup [f_s, f] \cup [b, b_s]$;

}

Remove GIR from the raw offset-curve;

Step 5. Construct valid offset curves from the remaining segments of the offset curve.

Step 1 of the algorithm will be explained with the LIR of the PS-curve that is shown in Figure 22.16. As depicted in Figure 22.16-a, the convex vertex shown as $t_c = 0$ is selected as a local seed point in Step 1-1. In Step 1-3, the pairwise-interference-detection (PID: f, b) function is called with $f = b = 0$, which will return the parameter values ($f = f_e = 3.5, b = b_e = 23.2$) at the contact-points of the common tangential circle. In Step 1-4, the LIR₁ = [23.2, 3.5] is deleted from the PS-curve, as depicted in Figure 22.16-b. In the next iteration of the *While* loop, another convex vertex at $t_c = 20$ is selected, in Step 1-1; and then, in Step 1-3, the PID function is called with $f = b = 20$ and the parameter values ($f = f_e = 4.2$ and $b = b_e = 15.1$) for a new LIR are returned. Next, in Step 1-4, the new interference range, LIR₂ = [15.1, 4.2], is deleted (see Figure 22.16-c). During a further iteration of the *While* loop, another interference range, LIR₃ = [9.7, 4.3],

is detected and deleted (see Figure 22.16-d). In the example shown in Figure 22.16, one may observe that a segment is visited only once, even though it may belong to more than one LIR (since $LIR_1 \subset LIR_2 \subset LIR_3$).

A comment on the choice of local seed points in Step 1 may be in order. For the PS-curve of Figure 22.17, the convex vertices shown as $t = 0, 20, 15$ (hollow points) were selected ‘at random’ as local seed points (Step 1-1 of the algorithm). In theory, the performance of Step 1 can be improved by employing an ‘optimal’ strategy for selecting a local seed point. In practice, however, defining such a strategy is not easy, and finding a better seed point imposes an additional computational burden.

Since all the segments belonging to local interference ranges have been deleted from the PS-curve during Step 1, it now contains only five linear segments and one reflex segment (Figure 22.17-b). In Step 2 of the algorithm, a raw offset curve of the resulting PS-curve is constructed (Figure 22.17-c). In Step 3 of the algorithm, self-intersection points of the raw offset curve are computed (Figure 22.17-d).

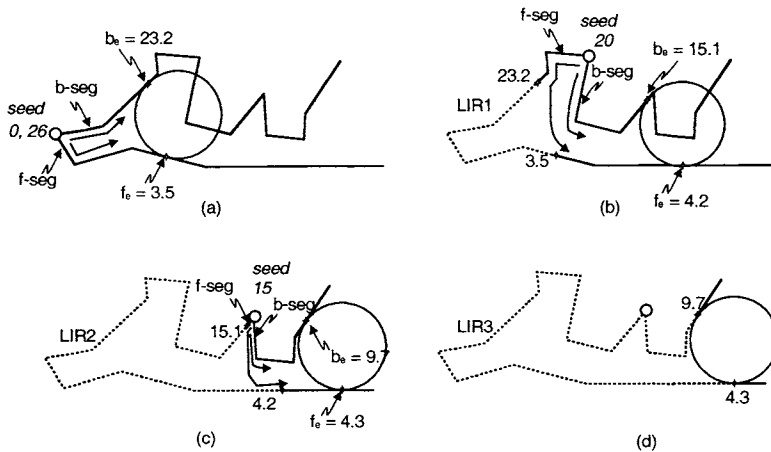


Figure 22.16. Detection and deletion of local interference ranges.

Now we describe in detail the operations performed in Step 4 of the algorithm. In Step 4-1, a self-intersection point P (Figure 22.17-d) is selected and tracing directions are assigned by setting $f_s = 4.5$ and $b_s = 8.2$. Since the self-intersection point P of the raw offset curve is the center point of a tangential circle of the PS-curve, the tracing directions are assigned in such a way that the tangential circle ‘gouges’ the PS-curve. Now, using the selected point P ($f_s = 4.5, b_s = 8.2$) as a global seed point, the PID (f, b) function is called with $f = 4.5$ and $b = 8.2$, in Step 4-3. The PID function will then return the parameter values at another self-intersection point Q ($f = 4.7, b = 6.8$). The two self-intersection points P and Q are ‘marked’ in Steps 4-1 and 4-4, respectively. In Step 4-5, the global interference range, $GIR = [4.5, 4.7] \cup [6.8, 8.2]$, that was detected in Step 4-3 is collected.

The collected GIRs are deleted after the *While* loop (i.e. in Step 4-6) because otherwise small portions of some GIRs might be traced more than once.

22.5.3. Area scan algorithm

This section describes an area scan algorithm mainly used in direction-parallel area milling. The algorithm handles multiply-connected areas and is not restricted to specific types of contour elements such as line segments or circular arcs (Park and Choi [17]). For the sake of simplicity, however, we assume that the area curves are PS-curves consisting of a large number of points. In practice, area curves are usually given as PS-curves obtained from surface-surface intersections (i.e. the intersection between the CL-surface and a plane) or feature curves extracted from the CL-surface. We divide the algorithm into two modules: 1) finding the **optimal inclination** and 2) calculating and storing tool path elements.

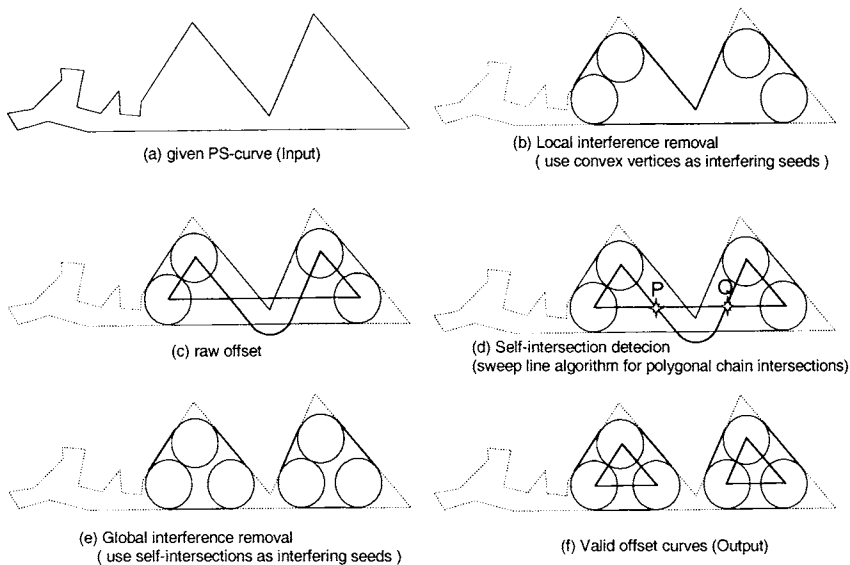


Figure 22.17. A raw offset curve and the deletion of the global interference range.

Optimal inclination

In actual machining, the inclination of milling may be decided by the user, based on technological requirements such as the feature (slope) of the part surface or constraints imposed by the machining configuration (NC-machine, jigs and fixtures). Otherwise the tool path planning system should be able to find an optimal inclination that satisfies three objectives: 1) minimizing the number of tool retractions, 2) minimizing the number of tool

path elements and 3) maximizing the average length of tool path elements. Figure 22.18 shows that the inclination is related to three objectives.

Note that the number of tool retractions may not be decided by the inclination alone, because it is also influenced by the tool path linking method. In order to incur the minimum number of tool retractions, some researchers have suggested algorithms that perform two-step optimizations: global optimization (selecting optimal inclination) and local optimization (tool path linking method). But for the other two objectives, minimizing the number of tool path elements and maximizing the average length of tool path elements, there is no available prior research, to the best of our knowledge. We do not explicitly consider the average length of tool path elements, because the minimum number of tool path elements will in any case maximize their average length.

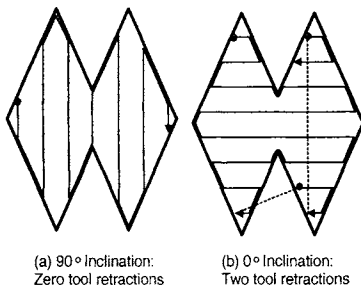


Figure 22.18. Inclination of milling and its objectives.

Before giving a formal description of the proposed module, we first introduce some basic terminology. A vertex is called **reflex** if the internal angle between its incident segments is greater than π , and convex otherwise (Figure 22.19). Given some inclination, a reflex vertex ν is called a **scan-reflex vertex** if there is a neighborhood δ at ν such that the boundary of the inside area is on one side of the line parallel to the inclination and passing through ν (Figure 22.19). A sequence of consecutive reflex vertices is called a **reflex profile** if its tangent range is smaller than π (Figure 22.20-a), otherwise it is divided into several reflex profiles (Figure 22.20-b). Note that a reflex profile has at most one scan-reflex vertex with respect to an arbitrary inclination. The length of a reflex profile is defined as the sum of the lengths of all its segments.

To fulfil the objectives listed above, an inclination should be chosen by considering not only the geometric shape of the area but also the tool path interval. Some researchers have suggested algorithms that select an inclination which minimizes the number of local extrema (scan-reflex vertices), but without considering the tool path interval. As a result these algorithms may not properly deal with the local (small) features of the boundary curves and thus fail to minimize the number of tool path elements.

The proposed module has to find the inclination that minimizes the number of scan-reflex vertices after removing reflex profiles with a length smaller than the tool path

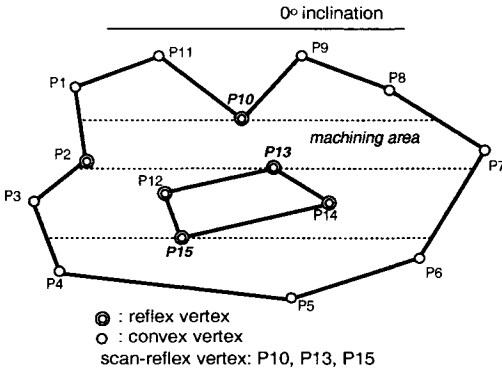


Figure 22.19. Elements in an area curve.

interval, and also local (small) features. The algorithm is described as follows:

- Step 1. Identify all the reflex vertices on the area curve, and construct a reflex profile set by grouping consecutive reflex vertices.
- Step 2. Remove reflex profiles with a length smaller than the tool path interval from the reflex profile set. (Local features are removed from the reflex profile set.)
- Step 3. Compute tangent ranges of the reflex profiles in the set and sort them (Figure 22.21-b).
- Step 4. Find an angular range belonging to the smallest number of tangent ranges (Figure 22.21-c).
- Step 5. Pick the central inclination inside the selected angular range.

Note that the angular range found in Step 4 contains the minimum number of scan-reflex vertices because a reflex profile gives at most one scan-reflex vertex when the inclination belongs to the tangent range

Calculating and storing tool path elements

This module has two objectives: one is to find tool path elements efficiently and the other is to store them in a suitable data structure for tool path linking. For the former purpose, we use the plane sweep paradigm and the concept of a monotone chain. A graph-like structure, the tool path element net (TPE-net) is suggested to achieve the latter goal.

A chain C is monotone with respect to a line L if C has at most one intersection point with each line L' perpendicular to L (Figure 22.22-a). The line L is called the monotone direction and the line L' becomes a sweep line.

Without loss of generality, we may assume that the x-axis (0°) is the inclination found in the previous module: i.e. the reference lines (or tool path elements) are horizontal. Firstly, we decompose area curves into monotone chains with respect to the y-axis. After decomposition, the tool path elements are calculated by sweeping a horizontal line across the monotone chains from top to bottom. During sweeping we maintain a monotone chain

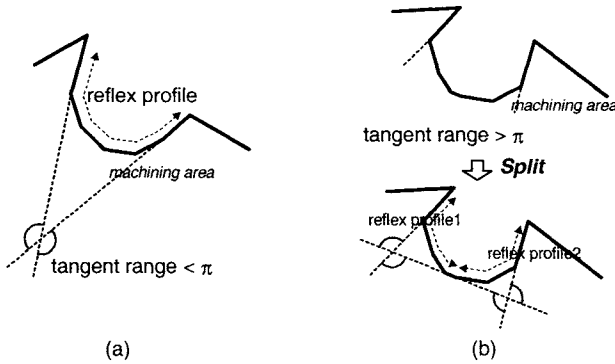


Figure 22.20. Reflex profiles.

list (MCL) which is crossed by a reference line, and the list is ordered by the x -values at their intersection. Note that the number of monotone chains is not changed unless the sweep passes over a scan-reflex vertex. At an event corresponding to a scan-reflex vertex ν , a pair of monotone chains is inserted into the MCL if ν has a local maximum y -value, or deleted from the MCL if ν is locally minimal (Figure 22.23). Then at each reference line, the tool path elements can easily be obtained by checking intersections between the reference line and the monotone chains in the MCL.

A tool path element E_{ij} is intuitively represented using the ‘TPE-node’ (tool path element node) (Figure 22.24-b). We suggest a graph-like structure to denote the connectivity relationships among TPE-nodes called a TPE-net (tool path element net) consisting of TPE-nodes and arcs (Figure 22.24-c). Note that to achieve the fifth requirement ‘movement along boundary curves’, every arc in the TPE-net should contain geometric information about the corresponding segment of the boundary curve. Owing to the intuitiveness of the TPE-net, we can easily construct and handle it in the next module, tool path linking.

22.5.4. Point-sequence curve fairing

This section describes a fairing algorithm for a PS-curve such as a pencil curve consisting of pencil-points. In this case, the 3D coordinates $\mathbf{r}_j = (x_j, y_j, z_j)$ of points can be decomposed into ‘domain’ coordinates $\mathbf{p}_j = (x_j, y_j)$ and ‘height’ coordinates $\mathbf{q}_j = (s_j, z_j)$, where s_j is the cumulative length of the PS-curve given by

$$s_j = \sum_{i=0}^j |\mathbf{p}_i - \mathbf{p}_{i-1}| \quad \text{for } j = 1, 2, \dots \quad \text{with } s_0 = 0. \tag{22.9}$$

PS-curve fairing is performed using a systematic fairing scheme based on difference operators. Our strategy is to apply the point-data fairing operations separately to the domain coordinates (\mathbf{p}_j) and the height coordinates (\mathbf{q}_j).

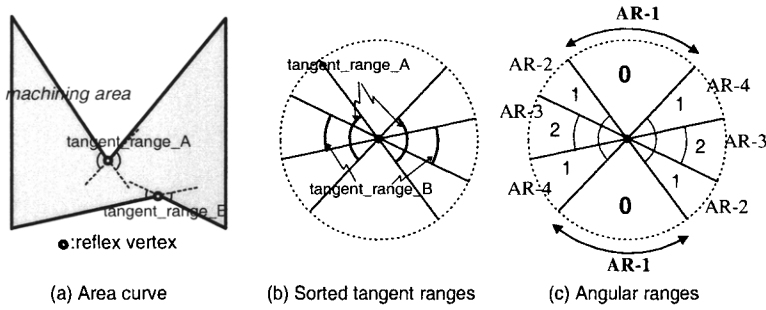


Figure 22.21. Finding an angular range belonging to the smallest number of tangent ranges.

For a 3D point sequence \mathbf{r}_j , the n th difference is defined as

$$D_j^n = D_{j+1}^{n-1} - D_j^{n-1} \quad \text{with } D_j^0 = \mathbf{r}_j. \tag{22.10}$$

Then, by setting Equation (22.10) to zero for $n = 2$, the ‘ideal’ position $\hat{\mathbf{r}}_j$ for the input point \mathbf{r}_j can be computed from the **2nd-difference fairing equation** given by

$$\hat{\mathbf{r}}_j = \frac{\mathbf{r}_{j+1} + \mathbf{r}_{j-1}}{2}. \tag{22.11}$$

Similarly, by setting Equation (22.10) to zero for $n = 4$, the ideal position $\hat{\mathbf{r}}_j$ can be computed from the **4th-difference fairing equation** given by

$$\hat{\mathbf{r}}_j = \frac{\mathbf{r}_{j+1} + \mathbf{r}_{j-1}}{2} + \frac{(\mathbf{r}_{j-1} - \mathbf{r}_{j-2}) + (\mathbf{r}_{j+1} - \mathbf{r}_{j+2})}{6}. \tag{22.12}$$

The physical meaning of the above fairing equations is shown in Figure 22.25: 2nd-difference fairing straightens the curve, while 4th-difference fairing leads to a curve with

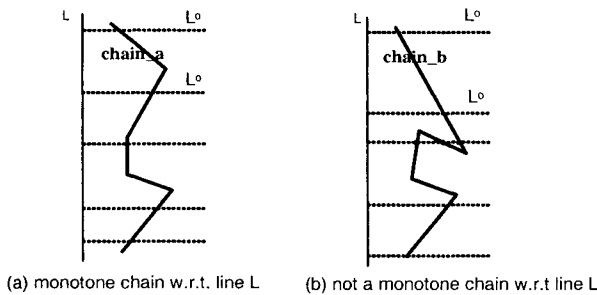


Figure 22.22. A general chain and a monotone chain.

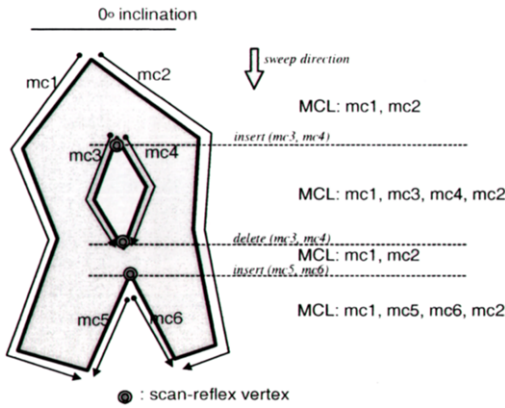


Figure 22.23. Maintaining the MCL (monotone chain list).

linear change in curvature (if the point spacing is uniform). In the literature, a quantity similar to the sum of squares of the right-hand side terms in Equation (22.12) is often used as a global smoothness measure of a digitized curve (Eck and Jaspert [11]). For local straightening, we use the 2nd-difference fairing equation (22.11), while the 4th-difference fairing equation (22.12) is employed for global smoothing.

In practice, the input point sequence may have uneven spacing. Thus, the above fairing expressions have to be normalized with respect to their chord lengths. For this purpose, we will define the following quantities:

$$d_{-2} = |\mathbf{r}_{j-2} - \mathbf{r}_{j-1}|, \quad d_{-1} = |\mathbf{r}_{j-1} - \mathbf{r}_j|, \quad d_{+1} = |\mathbf{r}_{j+1} - \mathbf{r}_j|, \quad \text{and} \quad d_{+2} = |\mathbf{r}_{j+2} - \mathbf{r}_{j+1}|. \quad (22.13)$$

Then, from Equations (22.11) and (22.12), the following ‘normalized’ fairing equations may be obtained:

$$\hat{\mathbf{r}}_j = \left(\frac{d_{-1}}{d_0} \mathbf{r}_{j+1} + \frac{d_{+1}}{d_0} \mathbf{r}_{j-1} \right) / 2 \equiv \mathbf{m} \quad (22.14)$$

and

$$\hat{\mathbf{r}}_j = \mathbf{m} + \left(\frac{d_0}{d_{-2}} (\mathbf{r}_{j-1} - \mathbf{r}_{j-2}) + \frac{d_0}{d_{+2}} (\mathbf{r}_{j+1} - \mathbf{r}_{j+2}) \right) / 6, \quad (22.15)$$

where $d_0 = (d_{-1} + d_{+1})/2$. Note that Equations (22.14) and (22.15) become equivalent to Equations (22.11) and (22.12) when we have $d_{-2} = d_{-1} = d_{+1} = d_{+2}$.

In actual fairing, the ‘faired’ position $\tilde{\mathbf{r}}_j$ is usually determined by taking a linear combination of the ideal position $\hat{\mathbf{r}}_j$ and the input point \mathbf{r}_j , as follows:

$$\tilde{\mathbf{r}}_j = \hat{\mathbf{r}}_j + \Phi \cdot (\mathbf{r}_j - \hat{\mathbf{r}}_j) \quad \text{subject to} \quad |\tilde{\mathbf{r}}_j - \mathbf{r}_j| \leq \tau, \quad (22.16)$$

where $\Phi \in [0, 1]$ is a **damping factor** and τ is a **fairing tolerance**. The blending operation (22.16) is often called a ‘damping correction’. In practice, a damping factor of 0.4 to 0.6 is commonly used.

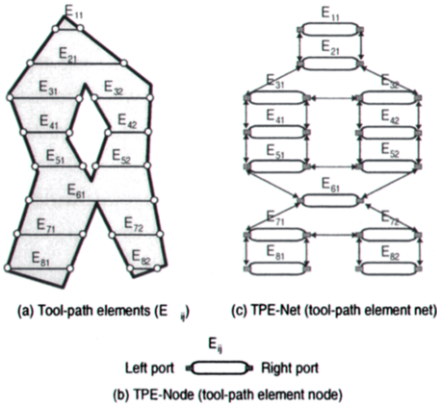


Figure 22.24. The TPE-net and a TPE-node.

Domain-coordinates fairing

To fair the domain-coordinate points $\{\mathbf{p}_j\}$, the ideal position $\hat{\mathbf{p}}_j$ can be obtained by applying the normalized fairing equations in (22.14) and (22.15). Thus, we have:

$$\hat{\mathbf{p}}_j = \left(\frac{d_{-1}}{d_0} \mathbf{p}_{j+1} + \frac{d_{+1}}{d_0} \mathbf{p}_{j-1} \right) / 2 \equiv \mathbf{m} \tag{22.17}$$

and

$$\hat{\mathbf{p}}_j = \mathbf{m} + \left(\frac{d_0}{d_{-2}} (\mathbf{p}_{j-1} - \mathbf{p}_{j-2}) + \frac{d_0}{d_{+2}} (\mathbf{p}_{j+1} - \mathbf{p}_{j+2}) \right) / 6, \tag{22.18}$$

where $d_0 = (d_{-1} + d_{+1})/2$ and $\{d_i\}$ are defined as follows:

$$d_{-2} = |\mathbf{p}_{j-2} - \mathbf{p}_{j-1}|, \quad d_{-1} = |\mathbf{p}_{j-1} - \mathbf{p}_j|, \quad d_{+1} = |\mathbf{p}_{j+1} - \mathbf{p}_j| \quad \text{and} \quad d_{+2} = |\mathbf{p}_{j+2} - \mathbf{p}_{j+1}|.$$

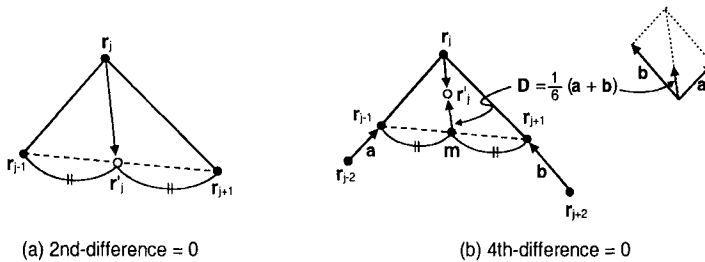


Figure 22.25. The physical meaning of difference fairing.

The 2nd-difference fairing equation (22.17) is used for local straightening of the domain coordinates of the pencil-points, and the 4th-difference fairing (22.18) is used for global smoothing. In Equations (22.17) and (22.18), the damping-correction operation (22.16) is applied to the ideal position \mathbf{p}'_j and the input-point \mathbf{p}_j in order to obtain a corrected position \mathbf{p}''_j . That yields:

$$\tilde{\mathbf{p}}_j = \hat{\mathbf{p}}_j + \Phi_d \cdot (\mathbf{p}_j - \hat{\mathbf{p}}_j) \quad \text{subject to } |\tilde{\mathbf{p}}_j - \mathbf{p}_j| \leq \tau_d, \tag{22.19}$$

where Φ_d and τ_d are the damping factor and fairing tolerance for the domain fairing.

Height-coordinates fairing

A unique feature of the process of fairing the height coordinates is that 1) only height values $\{z_j\}$ in $\{\mathbf{q}_j = (s_j, z_j)\}$ are allowed to move and 2) the domain chord-lengths $\{s_j\}$ are used in normalizing the fairing expressions. We define the following quantities:

$$d_{-2} = |s_{j-2} - s_{j-1}|, \quad d_{-1} = |s_{j-1} - s_j|, \quad d_{+1} = |s_{j+1} - s_j| \quad \text{and} \quad d_{+2} = |s_{j+2} - s_{j+1}|.$$

Then, from Equations (22.14) and (22.15), we obtain the normalized fairing expressions for $\{\mathbf{q}_j = (s_j, z_j)\}$ given below:

$$\hat{z}_j = \left(\frac{d_{-1}}{d_0} z_{j+1} + \frac{d_{+1}}{d_0} z_{j-1} \right) / 2 \equiv m \tag{22.20}$$

and

$$\hat{z}_j = m + \left(\frac{d_0}{d_{-2}} (z_{j-1} - z_{j-2}) + \frac{d_0}{d_{+2}} (z_{j+1} - z_{j+2}) \right) / 6, \tag{22.21}$$

where $d_0 = (d_{-1} + d_{+1})/2$.

The ‘faired’ position \tilde{z}_j is obtained from the following correction operation:

$$\tilde{z}_j = z_j + \Phi_h \cdot (\hat{z}_j - z_j) \quad \text{subject to } |\tilde{z}_j - z_j| \leq \tau_h, \tag{22.22}$$

where Φ_h is the damping factor and τ_h is the fairing tolerance for the height-fairing.

Overall fairing procedure

Based on the results presented so far, the overall PS-curve fairing procedure may be summarized as follows:

- Step 1.* Local ‘saw-tooth pattern’ straightening of $\{\mathbf{p}_j\}$ by employing Equations (22.17) and (22.19).
- Step 2.* Global smoothing of $\{\mathbf{p}_j\}$ using Equations (22.18) and (22.19).
- Step 3.* Local ‘saw-tooth pattern’ straightening of $\{z_j\}$ utilizing Equations (22.20) and (22.22).
- Step 4.* Global smoothing of $\{z_j\}$ using Equations (22.21) and (22.22).

At each step, the correction operation from Equation (22.19) or (22.22) is applied repeatedly. In each iteration, only the data points that have a ‘local maximum’ deviation are corrected, and the iteration is terminated when no significant improvement is observed.

For this purpose, we define the deviation ν_j of a data point (from the ideal position) to be

$$\nu_j = \frac{|\hat{\mathbf{p}}_j - \mathbf{p}_j|}{|\hat{\mathbf{p}}_{j+1} - \mathbf{p}_{j-1}|} \quad \text{or} \quad \nu_j = \frac{|\hat{z}_j - z_j|}{|s_{j+1} - s_{j-1}|}. \quad (22.23)$$

Then, in a local straightening, the j -th data point may be corrected only when $\nu_j \geq \max(\nu_{j-1}, \nu_{j+1})$ is satisfied. In a global smoothing, the j -th data point is corrected when we have $\nu_j \geq \max(\nu_{j-2}, \nu_{j-1}, \nu_{j+1}, \nu_{j+2})$.

22.5.5. Collision detection algorithms

In the C-space approach described in Section 22.4.1.2, ‘rapid-move’ collisions can be avoided by preventing rapid-moves (G00 NC code blocks) from entering the machining C-space volume V_M . This is equivalent to confining the rapid moves to the space above the preform CL-surface S_p .

The C-space method allows a straightforward mechanism for preventing collisions during cutting moves (G01 or G02/03 NC code blocks) as well. As shown in Figure 22.26-a, an endmill assembly consists of four elements: cutting-edge, dead-center, shank and holder. Cutting actions take place only at the cutting edge. The dead-center is the center region of the base of the endmill, which has no cutting capability. If a non-cutting element is in contact with the workpiece during a machining operation, a ‘cutting-move’ collision occurs. Thus, there are three types of cutting-move collision: dead-center collision, shank-collision and holder-collision.

It should be observed that a dead-center collision can only occur during downward milling, while a shank collision may occur during upward milling. On the other hand, a holder collision can occur in both downward and upward milling. Now, as depicted in Figure 22.26-b, we define an ‘inverse’ tool (IT) for each element of the endmill-assembly: a cutting-edge IT, a dead-center IT, a shank IT and a holder IT. Note that we use the same reference point C in all the inverse tools. The next step is to generate ITO (inverse tool offset) surfaces as follows:

1. The cutting-edge ITO surface is the design CL-surface (S_D) for the cutting-edge IT.
2. The dead-center ITO surface is the preform CL-surface (S_P) for the dead-center IT.
3. The shank ITO surface is the preform CL-surface (S_P) for the shank IT.
4. The holder ITO surface is the preform CL-surface (S_P) for the holder IT.

Then a necessary condition for a holder collision is that “there exists a region where the holder ITO surface is higher than the cutting-edge ITO surface”, as indicated in Figure 22.26-c. Similarly, a necessary condition for a shank collision (dead-center collision) may be expressed as “there exists a region where the shank ITO surface (dead-center ITO surface) is higher than the cutting-edge ITO surface when the endmill is moving upward (downward)”.

The C-space-based collision-detection procedure introduced in this section is not limited to roughing. It is applicable to any type of 3-axis NC machining.

22.6. CONCLUSION

Sculptured surface machining (SSM) is a means to realize sculptured surfaces created by engineering designers and often becomes a vital part of other non-machining processes,

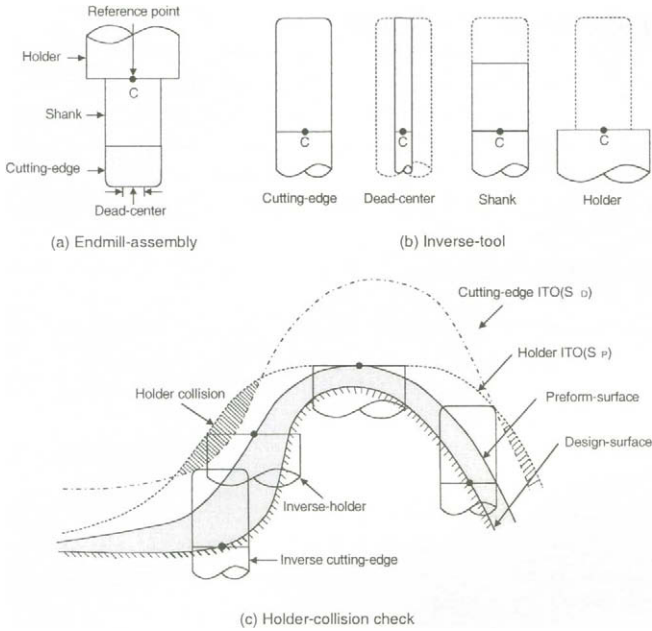


Figure 22.26. Collision detection.

such as sheet-metal stamping and plastic injection molding. CAGD people should know about the SSM process because 1) geometric problems of SSM are closely related to CAGD and 2) the seamless connection of product design and the SSM process is a key factor in improving product quality while reducing development time.

This chapter has introduced the SSM (sculptured surface machining) process, including the UMO (unit machining operation) and identified the geometric issues that occur in the information processing stages of SSM, which are as follows: 1) interference handling, 2) CL-surface construction for the C-space approach, 3) 2D PS(point sequence)-curve off-setting, 4) area scan algorithms, 5) PS-curve fairing and 6) collision detection algorithms.

REFERENCES

1. T. Altan et al. Advanced techniques for die and mold manufacturing. *Annals of the CIRP*, 42(2):707-716, 1993.
2. S. Austin, R.B. Jerard, and S. Drysdale. Comparison of discretization algorithms for NURBS surfaces with application to numerically controlled machining. *Computer-Aided Design*, 29(1):71-83, 1997.
3. B.K. Choi et al. Triangulation of acattered data in 3D space. *Computer-Aided Design*, 20(5):239-247, 1988b.
4. B.K. Choi and C.S. Jun. Ball-end cutter interference avoidance in NC machining of

- sculptured surfaces. *Computer-Aided Design*, 21(6):371–378, 1989.
5. B.K. Choi. *Surface Modeling for CAD/CAM*. Elsevier, Amsterdam, 1991.
 6. B.K. Choi et al. Development of a 9-axis marine propeller machining system. Final Report submitted to Hyundai Heavy Industries Ltd, KAIST, Korea (in Korean), 1991.
 7. B.K. Choi, D.H. Kim, and R.B. Jerard. C-space approach to tool-path generation for die and mold machining. *Computer-Aided Design*, 29(9):657–669, 1997.
 8. B.K. Choi and S.C. Park. A pair-wise offset algorithm for 2D point-sequence curve. *Computer-Aided Design*, 31(12):735–745, 1998.
 9. B.K. Choi and R.B. Jerard. *Sculptured Surface Machining*. Kluwer Academic Publishers, 1998.
 10. J.P. Duncan and S.G. Mair. *Sculptured Surfaces in Engineering and Medicine*. Cambridge University Press, Cambridge - London - New York, 1983.
 11. M. Eck and R. Jaspers. Automatic fairing of point sets. In N. Sapidis, editor, *Designing Fair Curves and Surfaces*, pages 44–60, SIAM, 1994.
 12. P. Fallbohmer et al. Survey of the U.S. die and mold manufacturing industry. *ERC/NSM-D-95-41*, The Ohio State University, U.S.A, 1995.
 13. C.G. Jensen and D.C. Anderson. Accurate tool placement and orientation for finish surface machining. *Concurrent Engineering*, pages 127–145, PED 59, ASME, 1992.
 14. S. Marshall and J.G. Griffiths. A new cutter-path topology for milling machines. *Computer-Aided Design*, 26(3):204–214, 1994.
 15. F. Mason. 55 for high-productivity airfoil milling. *American Machinist*, pages 37–39, 1991.
 16. R.M. Murray et al. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.
 17. S.C. Park and B.K. Choi. Tool-path planning for direction-parallel area milling. *Computer-Aided Design*, 32(1):17–25, 2000.
 18. T. Saito and T. Takaashi. NC Machining with G-buffer method. *Computer Graphics*, 25(4):207–216, July 1991.
 19. H. Schulz and S. Hock. High-speed milling of dies and moulds - cutting conditions and technology. *Annals of CIRP*, 44(1):35–38, 1995.
 20. X. Sheng and B.E. Hirsch. Triangulation of trimmed surfaces in parametric space. *Computer-Aided Design*, 24(8):437–444, 1992.
 21. D.J. Suresh and D.C.H. Yang. Constant scallop height machining of free form surfaces. *Journal of Engineering for Industry*, 116:253–259, 1994.
 22. Y. Takeuchi et al. 5-axis control machining based on solid model. *J. Precision Eng.*, 56(11):111–116 (in Japanese), 1990.
 23. W. Tiller and E.G. Hansen. Offset of two-dimensional profiles. *IEEE Computer Graphics and Applications*, 4(9):36–46, September 1984.

Chapter 23

Cyclides

Wendelin Degen

23.1. INTRODUCTION

Cyclides were detected and first studied by Ch. Dupin (1784–1873) in [22] and are since called “Dupin cyclides”. At that time differential geometry was in a very early stage: Only some investigations of L. Euler (1707–1783) and the fundamental work of G. Monge (1746–1818) “L’Application de l’analyse à la géométrie” proceeded but the famous treatise of C.F. Gauss (1777–1855) “Disquisitiones generales circa superficies curvas” appeared only a few years later.

In parallel, after projective geometry was founded by Poncelet (1788–1867), research on algebraic curves and surfaces had a first blossom. Dupin cyclides belong to both of these areas: In differential geometry, they can be defined as surfaces having *constant* main curvatures along the corresponding curvature lines. This property has been widely generalized into the differential geometry of manifolds in higher dimensional spaces up to present (see [13] for instance).

On the other hand, they are algebraic surfaces of fourth or third order ¹ being simultaneously enveloped by two families of spheres. This latter property is characteristic and serves often as an alternative definition. As algebraic surfaces they can be represented by a single polynomial equation of degree up to four; but they proved, in addition, to be *rational*, thus they can be parametrized by quadratic functions of u and v . These two kinds of representations that can be converted into each other confers the Dupin cyclides many advantages, in particular with respect to applications.

It was soon realized that Dupin cyclides of fourth order have the isotropic circle at infinity as its double curve. This gave rise to extensive investigations on algebraic surfaces. Kummer [29] and Casey [11] found interesting generalizations of Dupin cyclides, ² however

¹In addition, there are some cases of second order cyclides (the natural quadrics) that can be considered as “trivial Cyclides”

²Those of Casey are called “generalized cyclides”

these topics must be renounced in the present chapter (see the excellent treatise of Jessop [25] for details).

But there is another intermediate class of surfaces, found by Degen [15–17], they are defined to be the envelope of two families of *general quadrics* such that each of these quadrics is tangent to the surface along a *conic*, with the additional property that these conics build up a *conjugate net* on the surface (what holds also for Dupin cyclides). These surfaces, now called “supercyclides” (originally “double Blutel surfaces”) will be treated in Section 23.3.

Dupin cyclides were recovered for CAGD purposes about 1988; they extended the class of surfaces so far used in geometric modeling (essentially the natural quadrics) considerably. In particular, the virtues of cyclides were realized when serving as blending surfaces. Still much more flexibility is attained by using supercyclides. The long list of papers that since appeared show the importance of cyclides and supercyclides in CAGD.

23.2. THE GEOMETRY OF DUPIN CYCLIDES

23.2.1. Dupin cyclides in classical differential geometry

As usual in local differential geometry, one starts with a regular surface patch in parameter representation $\phi \dots \mathbf{x} : I_1 \times I_2 \rightarrow \mathbb{R}^3$, where the curvature lines are used as iso-parameter lines. (This is possible if the patch has no umbilics; see [10] for basic information on classical differential geometry). They will be denoted by $\mathbf{C}_1(u_0) = \{\mathbf{x}(u, v_0) \mid u \in I_1\}$ and $\mathbf{C}_2(u_0) = \{\mathbf{x}(u_0, v) \mid v \in I_2\}$. We further assume differentiability (at least $\mathbf{x} \in C^3[I_1 \times I_2]$) and regularity $\mathbf{x}_u \wedge \mathbf{x}_v \neq \mathbf{0}$ throughout.

Let $\mathbf{n} : I_1 \times I_2 \rightarrow \mathbb{R}^3$ be the normal vector field ($\mathbf{n} = \mathbf{x}_u \wedge \mathbf{x}_v / \|\mathbf{x}_u \wedge \mathbf{x}_v\|$), then the *formulas of Rodrigues* and the defining property of the Dupin cyclides are expressed by ³

$$\mathbf{x}_u = -r_1 \mathbf{n}_u, \quad \mathbf{x}_v = -r_2 \mathbf{n}_v, \quad r_{1u} = 0, \quad r_{2v} = 0. \tag{23.1}$$

This implies that the *curvature centers*

$$\mathbf{p} = \mathbf{x} + r_2 \mathbf{n}, \quad \mathbf{q} = \mathbf{x} + r_1 \mathbf{n}. \tag{23.2}$$

depend only on *one* parameter, thus describing two curves $\mathbf{P} \dots \mathbf{p} : I_1 \rightarrow \mathbb{R}^3$, $\mathbf{Q} \dots \mathbf{q} : I_2 \rightarrow \mathbb{R}^3$. Furthermore we consider the *curvature spheres* $\mathbf{S}_1(u)$ ($\mathbf{S}_2(v)$) having $\mathbf{p}(u)$ ($\mathbf{q}(v)$) as mid point and $r_2(u)$ (respectively $r_1(v)$) as radius ⁴ and conclude that they also depend only on one parameter. Obviously, the curvature spheres $\mathbf{S}_1(v)$ and $\mathbf{S}_2(u)$ touch each other at the surface point $\mathbf{x}(u, v)$; but by $\mathbf{n} \sim \mathbf{p} - \mathbf{q}$ both are tangent also to Φ at that point. Thus both families of curvature spheres envelope the cyclide; if one of them moves through its family and another one of the other family is kept fixed, the points of contact generate the corresponding curvature line. But since the *characteristics* (defined as $\lim_{h \rightarrow 0} \{\mathbf{S}(u_0) \cap \mathbf{S}(u_0 + h)\}$) of a one parameter family of spheres are *circles*, so are the curvature lines of a Dupin cyclide. Furthermore, as $[\mathbf{p}(u), \mathbf{p}'(u)]$ is the rotational axis of

³We wrote here these formulas in an inverse manner using the main curvature *radii* $r_i = 1/\kappa_i$ instead the curvatures κ_i ($i = 1, 2$) itself, assuming that these quantities do not vanish anywhere. Thus the *trivial cyclides*, the right circular cones and cylinders, as well as the sphere itself, are excluded in the sequel.

⁴The curvature spheres are thought to be *oriented*; a negative radius means that the normal vector points into the interior.

the enveloping cone $C_2^*(u_0)$ its apex $\mathbf{a}(u_0)$ does not depend on v and thus all the planes of $C_1(v)$ ($v \in I_1$) contain the tangent $a_1 := [\mathbf{a}(u), \mathbf{a}'(u)]$. Since these planes do not depend on u , this line a_1 is *fixed in space*. Hence these planes belong to a *pencil with axis a_1* . As the analogous properties hold also for the other family, we can summarize:

Theorem 1 (a) *The focal surfaces of the normals of a Dupin cyclide* ⁵ *degenerate into two curves \mathbf{P} and \mathbf{Q} .* (b) *The curvature spheres of a Dupin cyclide consist of two one parameter families \mathcal{S}_1 and \mathcal{S}_1 .* (c) *Each curvature sphere envelopes the cyclide along a curvature line (more precisely $\mathcal{S}_2(v_0)$ is tangent to Φ along the curvature line $C_1(v_0)$ and, analogously, $\mathcal{S}_1(u_0)$ is so along $C_2(u_0)$).* (d) *All the curvature lines are circles.* ⁶ *Their planes belong to two pencils (more precisely: the planes of $C_1(v_0)$ belong to the pencil with axis a_1 , those of $C_2(u_0)$ belong that one with axis a_2 .* (e) *The apexes $\mathbf{a}(u)$ of the circumscribed cones along $C_2(u)$ lie on the axis a_1 and those of the other family, $\mathbf{b}(u)$, lie on a_2 .* (f) *The mid point curves \mathbf{P} and \mathbf{Q} are planar curves whose planes \mathbf{E} and \mathbf{F} contain the axes a_1 and a_2 respectively.*

The last property (f) is implied by the fact that the tangent $[\mathbf{p}(u), \mathbf{p}'(u)]$ meets the axis a_1 and analogously for the other family. — Finally, simple geometric reasoning leads to

Theorem 2 (a) *The mid point curves \mathbf{P} and \mathbf{Q} of the two families of enveloping spheres are conics lying in two planes \mathbf{E} and \mathbf{F} which are orthogonal to each other and symmetry planes of the cyclide.* (b) *both axes a_1 and a_2 are orthogonal to the intersection line $\mathbf{l} := \mathbf{E} \cup \mathbf{F}$.* (c) *One of the curves \mathbf{P} or \mathbf{Q} , say \mathbf{Q} , may degenerate into a straight line; then \mathbf{Q} must coincide with a_2 and a_1 is in \mathbf{E} at infinity.*

Furthermore, from (23.2) one can take

$$\mathbf{x}(u, v) = \frac{1}{r_1(v) - r_2(u)} (r_1(v)\mathbf{p}(u) - r_2(u)\mathbf{q}(v)). \tag{23.3}$$

Thus the surface is determined once the curves \mathbf{P} and \mathbf{Q} as well as the radius functions r_1 and r_2 are known.

As the radius functions can be replaced by $\bar{r}_1(v) = r_1(v) + c$ and $\bar{r}_2(u) = r_2(u) + c$ (while the normals are maintained) one gets:

Theorem 3 *The offset surfaces of a Dupin cyclide are again Dupin cyclides; they share the same normals with the original cyclide and have the radius functions with an arbitrary constant c added* ⁷.

It remains to determine the exact possibilities of the different kinds of conics that can be combined to a pair of curves \mathbf{P} and \mathbf{Q} for a Dupin cyclide. This will be done in the next section.

⁵A regular patch with non-vanishing main curvatures and without umbilics (i.e. $r_i \neq 0$, ($i = 1, 2$) $r_1 \neq r_2$) is assumed.

⁶Later on, when a Dupin cyclide will be considered *globally*, we will allow spheres to have radius $r = 0$ (leading to a *singular point*) or to have radius $r = \infty$ (then being a plane). In the latter case, the curvature line degenerates into a straight line.

⁷However, new singularities occur when \bar{r}_1 or \bar{r}_2 has a zero.

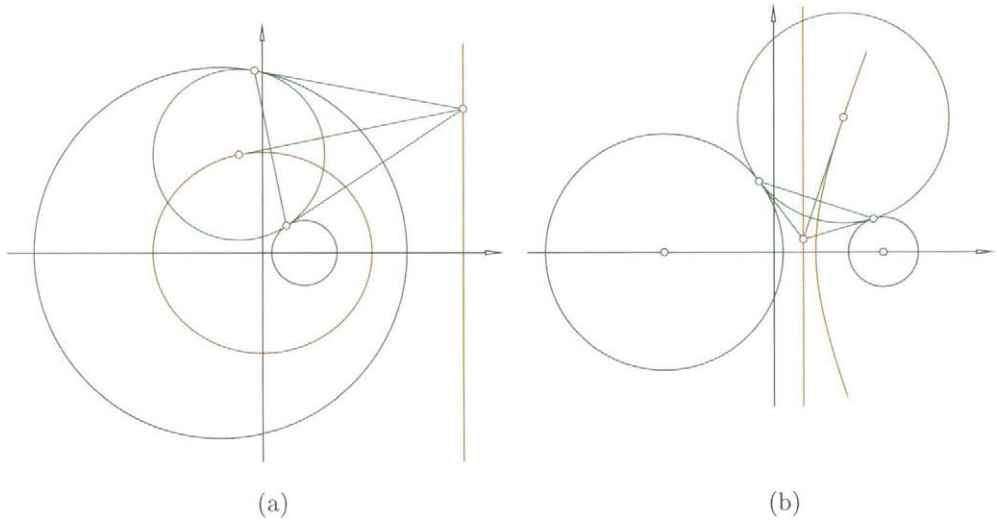


Figure 23.1. (a) A generating sphere of a general Dupin cyclide intersected with the first symmetry plane; and (b) same as (a) but in the second symmetry plane.

23.2.2. The three main types of Dupin cyclides and their parameter representations

Theorem 4 *There are the following three types of regular Dupin cyclides*

I Tori: \mathbf{P} is a circle, the surface is rotationally symmetric, \mathbf{Q} degenerates into a line being simultaneously the second axis a_2 and the axis of rotation. The first axis a_1 is in the plane \mathbf{E} at infinity.

II General Dupin cyclides: \mathbf{P} is an ellipse in \mathbf{E} and \mathbf{Q} is a hyperbola in \mathbf{F} , orthogonal to \mathbf{E} . These conics share their vertices and foci but with reversed roles. All these points lie on $\mathbf{l} = \mathbf{E} \cap \mathbf{F}$. The axes a_1 and a_2 lie in \mathbf{E} , \mathbf{F} respectively and are orthogonal to \mathbf{l} .

III Parabolic Dupin cyclides: \mathbf{P} and \mathbf{Q} are both parabolas having \mathbf{l} as their common axis. They also share their vertex and focus in reversed roles.

Proof: The exceptional case of a torus is already contained in Theorem 3. In all the other cases, \mathbf{P} and \mathbf{Q} are true real conics. Since \mathbf{E} and \mathbf{F} are symmetry planes, the vertices lie on \mathbf{l} . In the plane \mathbf{E} there are the intersection circles $\mathbf{D}_1(u) = \mathbf{S}_1(u) \cup \mathbf{E}$ of the first family of spheres \mathbf{S}_1 , having their mid points at $\mathbf{p}(u)$. From the other family \mathbf{S}_2 there is at least one profile circle $\mathbf{D}_2(v_0)$ having its mid point at (one of) the vertex $\mathbf{q}(v_0)$ on \mathbf{l} . Thus, all the circles $\mathbf{D}_1(u)$ touch $\mathbf{D}_2(v_0)$ at $\mathbf{x}(u, v_0)$. Furthermore, the

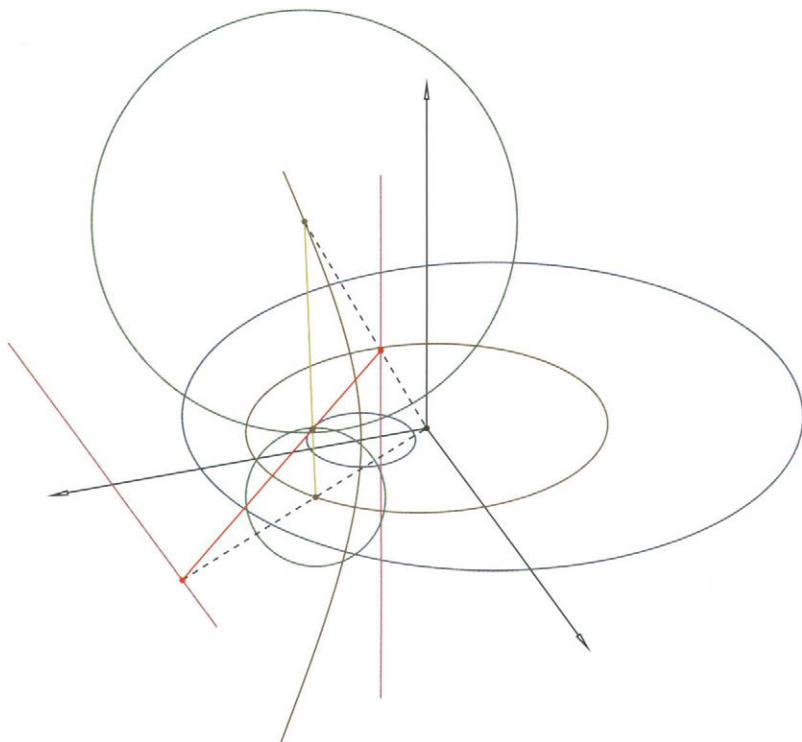


Figure 23.2. The two enveloping spheres at a point of a Dupin cyclide and the mid point curves in the symmetry planes.

tangents to \mathbf{P} at $\mathbf{p}(u)$ and to $\mathbf{D}_2(v_0)$ at $\mathbf{x}(u, v_0) \in \mathbf{E}$ must intersect on the point $\mathbf{a}(u)$ of a_1 .

These conditions can be exploited with methods of elementary geometry in all the three cases when \mathbf{P} is an ellipse, a hyperbola or a parabola. The crucial conclusion is the following: Using a *rational* parametrization in any case, setting $\mathbf{a}(u) = (d, \alpha(u), 0)$ in a suitable coordinate system with \mathbf{l} as x-axis, \mathbf{E} as x-y-plane and \mathbf{F} as x-z-plane one gets for α a *rational* function. This implies that $\omega(u) := \|\mathbf{p}(u) - \mathbf{q}(v_0)\|$ must be rational too. The radicand of $\omega(u)$ turns out to be a biquadratic function and so it must be a complete square. The calculations with coordinates yield in any case that $\mathbf{q}(v_0)$ is a *focal point* of \mathbf{P} .

Elaborating this idea yields for \mathbf{Q} a hyperbola when \mathbf{P} is an ellipse, an ellipse when \mathbf{P} is a hyperbola and a parabola when \mathbf{P} is a parabola. Since in the first two cases only the two curves \mathbf{P} and \mathbf{Q} are permuted, there remain, besides the tori, only the two further cases of the theorem. \square

The *parameter representations* of the Dupin cyclides are obtained by (23.3). We use rational parameters for \mathbf{P} and \mathbf{Q} in all cases (They can, if wanted, easily be converted

into trigonometric forms by setting $\cos(\phi) = \frac{1-u^2}{1+u^2}$ and $\sin(\phi) = \frac{2u}{1+u^2}$ and likewise for v .) Then the radius functions are obtained by the method described above. One obtains the following results:

Case I. Tori:

Mid point curves:

$$\mathbf{p}(u) = R \left(\frac{1-u^2}{1+u^2}, \frac{2u}{1+u^2}, 0 \right), \quad \mathbf{q}(v) = R \left(0, 0, \frac{2v}{1-v^2} \right). \quad (23.4)$$

Radius functions:

$$r_2 = r = \text{const}, \quad r_1(v) = r - R \frac{1+v^2}{1-v^2}. \quad (23.5)$$

Parameter representation of the cyclide:

$$\mathbf{x}(u, v) = \left(R - r \frac{1-v^2}{1+v^2} \right) \left(\frac{1-u^2}{1+u^2}, \frac{2u}{1+u^2}, 0 \right) + r \left(0, 0, \frac{2v}{1-v^2} \right). \quad (23.6)$$

Depending on the ratio of R and r , there are three subcases of tori:

1. $r > R$: All the spheres $\mathbf{S}_1(u)$ intersect the second axis a_2 in a pair real, different points where the radius function r_1 has zeros and the cyclide has *singular points* (like the apex of a cone); it is called a “spindle cyclide”.
2. $r = R$: The axis a_1 is *tangent* to all the spheres $\mathbf{S}_1(u)$ of \mathbf{S}_1 . The surface has a singularity (a sharp double spike) at the point of contact and is called “limit torus”.
3. $r < R$: The spheres $\mathbf{S}_1(u)$ don't intersect. The surface has no singularities and is ring-shaped.

Case II. General Dupin cyclides

Mid point curves:

$$\mathbf{p}(u) = \left(\frac{1-u^2}{1+u^2}a, \frac{2u}{1+u^2}b, 0 \right), \quad \mathbf{q}(v) = \left(\frac{1+v^2}{1-v^2}f, 0, \frac{2v}{1-v^2}b \right) \quad (23.7)$$

with $f^2 = a^2 - b^2$ saying that $\mathbf{q}(v_0)$ is a focal point⁸ of \mathbf{P} .

Radius functions:

$$r_2(u) = c - \frac{1-u^2}{1+u^2}f, \quad r_1(v) = c - \frac{1+v^2}{1-v^2}a. \quad (23.8)$$

Parameter representation of the cyclide:

$$\mathbf{x}(u, v) = \frac{((1+v^2)fG(u) - (1-u^2)aF(v), -2buF(v), 2bvG(u))}{(1-v^2)G(u) - (1+u^2)F(v)} \quad \text{where} \quad (23.9)$$

$$G(u) = (1+u^2)c - (1-u^2)f, \quad F(v) = (1-v^2)c - (1+v^2)a. \quad (23.10)$$

⁸We keep the denotation f for the abscissa of the focal point though usually denoted by e (the “eccentricity” of the ellipse).

Abcissae of the axes:

$$a_1 : x = \frac{ac}{f}, z = 0; \quad a_2 : x = \frac{cf}{a}, y = 0. \tag{23.11}$$

From these formulas one can take that there are three subcases for each of the positions of the axes with respect to **P** and **Q** respectively. However, since both axes depend on the same parameter c there are only the following five subcases:

1. $c < f$: The axis a_1 intersects **P** in a pair of real, different points where the radius function r_2 has zeros and the cyclide has *singular points* (like the apex of a cone); it is called a “spindle cyclide”.
2. $c = f$: The axis a_1 is *tangent* to **P** at the point $(a, 0, 0)$ and so are all the spheres of \mathcal{S}_1 . The surface has a singularity (a sharp double spike) at that point.
3. $f < c < a$: Both axes don't intersect **P** or **Q**. The surface has no singularities and is ring-shaped. The second axis a_2 passes through the hole while the first one lies outside the surface.
4. $c = a$: The axis a_2 is *tangent* to **Q** at the point $(f, 0, 0)$ and so are all the spheres of \mathcal{S}_2 . The surface has a singularity (a sharp double spike) at that point.
5. $c > a$: The axis a_2 intersects **Q** in a pair of real, different points where the radius function r_1 has zeros and the cyclide has *singular points* (like the apex of a cone); it is called a “horn cyclide”.

Case III. Parabolic Dupin cyclides

Mid point curves:

$$\mathbf{p}(u) = (u^2 - 1/2, 2u, 0) f, \quad \mathbf{q}(v) = (1/2 - v^2, 0, 2v) f \tag{23.12}$$

(where the mid point between the vertex and focus serves as origin).

Radius functions:

$$r_2(u) = c + (u^2 + 1/2)f, \quad r_1(v) = c - (v^2 + 1/2)f. \tag{23.13}$$

with an arbitrary constant c . One can assume $c \geq 0$ since otherwise the two parabolas can be permuted.

Abcissae of the axes:

$$a_1 : x = -f - c, z = 0; \quad a_2 : x = +f + c, y = 0. \tag{23.14}$$

Parameter representation of the cyclide:

$$\mathbf{x}(u, v) = \frac{(f(u^2 - v^2) - c(u^2 + v^2 - 1), u((2v^2 + 1)f - 2c), v(f(2u^2 + 1) + 2c))}{1 + u^2 + v^2}. \tag{23.15}$$

One derives from this representation:

Theorem 5 *Parabolic Dupin cyclides contain both their axes as degenerated curvature lines $C_1(\infty)$ and $C_2(\infty)$ respectively.*

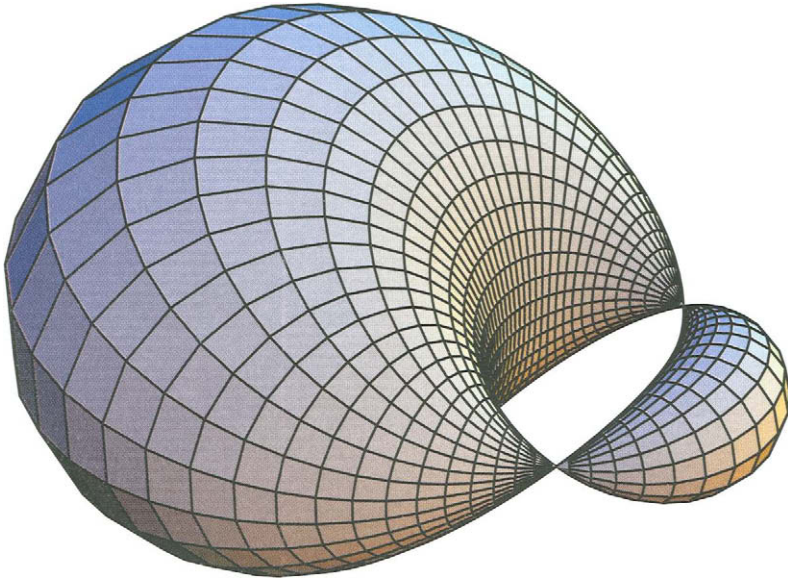


Figure 23.3. Horn cyclide.

This is an essential difference to both the other main classes: In general, a certain plane of a curvature circle intersects the cyclide of class I or II yet in another curvature circle (of the same family) in accordance with the fact that it is an algebraic surface of *fourth* order (see Section 23.2.3). But if the cyclide is parabolic, then the intersection of a plane from the pencils through the axes, the intersection consists of the axis itself and the corresponding curvature circle in accordance with the fact that parabolic cyclides are of the *third* order.

For parabolic cyclides there are also three subcases:

1. $0 < c < f/2$: The cyclide is everywhere regular and has a hole between the two axes.
2. $c = f/2$: The cyclide has a singularity on the second axis at the point $(f/2, 0, 0)$ ($= \mathbf{E} \cap a_2$) which is a sharp double spike.
3. $c > f/2$: The cyclide has two singularities on a_2 (at $a_2 \cap \mathbf{Q}$).

23.2.3. Implicit equations

By eliminating the parameters in the representations (23.6), (23.9), (23.15), one gets the following implicit representations of the corresponding cyclides.

Case I. Tori:

$$\Phi_I \dots (x^2 + y^2 + z^2 + R^2 - r^2)^2 - 4R^2(x^2 + y^2) = 0. \tag{23.16}$$

So the tori are *algebraic surfaces of fourth order*. The terms of fourth order being $(x^2 + y^2 + z^2)^2$ confirms that the isotropic circle at infinity is the double curve of Φ_I .

Case II. General Dupin cyclides

The implicit equation of general Dupin cyclides is:

$$\Phi_{II} \dots (x^2 + y^2 + z^2 + a^2 - f^2 - c^2)^2 - 4(ax - cf)^2 - 4y^2(a^2 - f^2). \tag{23.17}$$

So the general Dupin cyclides are, like the tori, *algebraic surfaces of fourth order*. The terms of fourth order are the same as for tori; this confirms that the isotropic circle at infinity is the double curve also for the general Dupin cyclides.

One can see that this equation carries over to (23.16) for $f = 0$ and with $R = a, r = c$. So a torus can be considered as a limit case of a general Dupin cyclide (the abscissa of a_1 tends to ∞ and that of a_2 to 0 in accordance with Theorem 4, I).

Case III. Parabolic Dupin cyclides

For parabolic Dupin cyclides parameter elimination yields

$$\Phi_{III} \dots (x + c)(x^2 + y^2 + z^2) + (y^2 - z^2)f - (f^2 + c^2)x + (f^2 - c^2)c. \tag{23.18}$$

So the parabolic Dupin cyclides are algebraic surfaces of *third order* and the intersection with the plane at infinity splits into the isotropic circle and the straight line $x = 0$.

One realizes that the two axes $a_1 \dots x = -c - f, z = 0$ and $a_2 \dots x = -c + f, y = 0$ are contained in Φ_{III} . So the planes of the two pencils with those axes intersect the surface in that axis and one of the generating circles $C_1(v)$ respectively $C_2(u)$.

23.3. SUPERCYCLIDES

23.3.1. Curves and surfaces in the projective space

In this Section we use *projective geometry* throughout. This has many advantages in theory as well as in applications: There is no need to distinguish between intersecting and parallel lines and planes, or between cones and cylinders and all rational parametrizations can be converted into polynomial ones simply by a renormalization. (For basic information see [8], Ch. V). In particular, points are represented by vectors $\mathbf{p} \in \mathbb{R}^4 \setminus \{(0, 0, 0, 0)\}$ that can arbitrarily *renormalized* by $\bar{\mathbf{p}} = \rho \mathbf{p}$ with $\rho \in \mathbb{R} \setminus \{0\}$.

Curves and surfaces are represented by vector-valued differentiable *functions* $\mathbf{x} : \mathbf{D} \rightarrow \mathbb{R}^4$ where \mathbf{D} is an real interval for curves and an open, connected domain of \mathbb{R}^2 for surfaces. In both cases, the representation can be renormalized (without changing the geometric object) by an arbitrary real-valued differentiable *function* $\rho : \mathbf{D} \rightarrow \mathbb{R} \setminus \{0\}$.

23.3.2. Basic properties of supercyclides

Now we turn to supercyclides. However we must renounce all the theoretical results leading to the properties and the representation we will describe in the sequel (see [15] - [17] and [5] for the details). Furthermore, there is a slight difference between the notions of a “double Blutel surface” and a “supercyclide”. However, in order to allow a unified

treatment, we exclude supercyclides with intersecting axes (see Theorem 6) but include those of third order (as in the case of Dupin cyclides); nevertheless we maintain the term “supercyclide” for shortness.

Furthermore we emphasize that Dupin cyclides are special supercyclides with additional properties coming from the Euclidean structure of the space. So supercyclides share many properties with Dupin cyclides and it turned out that, indeed, most of them are *complex projective transforms* of Dupin cyclides [5].

The theory leads to the final result that the supercyclides (in the sense described above) have a *common parameter representation* of the following type

$$\phi \dots \mathbf{x}(u, v) = \mathbf{p}(u) + \mathbf{q}(v) \quad (23.19)$$

where $\mathbf{p}(u)$ and $\mathbf{q}(v)$ are defined by

$$\mathbf{p}(u) := \frac{A(u)\mathbf{a} + B(u)\mathbf{b}}{F(u)}, \quad \mathbf{q}(v) := \frac{C(v)\mathbf{c} + D(v)\mathbf{d}}{G(v)}. \quad (23.20)$$

Furthermore, \mathbf{a} , \mathbf{b} are base points for the first axis a_1 and \mathbf{c} , \mathbf{d} are base points for the second axis a_2 (all the four vectors together build a basis of \mathbb{R}^4 since the axes are assumed to be skew). In addition, it is assumed that A, B, F (C, D, G) are linearly independent quadratic functions of u (respectively of v).

Surfaces with a representation of the form (23.19) are called *projective translation surface* (see [20], compare with (23.3)). Furthermore, renormalizing with F yields

$$\bar{\mathbf{x}}(u, v) := F(u)\mathbf{x}(u, v) = A(u)\mathbf{a} + B(u)\mathbf{b} + F(u)\mathbf{q}(v). \quad (23.21)$$

and a similar equation is obtained by renormalizing with $G(v)$. This shows:

Theorem 6 (a) *The iso-parameter lines $\mathbf{C}_1(v_0) : v = v_0$ and $\mathbf{C}_2(u_0) : v = v_0$ are non-degenerated conics.* (b) *The planes $\mathbf{E}_1(v_0)$ of $\mathbf{C}_1(v_0)$ contain the first axis a_1 (so belonging to a pencil).* (c) *The planes $\mathbf{E}_2(u_0)$ of $\mathbf{C}_2(u_0)$ contain the second axis a_2 (so belonging to a pencil).*

More precisely, one obtains

$$\mathbf{E}_1(v_0) = \mathbf{a} \wedge \mathbf{b} \wedge \mathbf{q}(v_0), \quad \mathbf{E}_2(u_0) = \mathbf{c} \wedge \mathbf{d} \wedge \mathbf{p}(u_0). \quad (23.22)$$

Since the parameter $\lambda = D(v)/C(v)$ controls the position of $\mathbf{E}_1(v)$ within the pencil (or the point $\mathbf{q}(v)$ on the second axis a_2), we conclude: *Every plane \mathbf{E}_λ of the pencil through a_1 intersects the surface in a pair of conics $\mathbf{C}_1(v_1)$, $\mathbf{C}_1(v_2)$ where v_1, v_2 are the roots of the equation $\lambda C(v) - D(v) = 0$ (possibly being complex or coincident).* This property suggests that the surface has order four.

However the two quadratic functions may have a *common root*: Then the corresponding linear factor cancels at the quotient for λ , so the mapping $v \mapsto \lambda = D(v)/C(v)$ is fractal linear, hence bijective from $\bar{\mathbb{R}}$ onto itself⁹; furthermore, observing (23.19), one realizes that in this case all the conics $\mathbf{C}_2(u)$ pass through the intersection point $\mathbf{p}(u)$ of $\mathbf{E}_2(u)$ with the first axis a_1 , i.e. this line is completely contained in the surface.

Of course, the same considerations can be applied to the second family of conics and their planes. So one has to distinguish between the following three classes of supercyclides:

⁹ $\bar{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$ denotes the closed set of reals

I **General supercyclides:**

None of the pairs A, B and C, D of quadratic functions has a common root.

II **Semi-parabolic supercyclides:**

C, D have a common root, but not A, B .

III **Parabolic supercyclides:**

Both of the pairs A, B and C, D of quadratic functions have a common root.

The semi-parabolic supercyclides have almost been neglected in the literature (only mentioned in [16] and [42]). So we do not draw the reader's attention to that case in the sequel. A second reason for this is that all the Dupin cyclides are contained in the Classes I and III, but not in Class II. (The tori and the general Dupin cyclides in Class I; the parabolic Dupin cyclides in Class III.)

The following result is reported without proof:

Theorem 7 *Supercyclides of the Classes I, II are algebraic surfaces of the fourth order, those of the Class III are of third order.*

Besides the possession of two families of conics, the supercyclides (of all the three classes) have the *dual property*: They are enveloped by two families of quadratic cones $C_1^*(v)$ ($v \in \mathbb{R}$) and $C_2^*(u)$ ($u \in \mathbb{R}$) in such a way that the tangents to the conics $C_1(v)$ (v varying) along the points of a fixed conic $C_2(u_0)$ coincide with the generators of the cone $C_2^*(u_0)$. (This means that the tangent planes of the surface, taken along the conic $C_2(u_0)$, envelope the cone $C_2^*(u_0)$.) The same property holds for the permuted families of conics and cones.

This can easily be seen by differentiating (23.19) with respect to u say:

$$\mathbf{x}(u, v)_u = \frac{d\mathbf{p}(u)}{du} \sim A(u)^* \mathbf{a} + B(u)^* \mathbf{b} \tag{23.23}$$

with $A^* = A'F - AF'$, $B^* = B'F - BF'$ (note that these functions are also *quadratic*). So the point on the right hand side of (23.23) *does not depend on* v and represents the apex of $C_2^*(u)_0$. So one realizes that *the apexes of the enveloping cones $C_2^*(u)$ and $C_1^*(v)$ lie on the first and second axis respectively.* (Thus the axes are self-dual, since the apex of a cone corresponds to the plane of a conic by duality.)

Equation (23.19) contains even more information: Looking into one fixed plane, say $E_1(v_0)$, one can see therein three objects: 1) the conic $C_1(v_0)$, 2) the first axis a_1 and 3) the intersection point $\mathbf{q}(v_0)$ of $E_1(v_0)$ with a_2 . Since their coefficients do not depend on v *all these configurations are projectively equivalent to each other* and the same is true for the other planes $E_2(u_0)$. More precisely, for any pair of planes $E_1(v_1)$ and $E_1(v_2)$, these configurations can even be *projected* onto each other from a third point on the second axis: One takes from (23.19) $\mathbf{x}(u, v_2) = \mathbf{p}(u) + \mathbf{q}(v_2) = \mathbf{x}(u, v_1) + \mathbf{z}$ with $\mathbf{z} = (\mathbf{q}(v_2) - \mathbf{q}(v_1))$ and this proves the assertion, \mathbf{z} being the projection center.

The different positions of these objects to each other lead to the further classification of supercyclides. In particular, the first (second) axis a_1 (a_2) can intersect the conic $C_1(v_0)$ ($C_2(u_0)$) in two different real points leading to two simple singular points on that axis or

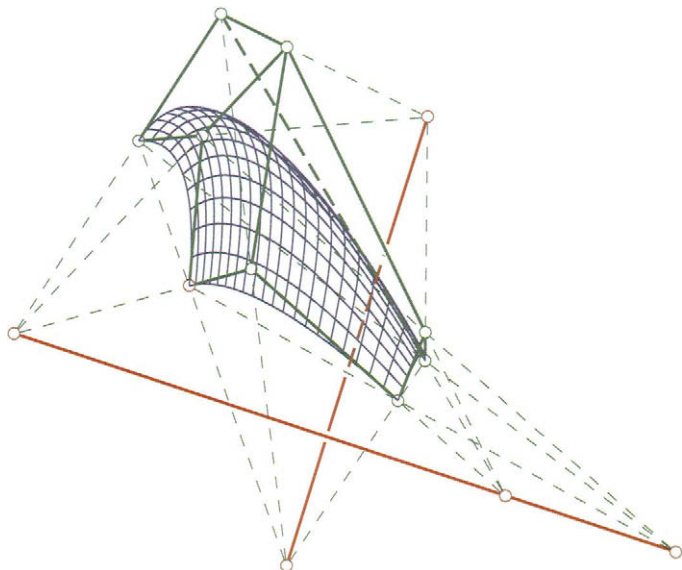


Figure 23.4. A supercyclide with its Bézier control net and axes.

it can be tangent to it leading to a higher singularity, a sharp double spine. (Note that in both cases these intersection points do not depend on v_0 (u_0 resp.)).

Omitting the further details (to be found in [16]), we mention only, that the polarity of $\mathbf{q}(v_0)$ and a_1 has the geometric meaning that the cyclide is a *projective canal surface* (see [9]) and if this holds for both families then it is a quadric. On the other hand, supercyclides of Class I have the following *normal form* (23.19), (23.20) :

$$\begin{aligned} A(u) &= 1, & B(u) &= u^2, & F(u) &= 2u + \sigma_0 + \sigma_2 u^2, \\ C(v) &= 1, & D(v) &= v^2, & G(v) &= 2v + \tau_0 + \tau_2 v^2. \end{aligned} \quad (23.24)$$

Furthermore, one of the coefficients σ_0, σ_2 (τ_0, τ_2 resp.) can be normalized to one if it does not vanish. So there remain two *invariants* (to be interpreted as cross ratios) $\kappa_1 = \sigma_0 \sigma_2$ and $\kappa_2 = \tau_0 \tau_2$ determining the supercyclide:

Theorem 8 *A supercyclide of Class I can be transformed into the normal form (23.24) and is determined by the two projective invariants κ_1, κ_2 up to a projectivity of space.*

23.4. CYCLIDES IN CAGD

23.4.1. Bézier representation of cyclides

As being rational biquadratic surfaces, cyclides can be represented in Bézier form and thus integrated into CAD systems. In Section 23.2, we used the basis functions $T_0(u) =$

$1 + u^2$, $T_1(u) = 1 - u^2$, $T_2(u) = 2u$ which are linearly independent. So they can be converted into the Bernstein basis functions $B_0(u)$, $B_1(u)$, $B_2(u)$ by the matrix

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix}.$$

Applying this to (23.9) for instance immediately yields the Bézier form in homogeneous coordinates. The component x_0 of $\mathbf{x}(u, v)$ is the denominator of the usual Euclidean vector and the coefficients at the Bernstein polynomials are its weights. This can be done by any computer algebra system.

However this would be a very special patch on that Dupin cyclide. To attain the whole flexibility, i.e. an *arbitrary patch* on it, one has to perform a *parameter transformation*

$$\bar{u} = \frac{au + b}{cu + d}, \quad \bar{v} = \frac{ev + f}{gv + h}, \tag{23.25}$$

first, so that the desired boundary curves belong to parameter values $\bar{u} = 0$ and $\bar{u} = 1$ and similarly with v .

A second method consists of constructing the cyclide directly by starting with its boundary curves (circular arcs in the case of Dupin cyclides) and then observing its general geometric properties derived in the previous sections. When doing this, there arise two questions: a) What are the conditions on the control points and weights of a second order rational Bézier curve to be a circular arc? - b) What is a complete set of conditions characterizing a (Dupin) cyclide?

As to question a), given the Bézier representation $\mathbf{x}(t) = \sum_{i=0}^2 B_i(t)\omega_i\mathbf{a}_i / \sum_{i=0}^2 B_i(t)\omega_i$ one defines $\mathbf{p} = (1/2)(\mathbf{a}_0 + \mathbf{a}_2)$, $\alpha = (1/2) \|\mathbf{a}_2 - \mathbf{a}_0\|$, $\mathbf{v}_1 = (\mathbf{a}_2 - \mathbf{a}_0)/\alpha$ and \mathbf{v}_2 so that $\mathbf{v}_1, \mathbf{v}_2$ are a ON basis, then the conditions for a circular arc are

$$\mathbf{a}_1 = \mathbf{p} + \beta\mathbf{v}_1, \quad (\alpha^2 + \beta^2)\omega_1^2 = \alpha^2\omega_0\omega_2. \tag{23.26}$$

The second question is more complicated. In the case of a Dupin cyclide, It can be split into two steps: Take first the conditions being sufficient for a supercyclide (since any Dupin cyclide is a special supercyclide) and then add the conditions which characterize the Dupin cyclides among the supercyclides. As to the first step, these conditions were given in [17]; they can be derived by the same method described above, but now applied to (23.19): Expanding the polynomials A, B, C, \dots into the Bernstein basis, say $A(u) = \sum_{i=0}^2 \alpha_i B_i(u)$ etc. one obtains by inserting this into (23.19)

$$\mathbf{b}_{i,k} = (\alpha_i\mathbf{a} + \beta_i\mathbf{b})\tau_k + \sigma_i(\gamma_k\mathbf{c} + \delta_k\mathbf{d}) \tag{23.27}$$

(σ_i, τ_k being the coefficients of F and G respectively). The resulting representation in homogeneous coordinates can be converted into a Euclidean one as before.

The following result [21], which is quoted without proof gives the answer to the second step:

Theorem 9 *A supercyclide of Class I is a Dupin cyclide if at least three conics of each family are circles lying in three different planes.*

23.4.2. Using cyclides as blendings

Blending is the most useful application of cyclides to CAGD and geometric modeling; many papers are devoted to that topic ([2], [3], [4], [7], [23], [26], [39], [44], [45]) and various generalizations have been found ([17], [18], [20], [33], [43]) in the past.

Usually, blending is done along prescribed curves on the surfaces to be blended; however, also “free blendings” where only the surfaces and some regions on it (where the transition curve is desired to lie in) are given, were discussed.

Because of the very extensive work on blending, only the principal methods can be quoted here, leaving the details for further reading in the literature. In this subsection, we deal with Dupin cyclides only, referring to the next subsection for the case of supercyclides.

The “one-sided” blending consists of a G^1 -continuous transition from a surface Φ to a Dupin cyclide Ψ along one of its curvature circles C . (We assume without further mentioning that both surfaces lie on *different* sides of the plane A of C and omit additional technical “side conditions”, e.g. that there is no other collision of those parts of the surfaces, which are of further interest.)

Since all Dupin cyclides have a right circular cone C^* or cylinder as envelope of the tangent planes around any curvature circle ¹⁰ C , the blending is done once both surfaces have C and C^* in common. Furthermore, the theory on cyclides implies that tangent cylinders instead of tangent cones can only occur when the cyclide is either a torus or when the circle C has extremal radius. Thus the one-sided blending consists only of the following two steps:

- Find a plane A intersecting Φ in a circle C so that the tangent planes of Φ around C envelop a right circular cone or cylinder C^* .
- Take a Dupin cyclide passing through C and having C^* as tangent cone around C .

This explains (in particular the first step), why blendings with Dupin cyclides are mostly constructed with natural quadrics, rotational quadrics, canal and pipe surfaces.

Obviously, the cyclide Ψ always exists and is by far not unique; due to Theorem 4, the only conditions are as follows

- The axis b of C^* is contained in one of the symmetry planes E or F
- The apex of C^* lies on the axis of that symmetry plane (a_1 for E and a_2 for F).
- The plane A contains the other axis (a_2 for E and a_1 for F).
- In the case of a cylinder the radii of the other circles of the same family as C must either be constant (Ψ then being a torus) or extremal at C .

Next we consider *two-sided* blendings: Then there are given a pair of objects C_1, C_1^* and C_2, C_2^* of the same kind as before. Usually one wants to span a part of a cyclide between these two geometric objects so that the circles belong to the same family ¹¹. The

¹⁰Degenerated Dupin cyclides and a blending along one of the axes of a parabolic cyclide are excluded here.

¹¹The other case that these circles belong to *different* families seems not to have been considered so far.

problem of existence becomes less trivial. However the theoretical results dealt with in Section 23.2 are able to solve it also in this case.

First, one concludes from the one-sided conditions that the two axes b_1 and b_2 of C_1^* and C_2^* must be coplanar and not identical, hence \mathbf{E} , respectively \mathbf{F} , is uniquely determined.

In the second step, one looks for the possibilities to blend a pair of different cones C_1^*, C_2^* with coplanar axes b_1, b_2 in \mathbf{E} , say, by a Dupin cyclide (neglecting the positions of the circles C_1, C_2 at the moment). This problem is solved by a theorem of Sabin ¹²

Theorem 10 *Two right circular cones or cylinders with different axes in the same plane can be blended by a Dupin cyclide if and only if they have a common inscribed sphere.* ¹³

The theorem does not say anything about the position of the circles C_1 and C_2 . On the other hand it is to be seen that one parameter remains free when the previous two steps are done. So one of them, say C_1 can be prescribed (for instance by choosing its mid point on the axis b_1 of C_1^*); then the position of C_2 is determined. This is a third condition to be imposed on the “geometric data” C_1, C_1^* and C_2, C_2^* for the existence of a blending cyclide. Then, in general, it does exist and is unique.

This theorem could be easily proved by methods of Lie geometry (see Section 23.5); however these are beyond the scope of this volume. An other way to recognize its validity is to use Theorem 3 on offsets of cyclides: Since the common inscribed sphere S has its midpoint at the intersection of the two axes, one can replace C_1^* and C_2^* by their inner offsets C_1°, C_2° at a distance d equal to the radius r of S ; then C_1°, C_2° have their apexes in common and so they can always be blended by a Dupin cyclide Ψ° . Now going back to the outer offsets of C_1°, C_2° and Ψ° yields the desired cyclide Ψ .

The proof of this last assertion as well as the many details and special cases must be omitted for brevity. Only a few remarks should be added in this context:

Remark 1 A parabolic Dupin cyclide occurs if and only if the two cones or cylinders C_1^* and C_2^* have a *common ruling*. This follows from Theorem 5 since the axes a_1 and a_2 of a parabolic cyclide (lie on it and) are degenerated curvature circles. So each family contains exactly one of it and any circular cone of the other family contains that line as a ruling.

Remark 2 For a general Dupin cyclide the tangent cones of the two *extremal longitudinal curvature circles* degenerate into planes (more precisely: into a pencil of lines in that plane having the intersection point with a_2 as its center). So a plane can also be blended with a cone or a cylinder by a Dupin cyclide; or even two different planes can be blended with each other ¹⁴.

Remark 3 The more general problem of constructing a free blending between surfaces is, in general, not solvable with Dupin cyclides. It would be desirable to have a class of surfaces that can blend say two intersecting quadrics of general shapes and positions ¹⁵. But the intersection of two quadrics is a *spatial* curve of fourth order (even if it has two

¹²Sabin communicated it to Pratt, who included the result into his paper [38]. The proof was originally given only for cones; later on it was completed by Shene [44] also for the cases of cylinders.

¹³The case of zero radius, i.e. cones with common apexes is included

¹⁴Theorems 3.1 and 3.2 of [2] seem to contradict to this assertion; but that is only caused by the author's more restrictive definition of a blending.

¹⁵One of those blending problems arises from the so-called “Cranfield object”

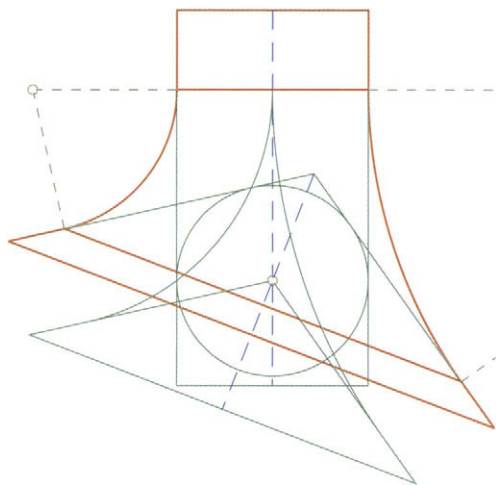


Figure 23.5. Construction of a cyclide blending between a cylinder and a cone using Sabin's theorem.

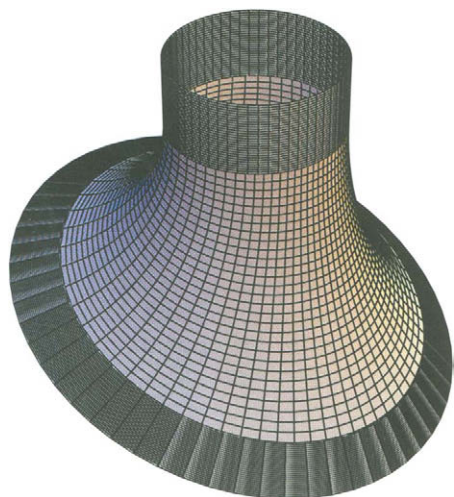


Figure 23.6. Cyclide blending between a cylinder and a cone (result of construction).

separate closed branches) which splits into two conics only in special cases. Nevertheless, there are examples where blendings with Dupin cyclides are possible; they can be found in [2] and [7]

23.4.3. Blending with supercyclides

Supercyclides have much more free parameters to meet blending requirements or specific desires on its shape. The junction curve can be an arbitrary (non degenerated) conic (or a part of it) assuming that it should be an iso-parameter line as before. Since any supercyclide possesses a tangent cone (or cylinder) along any non-degenerated iso-parameter conic, the surface to be blended with it must have that tangent conic too. Thus, in CAGD applications, blending is mostly done along the intersection of a quadric with a plane.

So the question arises whether two arbitrary quadrics \mathcal{Q}_1 and \mathcal{Q}_2 with given intersection planes \mathbf{E}_1 and \mathbf{E}_2 can be blended with a supercyclide along the intersection conics \mathbf{C}_1 and \mathbf{C}_2 (which should become two iso-parameter lines of the *same* family on the supercyclide). This configuration would completely determine the two axes of the supercyclide: $a_1 = \mathbf{E}_1 \cap \mathbf{E}_2$ and a_2 as the line joining the two apexes of the tangent cones \mathbf{C}_1^* and \mathbf{C}_2^* of \mathcal{Q}_1 and \mathcal{Q}_2 respectively (see Section 23.3.2). Thus one obtains — as a first condition for the existence of a blending supercyclide — that the axes must be skew ¹⁶. The second condition — also by the results of Section 23.3.2 — is more restrictive: There must exist a *central projection* with center \mathbf{z} on a_2 mapping \mathbf{C}_1 onto \mathbf{C}_2 .

If both conditions are satisfied, then a blending supercyclide exists and can easily be constructed as follows: One has to represent the two conics in Bézier form, say, so that

$$\mathbf{y}_2(u) = \mathbf{y}_1(u) + \mathbf{z} \tag{23.28}$$

holds (central projection with homogeneous coordinates!). Then one chooses a plane \mathbf{E} through a_2 containing two conic arcs of the second family. Since all of those join corresponding points of \mathbf{C}_1 and \mathbf{C}_2 , one selects one such pair, say $\mathbf{y}_1(0)$ and $\mathbf{y}_2(0)$, in \mathbf{E} and constructs that conic arc $\bar{\mathbf{C}}$ joining it. Since the two cones \mathbf{C}_1^* and \mathbf{C}_2^* have their apexes on a_2 , the two generators passing through $\mathbf{y}_1(0)$ and $\mathbf{y}_2(0)$ respectively *intersect* at a point \mathbf{q} and must be the tangents at the endpoints of $\bar{\mathbf{C}}$. Thus the three control points are known and it remains to choose a suitable weight arbitrarily. So, one gets

$$\bar{\mathbf{C}} \dots \mathbf{x}(v) = B_0(v)\mathbf{y}_1(0) + B_1(v)\mathbf{q} + B_2(v)\mathbf{y}_2(0). \tag{23.29}$$

Now taking $\mathbf{y}_1(u)$ and $\mathbf{y}_2(u)$ with *variable* parameter values v and replacing also \mathbf{q} by the corresponding points $\mathbf{q}(v)$ where the other generators of \mathbf{C}_1^* and \mathbf{C}_2^* intersect (note that the intersection of these cones splits into a conic, described by $\mathbf{q}(v)$, and a pair of straight lines), one obtains the representation of the kind (23.19) for the desired supercyclide.

Similar constructions can be designed when other entities are given. For instance, a pipe surface with an elliptic cross section has to join at one or at both sides another surface smoothly. Again, only the conics \mathbf{C}_1 and \mathbf{C}_2 as well as the tangent cones \mathbf{C}_1^*

¹⁶This condition is not essential because supercyclides with intersecting axes do exist; they were excluded here only for practical reasons (see [17], [42]).

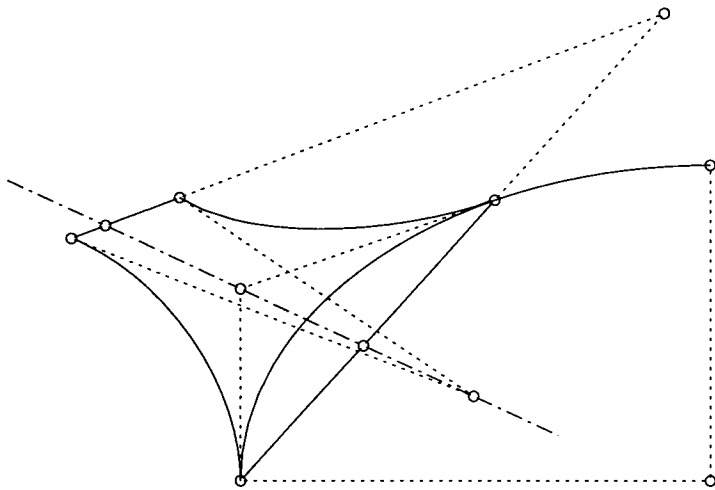


Figure 23.7. The blending construction in the symmetry plane.

and C_1^* are needed from the two surfaces at the ends. We assume that the whole figure should be symmetric with respect to a plane \mathbf{E} and the geometric data do so; furthermore a certain shape of the blending surface is wanted. Then one can perform the construction first in that symmetry plane (which plays the same role as \mathbf{E} in the example above). But now one has *two* conic arcs and the control points are already determined by the data. However, also in this case, there must be a central projection with center \mathbf{z} on a_2 mapping the one onto the other; this center is actually the intersection point of the axis a_2 with \mathbf{E} , because all plays in this plane (of course the lines $e_i = \mathbf{E}_i \cap \mathbf{E}$ ($i = 1, 2$) must intersect at \mathbf{z} since the planes \mathbf{E}_1 and \mathbf{E}_2 intersect in a_2). However these conditions are not yet sufficient: One has to observe that a central projection in a plane has always a straight line consisting only of fixed points and that the representation of Eq. (23.19) implies that the first axis a_1 coincides with it. The following figure shows the details, where the nozzle of a tea pot is designed.

The tea pot in Figure 23.8 is a rotational ellipsoid. At the front side a skew plane cuts out the bottom of the nozzle, thus having an elliptic cross section. The other end of the nozzle is designed to meet practical as well as aesthetic requirements.

If the construction in the symmetry plane is done, symmetry in space is really attained by taking the second axis orthogonal to that plane; if, additionally, the point \mathbf{z} and the vector \mathbf{v} (as a point at infinity on a_2) are chosen as base points for a_2 then a trigonometric

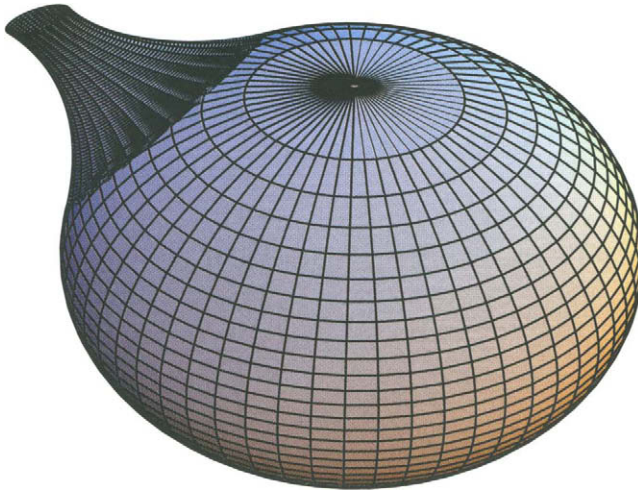


Figure 23.8. Blending of a rotational ellipsoid with a supercycloid.

representation with

$$C(\phi) = \alpha + \beta \cos(\phi), \quad D(\phi) = \lambda \sin(\phi), \quad G(\phi) = \gamma + \delta \cos(\phi), \tag{23.30}$$

is more favorable than that of (23.20) for a closed conic. (Note that $\sin(\phi)$ occurs only in the coefficient of \mathbf{v} .) The coefficients α, \dots, λ are calculated to obtain the proper extensions of the nozzle.

23.5. APPENDIX: STUDYING DUPIN CYCLIDES WITH LIE GEOMETRY

Some of the previous investigations may seem a little bit strange because one can not recognize a unifying principle behind them. Indeed, such a principle exists and is called “Lie geometry” [13] (already mentioned in Chapter 3 of this book).

The basic idea of Lie geometry is to map the set of all “Lie cycles” \mathcal{L} onto a hyperquadric \mathcal{Q} lying in the real projective space $P^5(\mathbb{R})$

$$\Lambda : \mathcal{L} \longrightarrow \mathcal{Q} \subset P^5(\mathbb{R}) \tag{23.1}$$

and to study configurations in the Euclidean space by looking at their properties in this “model space” of the Lie geometry.

Definition 1 *The set \mathcal{L} of Lie cycles is the union of the following sets:*

1. *The set \mathcal{S} of oriented spheres in \mathbb{E}^3 ,*
2. *the set \mathcal{U} of oriented planes,*

- 3. the set of points of \mathbf{E}^3 (also denoted by \mathbf{E}^3),
- 4. completed by one single point at infinity ∞ .

Thus:

$$\mathcal{L} = \mathcal{S} \cup \mathcal{U} \cup \mathbf{E}^3 \cup \{\infty\}.$$

The orientation of the spheres is positive, if the normal unit vectors point into the interior; then the radius is assigned a positive value. Otherwise the normal vectors point outwards and the radius is assigned a negative value. Similarly, a certain plane in \mathbf{E}^3 splits into two different Lie cycles according to the direction of their unit normal vectors.

The mapping (1) is defined separately for each of the different kinds of Lie cycles: For an oriented sphere $S_{\mathbf{a},r}$ (\mathbf{a} being the mid point and r the signed radius according to the orientation) the mapping Λ is given by ¹⁷

$$\Lambda(S_{\mathbf{a},r}) = (1, \mathbf{a}, \frac{1}{2}(\|\mathbf{a}\|^2 - r^2), r)\mathbb{R}. \tag{23.2}$$

This definition has to be suitably extended to points and planes: For points simply by setting $r = 0$ and for planes by passing to the limit $r \rightarrow \infty$; thus the image of the plane $E \dots \langle \mathbf{x}, \mathbf{n} \rangle - p = 0$ (with \mathbf{n} as oriented normal unit vector) is obtained as

$$\Lambda(E) = (0, \mathbf{n}, p, 1)\mathbb{R}. \tag{23.3}$$

Finally, one defines $\Lambda(\infty) = N = (0, 0, 0, 0, 1, 0)\mathbb{R}$. This point is called “the north pole” as in Moebius geometry (see Chapter 3, Section 3.2.2). The north pole is assumed to be different from all other geometric objects, to be incident with any plane but with no sphere. Now one can verify

Theorem 11 For all $\mathbf{S} \in \mathcal{L}$ the images $\Lambda(\mathbf{S})$ lie on the quadric \mathcal{Q} with the equation

$$\mathcal{Q} \dots \xi_1^2 + \xi_2^2 + \xi_3^2 - 2\xi_0\xi_4 - \xi_5^2 = 0 \tag{23.4}$$

and the mapping (23.1) is bijective. Furthermore, let $\mathbf{U} := T_N\mathcal{Q}$ be the tangent hyperplane to \mathcal{Q} at the point N with the equation

$$\mathbf{U} \dots \xi_0 = 0 \tag{23.5}$$

then the set \mathcal{U} of oriented planes is mapped onto $\mathbf{U} \setminus \{N\}$. Similarly, let \mathbf{M} be the hyperplane

$$\mathbf{M} \dots \xi_5 = 0 \tag{23.6}$$

then $\Lambda(\mathcal{P}) = \mathbf{M} \setminus \{N\}$.

Note that \mathbf{M} is not tangent to \mathcal{Q} and that, by (4), (5), (6), N is the only common point of \mathbf{U} , \mathbf{M} and \mathcal{Q} . The pole of \mathbf{M} will be denoted by P for use later on.

¹⁷The right hand side of the following equation is a class of proportional vectors of \mathbb{R}^6 , indexed by 0 ... 5; the three coordinates of \mathbf{a} have to be put at the places 1 ... 3.

Definition 2 *The subgroup L of projectivities of $P^5(\mathbb{R})$ leaving Q (as a whole) invariant is called the group of Lie transformations (or that of Lie motions). Geometric properties remaining invariant under Lie transformations are called Lie invariant.*

The study of Lie invariant properties of figures is called the “Lie geometry” (in the sense of F. Klein’s “Erlanger Programm”). By definition, the quadric Q itself is Lie invariant. It is often called “the absolute quadric of Lie geometry”.

Note that points and planes are *not* Lie invariant! They can be transformed into any other kind of Lie cycles; only the whole set \mathcal{L} of Lie cycles is Lie invariant. Thus the geometric objects U, M, N, P defined above are *not* Lie invariant but they carry over the *Euclidean* structure of E^3 into the model space. So, if one wants to obtain a Lie invariant property of some configuration $\mathcal{F} \subset E^3$, one has to consider nothing else than the relations of $\Lambda(\mathcal{F})$ with respect to Q ; if, in contrast, one wants to recover a *Euclidean* property, one has to observe the position of $\Lambda(\mathcal{F})$ with respect to U, M, N, P . These principles will be exploited in the sequel for the Dupin cyclides.

The Lie geometry can be considered as an extension of the *Moebius geometry* (with one more dimension): The hyperplane M together with the intersection quadric $Q' := Q \cap M$ is indeed a *model space of Moebius geometry*, because Q' has signature (4,1) and the restriction of the mapping Λ to the points of E^3 is given by

$$\Omega : E^3 \longrightarrow Q', \quad \Omega(\mathbf{p} = (1, \mathbf{p}, \frac{1}{2}(\|\mathbf{p}\|^2), 0)) \in \mathbb{R} \tag{23.7}$$

with

$$Q' \dots \xi_1^2 + \xi_2^2 + \xi_3^2 - 2\xi_0\xi_4 = 0, \quad \xi_5 = 0 \tag{23.8}$$

(set $r = 0$ in (23.2)). This is just the Moebius mapping if E^3 is completed by one point ∞ which is mapped to the north pole $(0, 0, 0, 0, 1, 0) \in \mathbb{R}$ in M . In this way *the Moebius geometry is imbedded into the Lie geometry*. In particular, one can choose any 3-space A , different from the tangent space $U \cap M$ at N to Q' (for instance $A \dots \xi_4 = 0$), and project Q' from the north pole N onto A (stereographic projection). By this procedure, the Euclidean space E^3 is embedded into $A \setminus U$, where the Euclidean structure is conserved ($A \cap U$ plays the role of the “plane at infinity” and $A \cap U \cap Q$ that of the absolute isotropic circle. (See also Chapter 3, Section 3.2.2)

We mention here without going into the details, that the Laguerre geometry can also be imbedded into the Lie geometry in a similar manner. Thus the Lie geometry contains both and one can show that it is in some sense “the smallest geometry” comprising both the Moebius and the Laguerre geometries. Clearly, Lie geometry can also be defined (in an analogous manner) for any dimension $n \geq 2$; but for the present purpose we confine ourselves to the case $n = 3$.

Now we come back to the Lie geometry itself and deal with one of its most fruitful notions which is called the “equi-oriented contact of two Lie cycles” and defined as follows:

Definition 3 *Two Lie cycles L_1 and L_2 have an equi-oriented contact if and only if one of the following conditions is satisfied:*

1. $L_1, L_2 \in S \cup U$ (two oriented spheres or planes) and they are tangent to each other with both normal vectors pointing into the same direction at the point of contact,

2. $L_1 \in \mathcal{S} \cup \mathcal{U}$ and $L_2 \in \mathbf{E}^3$ (one oriented sphere or plane and a point of \mathbf{E}^3) and the point L_2 is incident with the sphere or plane L_1 ,
3. $L_1 \in \mathcal{U}$ and $L_2 = \infty$ (an oriented plane and the point at infinity have always an equi-oriented contact)
4. $L_1, L_2 \in \mathbf{E}^3 \cup \{\infty\}$ and $L_1 = L_2$ (two points - finite or at infinity - have only an equi-oriented contact if they are coincident).

Now one can prove the following essential property of this notion:

Theorem 12 *A pair of Lie cycles L_1 and L_2 has an equi-oriented contact if and only if their Lie images $\Lambda(L_1)$ and $\Lambda(L_2)$ are conjugate (polar) to each other with respect to \mathcal{Q} .*

Proof: We give the proof only for the case $L_1, L_2 \in \mathcal{S} \cup \mathbf{E}^3$. Then we have by (2) $\Lambda(S_1) = (1, \mathbf{a}, \frac{1}{2}(\|\mathbf{a}\|^2 - r_1^2), r_1)\mathbb{R}$ and $\Lambda(S_2) = (1, \mathbf{b}, \frac{1}{2}(\|\mathbf{b}\|^2 - r_2^2), r_2)\mathbb{R}$. On the other hand, the polarity condition is by (23.4)

$$\xi_1\eta_1 + \xi_2\eta_2 + \xi_3\eta_3 - \xi_0\eta_4 - \xi_4\eta_0 - \xi_5\eta_5 = 0. \tag{23.9}$$

Inserting the above coordinates of $\Lambda(L_1)$, and $\Lambda(L_2)$ yields

$$\langle \mathbf{a}, \mathbf{b} \rangle - (\|\mathbf{a}\|^2 - r_1^2) - (\|\mathbf{b}\|^2 - r_2^2) - 2r_1r_2 = 0.$$

This is equivalent to $\|\mathbf{a} - \mathbf{b}\|^2 = (r_1 - r_2)^2$ showing that L_1, L_2 have indeed an equi-oriented contact. \square

Observing that any pair of different conjugate points on a quadric (in any projective space) span a line being *completely contained* in that quadric and vice versa, one recognizes at once:

Theorem 13 *Any line l being completely contained in \mathcal{Q} is either the image of a pencil of spheres which are tangent to a plane at a certain point (including these two Lie cycles itself) or the image of a family of parallel planes (with same orientation) completed by the point ∞ . The latter case occurs exactly if $l \in \mathcal{U}$ and this implies in addition $N \in l$.*

Now we turn to the Lie-geometric interpretation of *surfaces*. At any point $\mathbf{x}(u, v)$ of a surface one has just such a pencil of tangent spheres that is mapped onto a line $l(u, v)$ contained in \mathcal{Q} . So one has — as the Lie image of a surface \mathcal{F} — a *line congruence* \mathcal{K} (what means a two-parametric family of straight lines) in the model space. We express this fact simply as $\mathcal{K} = \Lambda(\mathcal{F})$. But line congruences have, in general, two *focal surfaces*, generated by the two focal points $F_1(u, v), F_2(u, v)$ on each line $l(u, v)$.

With a few lines of calculation (for instance using curvature lines as iso-parameter lines on the surface) one can easily prove:

Theorem 14 *The focal points F_i on each line $l(u, v)$ of the image $\Lambda(\mathcal{F})$ of a surface \mathcal{F} are the Lie images of the two curvature spheres $S_i(u, v)$ at that point of the surface ($i = 1, 2$).*

In particular, for a Dupin cyclide, we know from Section 23.2, Theorem 1, that these curvature spheres split into two one parameter families $\mathcal{S}_1 := \{\mathbf{S}_1(u) | u \in I_1\}$ and $\mathcal{S}_2 := \{\mathbf{S}_2(v) | v \in I_2\}$ and that, in addition, any sphere of the one family touches any sphere of the other at the corresponding surface point $\mathbf{x}(u, v)$. This is an equi-oriented contact if the radius functions are properly signed (see Section 23.2.2). Thus the focal surfaces $\mathcal{F}_i := \{F_i(u, v) \mid (u, v) \in I_1 \times I_2\}$ degenerate into two curves quite as those of the normals in the original Euclidean space. However, in the model space, $F_1(u)$ must be conjugate to $F_2(v)$ because of the equi-oriented contact (Theorem 12). Thus the whole linear subspaces E_1 and E_2 spanned by those curves \mathcal{F}_1 and \mathcal{F}_2 respectively are polar to each other with respect to \mathcal{Q} . As \mathcal{F}_i can not be a line (otherwise \mathcal{S}_i would be a pencil of spheres) one concludes $\dim(E_i) \geq 2 \quad (i = 1, 2)$. On the other hand, since \mathcal{Q} is a non-degenerated quadric in $P^5(\mathbb{R})$ one has $\dim(E_1) + \dim(E_2) = 4$ and this implies $\dim(E_i) = 2, (i = 1, 2)$. As the curves \mathcal{F}_i are contained in $E_i \cap \mathcal{Q}$, they must be conics and we can state:

Theorem 15 *The two focal curves \mathcal{F}_1 and \mathcal{F}_2 of the Lie image of a Dupin cyclide are a pair of real non-degenerated conics in two mutually polar planes E_1 and E_2 , i.e. $\mathcal{F}_i = E_i \cap \mathcal{Q}$ and $E_1 \cap E_2 = \emptyset$. Furthermore these focal curves are the Lie images of the two families of enveloping spheres: $\Lambda(\mathcal{S}_i) = \mathcal{F}_i, \quad (i = 1, 2)$.*

The inverse of this theorem is also true. However one has to include the following three degenerate cases into the notion of a ‘‘Dupin cyclide’’ (which were excluded in Section 23.2 by the assumption to be a regular surface):

1. A right circular cone or cylinder with the enveloping planes as \mathcal{S}_1 and the inscribed spheres as \mathcal{S}_2
2. The family of all spheres passing through a single circle with $r \neq 0$ (including the plane of it) as \mathcal{S}_1 and the points of this circle as \mathcal{S}_2 .
3. A pencil of planes as \mathcal{S}_1 and the points of the axis as \mathcal{S}_2

Then one has:

Theorem 16 (a) *Any pair of mutually polar planes each of which intersects the absolute quadric \mathcal{Q} in a real non-degenerated conic is the Lie image of a Dupin cyclide (in the sense of the previous theorem).*

(b) *Each Dupin cyclide is Lie equivalent to any other.*

(c) *The Dupin cyclide is one of the degenerate cases listed above if and only if $E_1 \subset \mathbf{U}$ or $E_2 \subset \mathbf{M}$ or both.*

To prove this, one firstly observes that any configuration $\mathcal{F}_1, \mathcal{F}_2$ of two conics with the property (a) is projectively equivalent to any other, since the absolute quadric \mathcal{Q} — having the signature $(4, 2)$ — can be transformed into the normal form

$$\eta_1^2 - 2\eta_0\eta_2 + \eta_4^2 - 2\eta_3\eta_5 = 0 \tag{23.10}$$

from which we take that E_1 defined by $\eta_3 = 0, \eta_4 = 0, \eta_5 = 0$ and E_2 defined by $\eta_0 = 0, \eta_1 = 0, \eta_2 = 0$ is indeed such a configuration $\mathcal{F}_1, \mathcal{F}_2$; thus, there is only one Lie type of it.

The degenerate cases are characterized by the fact that either one of the families of enveloping spheres consists only of planes (Case 1) or the other only of points (Case 2) or both together (Case 3). This proves (c) since planes are mapped into points of \mathbf{U} and points of \mathbf{E}^3 are mapped into points of \mathbf{M} . \square

As mentioned earlier, the different cases of *Euclidean preimages* of $\mathcal{F}_1, \mathcal{F}_2$ are distinguished by the *different positions* the hyperplanes \mathbf{U} and \mathbf{M} can have with respect to that configuration $\mathcal{F}_1, \mathcal{F}_2$. For brevity we exclude the degenerate cases from our further investigations.

First we deal with the position of $\mathcal{F}_1, \mathcal{F}_2$ with respect to \mathbf{U} . This hyperplane intersects the plane E_i in a line \mathbf{l}_i . Observing that \mathbf{U} is a tangent hyperplane of \mathcal{Q} and that \mathcal{Q} has signature (4,2), one can find (projective geometry of hyperquadrics) that there are exactly the following two cases:

- (A) One of the lines \mathbf{l}_i , ($i = (1, 2)$) intersects the corresponding conic $\mathcal{F}_i = E_i \cap \mathcal{Q}$ in a pair of conjugate complex points and the other line \mathbf{l}_j intersects $\mathcal{F}_j = E_j \cap \mathcal{Q}$ in a pair of real, different points
- (B) \mathbf{l}_i is tangent to the conic \mathcal{F}_i for both indices $i = 1, 2$.

Let, in Case (A), \mathcal{F}_2 be the conic with two real intersection points K_1, K_2 with \mathbf{U} . The north pole N can not lie on \mathbf{l}_2 (hence in E_2) since otherwise E_1 , being polar to E_2 , would be contained in \mathbf{U} . This would lead to a degenerate cyclide what we had excluded.

But there is the possibility that the line \mathbf{l}_2 meets the line joining N and the pole P of \mathbf{M} (since N is in \mathbf{M} , P lies in \mathbf{U} by main theorem of polar theory).

Thus the Case (A) splits into two subcases

- (I) The line $K_1 K_2$ meets the line NP (or equivalently: the four points $K_1 K_2, N$ and P are contained in a subplane of \mathbf{U}).
- (II) K_1, K_2, N and P span a 3-space.

The Case (B) will be maintained and denoted by (III) in the sequel. Thus we get three main classes of non degenerated Dupin cyclides as in Section 23.2.2.

Now we will state that these classes coincide with those of Section 23.2.2. As we know, the two families of enveloping spheres are mapped by Λ onto a pair of mutually polar conics $\mathcal{F}_1, \mathcal{F}_2$. In the Cases (A), (I) and (II), \mathcal{F}_2 intersects \mathbf{U} in the pair of real points K_1, K_2 . The preimages of them in \mathbf{E}^3 are two planes A_1 and A_2 which are tangent to all the spheres of \mathcal{S}_1 . Let $K_i = (0, \mathbf{n}_i, p_i, 1)\mathbb{R}$ (see (23.2)), then the condition (I) that PN meets $K_1 K_2$ implies that \mathbf{n}_1 and \mathbf{n}_2 are linearly dependent. So A_1 and A_2 are *parallel* to each other. Hence the cyclide must be rotational symmetric and consequently it is, indeed, a torus (in the sense of Section 23.2.2). In the other Case (A), (II), all the spheres of \mathcal{S}_1 are “squeezed” between two intersecting planes, what can not happen for parabolic Dupin cyclides. So the two Cases (II) and (III) correspond to those of Section 23.2.2. \square

The further classification into certain subclasses of (I), (II) and (III) described in Section 23.2.2 can be obtained by observing the positions of \mathcal{F}_1 and \mathcal{F}_2 with respect to the hyperplane \mathbf{M} . Since, for regular Dupin cyclides, the planes E_1 and E_2 are not contained

in \mathbf{M} (see Theorem 16 (c)), one has also the three possibilities of conjugate complex or coincident or real (different) intersection points of \mathcal{F}_i with \mathbf{M} . Denoting these three possibilities with the symbols C, T and R respectively, then from the nine combinations of them (for E_1 and E_2 independently) there remain only the five cases (CC), (CT), (CR), (TC), (RC) because of the following reason: If there would be a real intersection point R_i of E_i for both the indices then the line joining them (observe that E_1 and E_2 are skew, hence $R_1 \neq R_2$) would completely lie on the intersection quadric $\mathcal{Q}' := \mathcal{Q} \cap \mathbf{M}$ since E_1 is polar to E_2 . But this is an *oval* quadric (see (23.8)) having no lines at all.

Comparing these subcases for instance with those of Section 23.2.2, Case (II), one easily realizes that they correspond to the subcase numbers 3, 4, 5, 2, and 1 respectively. In a similar way, the subcases of (I) and (III) can be identified with the remaining possibilities of intersections of E_1 and E_2 with \mathbf{M} .

So the Lie geometry leads to a complete classification of the Dupin cyclides looking only at the position of \mathcal{F}_1 and \mathcal{F}_2 with respect to the hyperplanes \mathbf{U} and \mathbf{M} in the model space. Thus Lie geometry is a mean for better understanding the nature of Dupin cyclides.

REFERENCES

1. G. Albrecht and W.L.F. Degen. Construction of Bézier rectangles and triangles on the symmetric Dupin horn cyclide by means of inversion. *Computer Aided Geometric Design*, 14:349–375, 1997.
2. S. Allen and D. Dutta. Supercyclides and blending. *Computer Aided Geometric Design*, 14:637–651, 1997.
3. S. Allen and D. Dutta. Cyclides in pure blending I and II. *Computer Aided Geometric Design*, 14:51–75 and 77–102, 1997.
4. S. Allen and D. Dutta. Results on nonsingular cyclide transition surfaces. *Computer Aided Geometric Design*, 15:127–145, 1998.
5. M. Barner. Eine differentialgeometrische Kennzeichnung der allgemeinen Dupinschen Zykliiden. *Aequationes Mathematicae*, 34:277–286, 1987.
6. E. Blutel. Recherches sur les surfaces qui sont en même temps lieux de coniques et enveloppes de cônes du second degré. *Ann. sci. école norm. super.*, 7:155–216, 1890.
7. W. Boehm. On cyclides in geometric modeling. *Computer Aided Geometric Design*, 7:243–255, 1990.
8. W. Boehm. *Geometric Concepts for Geometric Design*. A.K. Peters, Wellesley, Massachusetts, 1998.
9. G. Bol. *Projektive Differentialgeometrie I – III*. Vandenhœck & Ruprecht, Göttingen, 1950–1967.
10. M.P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.
11. M. Casey. On cyclides and sphero-quartics. *Philosophical Transactions*, 161:585–721, 1871.
12. A. Cayley. On the cyclide. *Quarterly Journal of Pure and Applied Mathematics*, 12:148–165, 1873.
13. T.E. Cecil. Dupin submanifolds in Lie sphere geometry. *Differential Geometry and Topology*, Proceedings Tianjin, 1986–1987. *Lecture Notes in Math*, 1369, pages 1–48. Springer, NY, 1989.

14. V. Chandru, D. Dutta, and C.M. Hoffmann. On the geometry of Dupin cyclides. *The Visual Computer*, 5:277–290, 1989.
15. W.L.F. Degen. Surfaces with a conjugate net of conics in projektive space. *Tensor, N. S.*, 39:167–172, 1984.
16. W.L.F. Degen. Die zweifachen Blutelschen Kegelschnittflächen. *Manuscripta mathematica*, 55:9–38, 1986.
17. W.L.F. Degen. Generalized cyclides for use in CAGD. In A. Bowyer, editor, *The Mathematics of surfaces IV*, (The Institut of Math. & its Applic. Conference Series), pages 349–363. Clarendon Press, Oxford 1994.
18. W.L.F. Degen. Nets with plane silhouettes. In R.B. Fisher, editor, *The Mathematics of Surfaces V: Design and Application of Curves and Surfaces*, (The Institut of Math. & its Applic. Conference Series), pages 117–133. Clarendon Press, Oxford, 1994.
19. W.L.F. Degen. Projektive Differentialgeometrie. In O. Giering und J. Hoschek (Hrsg.), *Geometrie und ihre Anwendungen*, Carl Hanser Verlag München - Wien, 1994.
20. W.L.F. Degen. Conjugate silhouette nets. In A. Cohen, Ch. Rabut, and L.L. Schumaker, editors, *Curve and Surface Design (Saint Malo 1999)*. Vanderbilt Univ. Press, Nashville 2000.
21. W.L.F. Degen. Characterizing Dupin cyclides among supercyclides. (In preparation)
22. Ch. Dupin. *Applications de Géométrie et de Mécanique*. Bachelier, Paris, 1822.
23. D. Dutta, R.R. Martin, and M.J. Pratt. Cyclides in surface and solid modeling. *IEEE Computer Graphics and Applications*, 13:53–59, 1993.
24. A. Fladt and K. Baur. *Analytische Geometrie spezieller Flächen und Raumkurven*. Vieweg & Sohn, Braunschweig, 1975.
25. C.M. Jessop. *Quartic surfaces with singular points*, Cambridge Univ. Press, 1916.
26. K.J. Johnstone and C.-K. Shene. Dupin cyclides as blending surfaces for cones In R.B. Fisher, editor, *The Mathematics of Surfaces V: Design and Application of Curves and Surfaces*, (The Institut of Math. & its Applic. Conference Series), pages 3–29. Clarendon Press, Oxford 1994.
27. K.J. Johnstone. A new insertion algorithm for cyclides and swept surfaces using circle decomposition. *Computer Aided Geometric Design*, 10:1–24, 1993.
28. M. Kaps. Teilflächen einer Dupinschen Zykliide in Bézierdarstellung. PhD thesis, TU Braunschweig, 1990.
29. E. Kummer. Über die Flächen vierter Ordnung, welche eine Doppelcurve zweiten Grades besitzen. *Crelles's Journal f. Mathematik*, 69:142–184, 1870.
30. R. Krasuaskas. Rational Bézier surface patches on quadrics and the torus. Preprint 95-25, Vilnius University.
31. R. Krasuaskas and C. Mäurer. Studying cyclides with Laguerre geometry. Preprint no 1974, TU Darmstadt, Fachbereich Mathematik.
32. C. Mäurer. Rationale Bézier-Kurven und Bézier-Flächenstücke auf Dupinschen Zykliiden. PhD thesis, TU Darmstadt, 1997.
33. C. Mäurer. Generalized parameter representations of tori, Dupin cyclides and supercyclides. In A. Le Méhauté, C. Rabut, L.L. Schumaker, editors, *Curves and Surfaces with Applications to CAGD*, pages 205-302. Vanderbilt Univ. Press, Nashville, 1997.
34. C. Mäurer and R. Krasuaskas. Joining cyclide patches along quartic boundary curves.

- In M. Dæhlen, T. Lyche, and L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces II*, pages 359-366. Vanderbilt Univ. Press, Nashville, 1998.
35. R.R. Martin. *Principal Patches for Computational Geometry*. PhD thesis, Cambridge University Engineering Department, 1982.
 36. M. Paluszny and W. Boehm. General cyclides. *Computer Aided Geometric Design*, 15:699-710, 1998.
 37. H. Pottman and M.G. Wagner. Principal surfaces. In T.N.T. Goodman and R.R. Martin, editors, *The mathematics of Surfaces VII* (IMA Conference at Dundee, 1996), pages 337-362. Informations Geometers, Winchester, 1997.
 38. M.J. Pratt. Cyclides in computer aided geometric design. *Computer Aided Geometric Design*, 7:221-242, 1990.
 39. M.J. Pratt. The Virtues of Cyclides in CAGD. In T. Lyche and L.L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design II*, pages 457-473. Vanderbilt Univ. Press, Nashville, 1992.
 40. M.J. Pratt. Cyclides in computer aided geometric design II. *Computer Aided Geometric Design*, 12:131-152, 1995.
 41. M.J. Pratt. Dupin cyclides and supercyclides. In G. Mullineux, editor, *The mathematics of Surfaces VI* The Institute of Mathematics and its Applications, pages 43-66. Oxford University Press, Oxford, 1996.
 42. M.J. Pratt. Quartic supercyclides I: Basic theory. *Computer Aided Geometric Design*, 14:671-692, 1997.
 43. M.J. Pratt. On a class of Pythagorean-normal surfaces with planar lines of curvature. In R.J. Crips, editor, *The Mathematics of Surfaces VIII*, (IMA Conference Birmingham). Informations Geometers, Winchester, 1998.
 44. Ch.-K. Shene. Blending two cones with Dupin cyclides. *Computer Aided Geometric Design*, 15:643-673, 1998.
 45. Y.L. Srinivas and D. Dutta. Motion planning in three dimensions using cyclides. In T. Kunii, editor, *Visual Computing: Integrating Computer Graphics with Computer Vision*, pages 781-791. Springer-Verlag, Tokyo, 1992.
 46. Y.L. Srinivas and D. Dutta. Intuitive procedure for constructing geometrically complex objects using cyclides. *Computer-Aided Design*, 26:327-335, 1994.
 47. Y.L. Srinivas and D. Dutta. Blending and joining using cyclides. *ASME J. Mech. Des.* 116(4):1034-1041, 1994.
 48. Y.L. Srinivas and D. Dutta. Rational parametric representations of parabolic cyclides: Formulation and applications. *Computer Aided Geometric Design*, 12:551-566, 1995.
 49. Y.L. Srinivas, K.P. Kumar, and D. Dutta. Surface design using cyclide patches. *Computer-Aided Design*, 28:263-276, 1996.
 50. K. Ueda. Normalized cyclide Bézier patches. In M. Dæhlen, T. Lyche, L.L. Schumaker, editors, *Mathematical Methods in CAGD III*, pages 1-10. Vanderbilt Univ. Press, Nashville 1995.

Chapter 24

Geometry Processing

Thomas A. Grandine

24.1. INTRODUCTION

One of the fundamental capabilities required in geometric design is the ability to analyze candidate designs, both geometrically and physically. The latter involves estimating physical characteristics of a design in terms of structural strength, flow of fluids in and around the design, optical and electromagnetic properties, thermal characteristics, and so on. Balancing the desired and required physical and performance characteristics of any proposed geometric design is the essence of engineering. Although physical modeling of design geometry is beyond the scope of this book, it is touched on in more detail in the chapters on Shape Optimization, Finite Element Approximation with Splines, and An Industrial Look at CAGD.

This chapter will deal instead with the topic of geometric analysis. This type of analysis ignores dynamics and physics and focusses instead on those quantities which can be derived solely from given geometry. Examples of geometric analysis include determination of extreme points of the geometry, intersections of geometrical components, and metric information such as position, length, area, or volume. Such analysis is often called “geometry processing” in the literature because the collection of techniques required form the bridge between the geometric input and the computed output.

Most geometry processing can be accomplished through some combination of evaluation of functions, contouring, rootfinding, and numerical integration. Two of these topics are covered in some detail in other chapters of the book and won't be repeated here. Evaluation of functions is discussed in detail in the chapters on Bezier Techniques, Spline Basics, and Subdivision Surfaces. Contouring is needed for performing surface intersection and projection, and it can be found in Chapter 25 on Intersection problems. The two remaining topics, rootfinding and numerical integration, will be covered here, along with a discussion of how both can be used to create more sophisticated geometry processing techniques.

24.2. ROOT FINDING

Suppose one is given a mapping $f : [0, 1] \rightarrow \mathbb{R}$. The rootfinding problem is that of identifying a value or values u such that $f(u) = 0$. Much has appeared in the literature about this problem, as well as the more general problem of solving nonlinear systems of equations, see [7], and little of that material will be repeated here. Instead, this section will focus on special techniques that can be used to solve this equation in the special case where the function f is a spline, while the second part will deal with the more general problem of systems of spline equations.

One of the great difficulties with rootfinding from the geometry processing point of view is that the problem usually needs to be solved, often repeatedly, in the context of a much larger analysis that is being carried out. For example, given m points in space and a collection of n curves, the problem of identifying the closest point on any curve to each of the points requires, at least on the face of it, all solutions to mn different rootfinding problems to be found. The inability to solve even a single one of these problems places the reliability of the entire analysis in doubt.

Moreover, a global solution to each of the problems is required for this and other problems. What this means is that all of the values u which satisfy $f(u) = 0$ must be identified. Doing this reliably means that global knowledge of the function f is required in some form, e.g. bounds on function values and derivatives over arbitrary pieces of the function domain. Without this global knowledge, only local methods (i.e. methods which depend on the values of f and its derivatives at points) can be used, and such methods are inherently unreliable for solving the global problem. In particular, it's worth noting that geometry systems based solely on black box geometry, i.e. strict parametric evaluator systems in which the geometry processing facilities of the system have knowledge of the geometry only in terms of local information, are necessarily unreliable in terms of their geometry processing capability.

Although many function spaces have the property that global knowledge of a function can be deduced from its representation, this section will focus on rootfinding under the assumption the function f is a spline function represented as a linear combination of B-splines. This assumption is reasonable because spline functions are widely used for geometric design, and their representation makes it possible to construct sharp bounds on both function values and derivatives over arbitrary intervals.

One of the simplest methods which can take advantage of such bounds is the interval Newton method proposed by Hansen [6], with special customization for splines described in [4]. Each iteration of this nonlinear procedure starts with an interval $[a_n, b_n]$ in which zeroes of some function f are sought. The Newton-like step of the nonlinear procedure is to compute

$$[a'_{n+1}, b'_{n+1}] = x_n - \frac{f(x_n)}{f'([a_n, b_n])}, \quad (24.1)$$

where x_n is the midpoint of the interval $[a_n, b_n]$, and $f'([a_n, b_n])$ is an interval which bounds the value of the derivative of f over the interval $[a_n, b_n]$. The right hand side of this equation should be carried out using interval arithmetic. Hansen proves that all zeroes of f which are in the interval $[a_n, b_n]$ must necessarily also lie within the interval $[a'_{n+1}, b'_{n+1}]$.

Thus, the final step of the iteration is to consider only those solutions which lie in both intervals, i.e. construct a new interval $[a_{n+1}, b_{n+1}]$ satisfying

$$[a_{n+1}, b_{n+1}] = [a_n, b_n] \cap [a'_{n+1}, b'_{n+1}]. \quad (24.2)$$

The interval computation on the right hand side of (24.1) is straightforward. The numerator of the fraction is the scalar $f(x_n)$, while the denominator is the interval $f'([a_n, b_n])$, i.e. the interval obtained by evaluating f' over all of the points in the interval $[a_n, b_n]$. The result should be the interval obtained by dividing the numerator by each of the possible values of the denominator. If the denominator interval does not contain zero, this result is straightforward. If it does contain zero, then the numerator is divided by numbers which are both positive and negative and arbitrarily small. In this case, the resulting interval is actually the union of two intervals which stretch to infinity in both the positive and negative directions.

The size of the interval $f'([a_n, b_n])$ plays a crucial role in the convergence rate of this algorithm. If the derivative of f is not bounded sharply, then the interval $[a'_{n+1}, b'_{n+1}]$ will be larger than necessary, and so will $[a_{n+1}, b_{n+1}]$. On the other hand, if the bounds on the derivative of f are sharp, then the subsequent intervals in (24.1) will be smaller.

If f is a spline function, its derivative can be determined by differencing its B-spline coefficients, i.e. B-spline coefficients for a spline g can be determined by computing weighted differences of the B-spline coefficients for f such that g is the derivative of f everywhere. Moreover, bounds on g over any given interval can be determined by knot insertion at the endpoints of the interval, making use of the well-known convex hull property of B-splines, i.e. the observation that the value of a spline at any point in an interval is a convex combination of the values of the B-spline coefficients whose corresponding B-splines are non-zero over that interval. See the chapter 6 on Spline Basics for more details on this technique.

As an example, consider the problem of computing all of the zeroes of the cubic spline f defined by the knots $\{0, 0, 0, 0, 0.5, 1, 1, 1, 1\}$ and whose coefficients are given by $\{1, -2, -3, -1, 2\}$ over the interval $[0, 1]$. This spline is shown in Figure 24.1. The picture suggests that the equation $f(x) = 0$ has two roots.

Consider the interval Newton method on the problem of solving $f(x) = 0$. The derivative of the function f is the quadratic spline with knots $\{0, 0, 0, 0.5, 1, 1, 1\}$ and B-spline coefficients $\{-18, -3, 6, 18\}$. Initially, set $[a_0, b_0] = [0, 1]$ and consider the right hand side of (24.1). The midpoint is $x_0 = 0.5$, and $f(x_0) = -2.25$. Looking at g , it's easy to see that over the interval $[0, 1]$, the derivative of f lies in $[-18, 18]$. Thus, the fractional part of the right hand side of (24.1) must lie inside $[-\infty, -0.125] \cup [0.125, \infty]$. Thus,

$$[a'_1, b'_1] = [-\infty, 0.375] \cup [0.625, \infty],$$

and $[a_1, b_1]$, calculated by intersection of intervals, has the two pieces

$$[a_1, b_1] = [0, 0.375] \cup [0.625, 1].$$

At this point, the second of these intervals can be stored off to the side for later processing, and attention turned to the first piece. Here $x_1 = 0.1875$, and $f(x_1) = -1.412598$. By inserting three knots into the B-spline representation for g at 0.375, it's

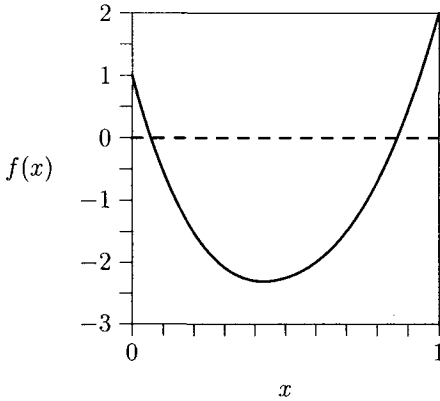


Figure 24.1. A function with two zeroes

revealed that $f'(x) \in [-18, -1.40625]$ for all $x \in [0, 0.375]$. Carrying out the computation for the right hand side of (24.1) again reveals that

$$[a'_2, b'_2] = [-0.817014, 0.109022],$$

so

$$[a_2, b_2] = [0, 0.109022].$$

Performing one more iteration reveals that $[a_3, b_3] = [0.060382, 0.0633485]$, and two more after that reveal $[a_5, b_5] = [0.06172202, 0.06172202]$, isolating one of the roots of the function to within seven decimal places.

Returning to the second part of $[a_1, b_1]$ which had been put aside, this can now be revisited. Assigning $[a_6, b_6] = [0.625, 1]$ and carrying out another iteration of (24.1) shrinks the interval to

$$[a_7, b_7] = [0.845676, 0.954051].$$

As before, another iteration reveals that $[a_8, b_8] = [0.861997, 0.872864]$, and two more after that result in $[a_{10}, b_{10}] = [0.8664224, 0.8664224]$, isolating the other root of the function to within seven decimal places. Thus, the interval Newton method has revealed that the spline function depicted in Figure 24.1 does indeed have two zeroes in the interval $[0, 1]$, and they occur at 0.06172202 and 0.8664224.

From the point of view of geometry processing reliability, the fact that the interval Newton method is mathematically guaranteed to locate all of the places at which f is zero puts those capabilities on sound mathematical ground. In particular, it is possible to make global analytical statements of fact about nonlinear geometry without qualification or assumption, something which is impossible when purely local search methods are used.

Of course, many applications require the solution of a system of nonlinear equations. For example, suppose one wishes to identify the closest point on a surface to a given point.

If the point is given by P and the surface mapping is given by $S : [0, 1]^2 \rightarrow \mathbb{R}^3$, then the point on the surface S closest to P can be found by identifying all the points $(u, v) \in [0, 1]^2$ for which $\|S(u, v) - P\|$ is locally minimized, and choosing the global minimum from among those candidates. A somewhat larger list of candidates can be constructed by finding all of the critical points of the distance, i.e. all solutions of the nonlinear system

$$\begin{aligned} S_u(u, v) \cdot (S(u, v) - P) &= 0 \\ S_v(u, v) \cdot (S(u, v) - P) &= 0 \end{aligned}$$

The set of solutions to this system will necessarily include all of the local minimizers for $\|S(u, v) - P\|$ in the interior of $[0, 1]^2$, so the global minimizer can be identified as a member of this set, too.

Thus, the crucial subproblem is to identify all of the zeroes of a nonlinear system of equations. As before, this is a global problem for which global information will have to be used. For the purposes of this chapter and section, this will be accomplished once again by requiring that the nonlinear system of equations be described by tensor product spline functions. Specifically, given an m -dimensional domain $X = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_m, b_m]$ and a set of m tensor product spline maps $f_i : X \rightarrow \mathbb{R}$ for $i = 1, \dots, m$, the goal is to find all $\vec{x} \in X$ such that $f_i(\vec{x}) = 0$ for $i = 1, \dots, m$.

As it turns out, the interval Newton method can be generalized to systems of equations, with equation (24.1) replaced by

$$X'_{n+1} = \vec{x}_n - J(X_n)^{-1}F(\vec{x}_n), \tag{24.3}$$

where $J(X_n)$ is the matrix of intervals whose entry in row i and column j is the interval which bounds the partial derivative of f_i with respect to its j th variable. Similarly, \vec{x}_n is the point located at the center of the box X_n , and $F(\vec{x}_n)$ is the vector whose i th component is $f_i(\vec{x}_n)$.

This turns out to be only one of several interval Newton methods for systems. In its many variations, the matrix $J(X_n)$ can take many forms, with some of the intervals replaced by scalars in some cases [6], usually resulting in faster convergence rates. However, these advantages depend crucially on knowing the algebraic form of the nonlinear system. Thus, systems involving “black box” functions cannot take advantage of any of these improvements.

As before, knowledge that the system of equations is a tensor product spline system of equations can be exploited. The interval Newton method accomplishes this by taking advantage of the fact that derivative values can be bounded accurately over arbitrary portions of the subdomains of the functions. Another method, the projected polyhedron (PP) method of Sherbrooke and Patrikalakis [8], accomplishes the same feat by taking advantage of the relationship B-spline coefficients have with the functions they represent.

Consider the B-spline representation of the tensor product spline function f_i in the nonlinear system above. This is given by

$$f_i(x_1, x_2, \dots, x_m) = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \dots \sum_{j_m=1}^{n_m} \alpha_{ij_1j_2\dots j_m} B_{1,j_1}(x_1) B_{2,j_2}(x_2) \dots B_{m,j_m}(x_m). \tag{24.4}$$

In this formula, B_{k,j_k} is the j_k th B-spline in the k th independent variable.

This B-spline representation can be used to construct two new functions of the k th independent variable alone which bound f_i . Consider

$$e_{ik}(x_k) = \sum_{j_k=1}^{n_k} \beta_{ij_k} B_{k,j_k}(x_k) \quad (24.5)$$

where

$$\beta_{ij_k} = \min_{j_1, \dots, j_{k-1}, j_{k+1}, \dots, j_m} \alpha_{ij_1 j_2 \dots j_m}.$$

Consider also

$$g_{ik}(x_k) = \sum_{j_k=1}^{n_k} \gamma_{ij_k} B_{k,j_k}(x_k) \quad (24.6)$$

where

$$\gamma_{ij_k} = \max_{j_1, \dots, j_{k-1}, j_{k+1}, \dots, j_m} \alpha_{ij_1 j_2 \dots j_m}.$$

Recall from Chapter 6 on Spline Basics that B-splines form a partition of unity and are non-negatively valued everywhere. These facts make clear that

$$e_{ik}(x_k) \leq f_i(x_1, x_2, \dots, x_m) \leq g_{ik}(x_k) \quad (24.7)$$

for all x_k regardless of the values of the other independent variables.

The principal idea behind the PP algorithm is the observation that, because of the inequalities (24.7), the nonlinear system cannot have a solution if there exist values of x_k for which e_{ik} and g_{ik} have the same sign, for f_i is bounded away from zero there. Since (24.7) must hold not only for each f_i , but also for each k , a systematic procedure, based on those inequalities, for pruning away from X those regions in which solutions cannot be found can be constructed.

In practice, identifying the actual zeroes of the univariate spline functions e_{ik} and g_{ik} is not performed. Instead, the convex hulls of the so-called “control polygons” for those two functions are calculated, and the zeroes of these are calculated to determine the subinterval of $[a_k, b_k]$ in which all the zeroes of f_i must lie. This computation is much less expensive than finding the zeroes of e_{ik} and g_{ik} , though its interval reduction estimates are more conservative.

Here is the entire algorithm:

- 0 Initialize $k = 0$.
- 1 Construct an initial box of search $X = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_m, b_m]$.
- 2 Scale and shift the box so that it becomes $[0, 1]^m$. Keep track of the scaling and shifting relationship to the initial box. The purpose of this operation is to improve the accuracy of the knot insertions which follow.
- 3 Increment k . If k is greater than m , then set $k = 1$.

- 4 For $i = 1, \dots, m$, calculate the B-spline coefficients of e_{ik} and g_{ik} . Form “control polygons” for these functions by using knot averages as abscissae and B-spline values as ordinates. Calculate the convex hull of the control polygons and determine which portion of the x -axis passes through it. Call that portion $[a_{ik}, b_{ik}]$. Finally, calculate

$$[a'_k, b'_k] = [a_{1k}, b_{1k}] \cap [a_{2k}, b_{2k}] \cap \dots \cap [a_{mk}, b_{mk}].$$

If $b'_k - a'_k > 0.7$, split the interval $[a'_k, b'_k]$ into two pieces, retain the first piece, and place the second aside for further processing.

- 5 Form

$$X' = [0, 1] \times \dots \times [a'_k, b'_k] \times \dots \times [0, 1].$$

- 6 Using all of the scalings and shiftings from [2], determine if X' is sufficiently small to have isolated a root. If so, report that result. Otherwise, set $X = X'$, replace the f_i with splines defined over the new X , using either simple knot insertion or the Oslo algorithm, and return to [2].

An example will clarify how this algorithm works. Suppose one has been given tensor product spline functions f and g of two variables. Suppose that they are both defined over $[1, 2] \times [1, 3]$, and that both are quadratic. Suppose that f has a single knot in the second independent variable, which can be labeled y , at $y = 2$, and suppose that g has a single knot in the first variable, which can be labeled x , at $x = 1.5$. Suppose that the matrix of B-spline coefficients for f is given by

$$\begin{pmatrix} 1 & 1 & 1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \\ 1 & 1 & 1 \end{pmatrix}$$

and that the matrix B-spline coefficients for g is given by

$$\begin{pmatrix} -1 & -2 & -1 & -1 \\ 0 & 0 & 0 & 0 \\ 2 & 1 & 2 & 2 \end{pmatrix}.$$

In these matrices, the rows correspond to x , and the columns correspond to y . Thus, the upper left hand corner of each matrix contains the B-spline coefficient which is the value of the functions f and g at the point $(1, 1)$, while the lower right hand corner contains the B-spline coefficient which is the value of f and g at the point $(2, 3)$.

The initial search box for the algorithm is $X = [1, 2] \times [1, 3]$. This is shifted and rescaled so that it becomes $[0, 1]^2$. None of the B-spline coefficients need to be changed. Now calculate the B-spline coefficients of e_{11} and g_{11} . This is accomplished by taking the minimum and maximum in each column of the matrix of B-spline coefficients for f . Thus, e_{11} has B-spline coefficients $\{-1, 1, -1\}$, while g_{11} has B-spline coefficients $\{1, 2, 1\}$. The knot averages provide abscissae of $\{0, 0.5, 1\}$, so the convex hull of the “control polygons” is shown in Figure 24.2.

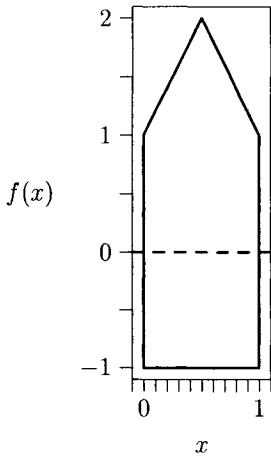


Figure 24.2. The convex hull of the “control polygon” for f

In this figure, note that the x -axis is shown by a dashed line, and the range of this line that is contained inside the convex hull is the range in which all of the zeroes of f are known to lie.

In this particular case, not much useful information is gleaned. The discovery that all of the zeroes of f occur when $x \in [0, 1]$ (which we really know to be the interval $[1, 2]$) is not very helpful. The next step is to repeat the process for e_{21} and g_{21} . The B-spline coefficients for these functions are obtained by taking the minimum and maximum in each column of the matrix of B-spline coefficients for g . Thus, e_{21} has B-spline coefficients $\{-1, -2, -1, -1\}$, while g_{21} has B-spline coefficients $\{2, 1, 2, 2\}$. The knot averages again provide abscissae for the “control polygons,” and these are given by $\{0, 0.25, 0.75, 1\}$, and their convex hull is shown in Figure 24.3.

As before, this isn’t very interesting. Refinement of the interval on the first variable revealed

$$[a'_1, b'_1] = [0, 1] \cap [0, 1] = [0, 1].$$

Since the reduction in interval size is not sufficiently small, the interval needs to be split. Place $[0.5, 1]$ aside for further processing, and consider

$$[a'_1, b'_1] = [0, 0.5].$$

Over this subdomain, the B-spline coefficients for f and g can be calculated again. They are given by

$$\begin{pmatrix} 1 & 1 & 1 \\ -1 & 0.5 & 0.5 \\ -1 & 0.5 & 0.5 \\ 1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} -1 & -2 & -1.5 \\ 0 & 0 & 0 \\ 2 & 1 & 1.5 \end{pmatrix}.$$

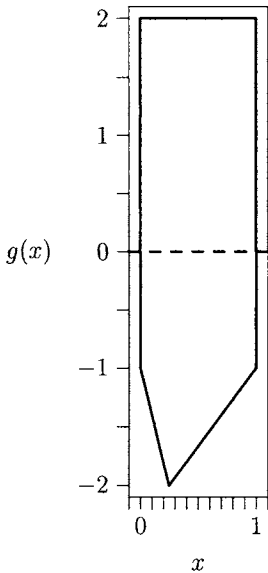


Figure 24.3. The convex hull of the “control polygon“ for g

Note that the number of B-spline coefficients for g changed because the interior knot in x for that function is no longer present.

At this point, the y variable can be processed. Before doing so, the intervals again need to be rescaled so that the B-spline coefficients just identified are assumed to define a function over $[0, 1]^2$. By recalling the first scaling, however, one can note that the domain at this point is really $[1, 1.5] \times [1, 3]$.

As before, the B-spline coefficients of e_{12} and g_{12} are now calculated. These are generated by taking the minimum and maximum of the rows of the two matrices of B-spline coefficients for f and g . Thus, the B-spline coefficients for e_{12} are given by $\{1, -1, -1, 1\}$, with knot averages $\{0, 0.25, 0.75, 1\}$ to get the abscissae of the “control polygon,” while the B-spline coefficients for g_{12} are given by $\{1, 0.5, 0.5, 1\}$. The convex hull is shown in Figure 24.4, and for the first time in the problem, a non-trivial reduced interval now appears.

The portion of the interval that is contained inside the polygon is $[0.125, 0.875]$, i.e. f can have no zeroes in either the first or last eighth of the interval. Now consider g , for which the B-spline coefficients of e_{22} and g_{22} are needed. These are given by $\{-2, 0, 1\}$ and $\{-1, 0, 2\}$, respectively. After generating the convex hulls and intersecting, it is found that g cannot have any zeroes outside $[0.3333333, 0.6666667]$. Thus,

$$[a'_2, b'_2] = [0.125, 0.875] \cap [0.3333333, 0.6666667] = [0.3333333, 0.6666667].$$

Once again, the B-spline coefficients for f and g are recalculated over this new subdo-

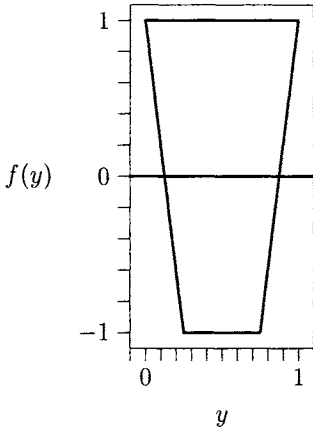


Figure 24.4. The convex hull of the “control polygon” for f (again)

main, and they are found to be

$$\begin{pmatrix} -0.7777778 & 0.5555556 & 0.5555556 \\ -1 & 0.5 & 0.5 \\ -1 & 0.5 & 0.5 \\ -0.7777778 & 0.5555556 & 0.5555556 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} -0.2222222 & -0.7777778 & -0.5 \\ 0.2222222 & -0.2222222 & 0 \\ 0.7777778 & 0.2222222 & 0.5 \end{pmatrix}.$$

The algorithm continues processing. These B-spline coefficients are taken to be the definition of a pair of functions defined over $[0, 1]^2$, but which in reality is $[1, 1.5] \times [1.666667, 2.333333]$. The next pass through will again focus on the x variable. For the functions at this stage, the B-spline coefficients of e_{11} and g_{11} are given by $\{-1, 0.5, 0.5\}$ and $\{-0.7777778, 0.5555556, 0.5555556\}$. Similarly, the B-spline coefficients for e_{21} and g_{21} are given by $\{-0.2222222, -0.7777778 - 0.5\}$ and $\{0.7777778, 0.2222222, 0.5\}$. Generating “control polygons”, convex hulls, and subintervals reveals

$$[a'_1, b'_1] = [0.2916667, 0.6666667] \cap [0, 1] = [0.2916667, 0.6666667].$$

The B-spline coefficients for f and g are now calculated over this subdomain, and they are found to be

$$\begin{pmatrix} -0.1134259 & 0.2407407 & 0.4074074 \\ -0.2526042 & 0.1458333 & 0.3333333 \\ -0.2526042 & 0.1458333 & 0.3333333 \\ -0.1134259 & 0.2407407 & 0.4074074 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} -0.4754051 & -0.5925926 & -0.5925926 \\ 0.0196759 & -0.0740741 & -0.0740741 \\ 0.5245949 & 0.40740741 & 0.40740741 \end{pmatrix},$$

while the region is known to be $[1.145833, 1.333333] \times [1.666667, 2.333333]$.

Subsequent iterations refine this box, as shown in Table 24.1. In practice, the computations would be carried out to additional precision, and additional iterations would be required in order to isolate the root.

Table 24.1
Search boxes for the PP algorithm example

Iteration	New Box
y	$[1.145833, 1.333333] \times [1.983603, 2.061728]$
x	$[1.204370, 1.226667] \times [1.983603, 2.061728]$
y	$[1.204370, 1.226667] \times [2.022270, 2.032882]$
x	$[1.210938, 1.211367] \times [2.022270, 2.032882]$
y	$[1.210938, 1.211367] \times [2.025611, 2.025822]$
x	$[1.211132, 1.211136] \times [2.025611, 2.025822]$
y	$[1.211132, 1.211136] \times [2.025706, 2.025708]$
x	$[1.211134, 1.211134] \times [2.025707, 2.025707]$

After this root has been identified, attention is returned to the other search box which was created back in the first iteration, when an interval was split in two. Additional refinement of this search box reveals that a second solution to this nonlinear system can be found at (1.794495, 1.858724).

In this example, the search box was split exactly once, and a unique solution to the nonlinear system was eventually found in each box. In many practical cases, the search box needs to be split many times, with no solution found in many of these boxes. The algorithm recognizes this situation during the computation of the interval $[a'_k, b'_k]$ when the intersection of all the intervals which form it is determined to be the empty set.

The PP algorithm has been a key component of the topology resolution scheme used in a surface intersection algorithm developed at Boeing ([5]), and it has formed one of the main computational workhorses of the U.S. Navy's DT_NURBS geometry library [2]. Undoubtedly, it will be an important algorithm for many years to come.

24.3. INTEGRATION

In addition to rootfinding, many geometric analyses require integration over curves and surfaces as a key component of the analysis. Such computations include length of curves, area of surfaces, center of gravity and moment of inertia of volumes, mass of variable density media, etc. Numerical integration is a well-studied and well-established discipline about which volumes have been written. This section will focus on some of the issues that arise when these methods are applied to problems in geometric modeling.

As an example, consider the problem of computing the length of a curve. Consider the parametric spline curve given by knots $\{0, 0, 0, 0, 1, 1, 1, 1\}$, with B-spline coefficients for the x component $\{0, 0.3, 0.7, 1\}$ and B-spline coefficients for the y component equal to $\{0, 1, 1, 0\}$. Because the knot sequence has no internal knots, this spline is made up of a single polynomial piece, and its curve is depicted in Figure 24.5.

The length of this curve is given by

$$\int_0^1 \sqrt{x'(u)^2 + y'(u)^2} du,$$

the integral of the parametric speed of the curve over the entire curve. In this case,

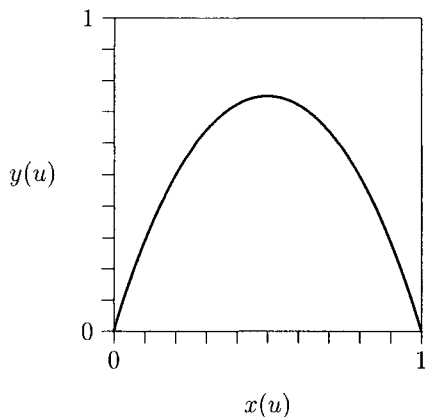


Figure 24.5. A simple curve with unknown length

both x' and y' are quadratic polynomials, so the integral is the square root of a quartic polynomial. This integral cannot be evaluated analytically by any practical method, so its value will have to be estimated numerically. A simple numerical method for evaluating an integral is given by Simpson's rule

$$\int_a^b f(u) du \approx (b-a) \left(\frac{1}{6}f(a) + \frac{4}{6}f\left(\frac{a+b}{2}\right) + \frac{1}{6}f(b) \right).$$

Usually, this does not provide a sufficiently accurate estimate of the value of the integral. In these cases, the integral can be chopped up into smaller pieces, and Simpson's rule can be applied to each piece. The estimated values of the integrals over each of the small pieces can then be summed to get a more accurate estimate of the value of the integral. To see the effect of doing this, consider the results in Table 24.2.

The first column of this table shows the number of equally spaced pieces that the arc length integral has been cut into. The second column is the estimate of the value of the total integral when all of the pieces are summed. The third column is the absolute value of the difference between the estimated integral value and the actual value. The fourth column is the calculated estimate of the order of convergence of the method, about which more needs to be said.

Recall that published quadrature formulas all come with error estimates. In the case of Simpson's rule, the estimate is given by [1]

$$E_S = -\frac{f^{iv}(\xi)(h/2)^4(b-a)}{180},$$

where h is the width of the subpieces of the integrals, and ξ is some (unknown) point in the interval $[a, b]$. The important observation is that reducing the value of h by a factor of two should result in an approximate reduction in the error of the integral by a factor of 16, a natural consequence of the exponent of the $h/2$ term in the error expression.

Table 24.2
Estimates of curve length using Simpson’s rule

Pieces	Integral Value	Error	Order
1	1.7440306508910550	1.4944564×10^{-1}	
2	1.9035081766468929	1.0031884×10^{-2}	3.8969564
4	1.8938597297032531	3.8343669×10^{-4}	4.7094603
8	1.8934767880912242	4.9507693×10^{-7}	9.5971199
16	1.8934762249053652	6.8108929×10^{-8}	2.8617369
32	1.8934762887314800	4.2828143×10^{-9}	3.9912129
64	1.8934762927462818	$2.6801250 \times 10^{-10}$	3.9981869
128	1.8934762929975435	$1.6750823 \times 10^{-11}$	3.9999964
256	1.8934762930132514	$1.0429435 \times 10^{-12}$	4.0054991
512	1.8934762930142324	$6.1950445 \times 10^{-14}$	4.0734026
1024	1.8934762930142981	$3.7747583 \times 10^{-15}$	4.0366585

The exponent in such error estimates is called the order of the method, because it, more than any other part of the estimate, controls the rate at which the quadrature formula will converge. The fourth column in Table 24.2 was computed as follows: Observe that the error is expected to be approximately ch^p for some unknown constant c . Take the estimated error in row k and divide it by the estimated error in row $k - 1$. This number should be approximately 2^{-p} , since the stepsize has been reduced by a factor of two from row to row. By taking logarithms, the value of p can be estimated as

$$p \approx \frac{\log e_{k-1} - \log e_k}{\log 2}.$$

This analysis shows, in the case of the arclength of the curve shown in Figure 24.5, that the expected rate of convergence for Simpson’s rule is actually observed in practice once h enters the asymptotic range for the error estimate. Thus, a good adaptive quadrature routine will be able to make useful estimates about the accuracy of the results by making use of this information. Required input accuracies can actually be achieved in practice because the theory does a good job of predicting the actual rate of convergence.

However, much can go wrong. Instead of the curve shown in Figure 24.5, consider the curve depicted in Figure 24.6. Like the first curve, this is a smooth, curvature-continuous curve, with no obvious geometric or algebraic singularities. Like the first, the curve is a parametric cubic. Unlike the first, it is made up of two polynomial pieces, and its knot sequence is given by $\{0, 0, 0, 0, 0.47, 1, 1, 1, 1\}$, with B-spline coefficients for the x component $\{0, 0.3, 0.5, 0.7, 1\}$ and B-spline coefficients for the y component equal to $\{0, 1, 1, 1, 0\}$.

Applying the same procedure as before, the length of this curve can be estimated using Simpson’s rule. That leads to the results shown in Table 24.3. This table shows very different results from those shown in Figure 24.5. To begin with, the error in the value of the integral for Simpson’s rule with 1024 subintervals is larger by a factor of nearly 3000.

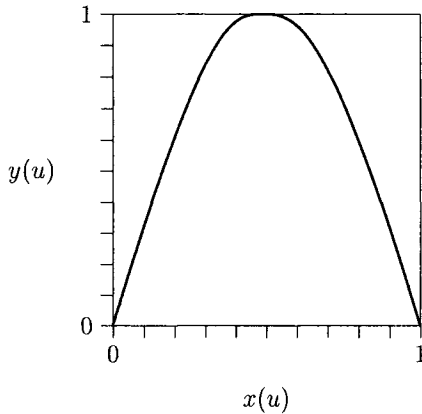


Figure 24.6. Another curve with unknown length

This is much larger than anything that can be accounted for by intermediate values of the fourth derivative of the integrand. Moreover, the estimated order of the method, the fourth column, is all over the place. It's very difficult to observe the predicted fourth order convergence of the method in the calculated integral estimates.

As with so many things in everyday life, the problem is that insufficient attention was paid to the fine print in the contract we signed with Simpson's rule before we started using it. In the derivation of the error estimate is an underlying assumption about the differentiability of the integrand. In particular, the actual formula depends on continuous fourth derivatives, while the order of the method depends on having continuous third derivatives. For the curve shown in Figure 24.6, the integrand for arc length has a jump discontinuity in the second derivative at parameter value $u = 0.47$. Thus, an important assumption on which the Simpson's rule error estimate is based has been violated, and the estimate itself cannot be proven or believed.

However, the results are very different when Simpson's rule is applied to the same curve a little differently. Table 24.4 contains the results for the length of the curve shown in Figure 24.6 again, but this time, the curve integral has been split into two pieces at its interior knot. Half of the Simpson's rule intervals are allocated to the first piece, while the other half of the intervals are allocated to the second piece. When the integral is broken at the discontinuity in second derivative, now the expected convergence rate can be observed in the results. Moreover, the actual error is lower for most of the entries in the table. In some cases, the error is as much as three orders of magnitude smaller for an equivalent number of function evaluations.

This example points out the importance of matching the differentiability of integrand to its assumed differentiability. The assumed differentiability of the quadrature formulas is often considerably higher than one might suppose, especially for Gauss quadrature formulas. For example, a seemingly innocent k point Gauss quadrature formula has an

Table 24.3
More estimates of curve length using Simpson's rule

Pieces	Integral	Error	Order
1	2.4981326614856338	1.7105619×10^{-1}	
2	2.3078041360542949	1.9272332×10^{-2}	3.1498673
4	2.3241119097377538	2.9645586×10^{-3}	2.7006420
8	2.3271127744132456	3.6306072×10^{-5}	6.3514627
16	2.3270757930144499	6.7532670×10^{-7}	5.7484815
32	2.3270764768619507	8.5208045×10^{-9}	6.3084521
64	2.3270764749041462	6.5629999×10^{-9}	3.7663424
128	2.3270764711423424	2.8011962×10^{-9}	1.2283124
256	2.3270764690880932	$7.4694695 \times 10^{-10}$	1.9069653
512	2.3270764682490546	$9.2091668 \times 10^{-11}$	3.0198632
1024	2.3270764683520975	$1.0951240 \times 10^{-11}$	3.0719764
2048	2.3270764683402407	$9.0549790 \times 10^{-13}$	3.5962391

Table 24.4
Still more estimates of curve length using Simpson's rule

Pieces	Integral	Error	Order
2	2.3103231743670891	1.6753294×10^{-2}	
4	2.3236486484941294	3.4278198×10^{-3}	2.2890816
8	2.3271117033668585	3.5235026×10^{-5}	6.6041372
16	2.3270763957527074	7.2588439×10^{-8}	8.9230548
32	2.3270764651729792	3.1681671×10^{-9}	4.5180195
64	2.3270764681427023	$1.9844393 \times 10^{-10}$	3.9968451
128	2.3270764683287486	$1.2397638 \times 10^{-11}$	4.0005942
256	2.3270764683403833	$7.6294526 \times 10^{-13}$	4.0223420
512	2.3270764683411107	$3.5527137 \times 10^{-14}$	4.4245862
1024	2.3270764683411551	$8.8817842 \times 10^{-15}$	2.0000000

error estimate that assumes that the integrand is $2k + 2$ times differentiable. Since most geometric modeling applications based on splines do not offer this much differentiability, some care must be taken when applying quadrature formulas.

As in the case of rootfinding, a weakness of parametric evaluator systems is exposed. Since the location of the derivative discontinuities is hidden from the geometric analysis routines in such systems, those systems are not able to align the quadrature formulas with the derivative discontinuities so that the error estimates of the quadrature formulas can be used reliably. Either very low order methods must be used, leading to inefficiencies because of the large number of function evaluations required, or the accuracy of the final result cannot be assured with any certainty.

Taking advantage of known derivative discontinuities is not always easy. Consider the problem of the determining the length of a curve which lies on a surface. Suppose that the surface is a tensor product spline surface, and that the description of the curve on it is given by a curve in parameter space for the surface. In other words, the surface is given by a mapping $F : [0, 1]^2 \rightarrow \mathbb{R}^3$, while the curve on the surface is given by a mapping $G : [0, 1] \rightarrow [0, 1]^2$. Then the composite mapping $F \circ G$ describes a curve on the surface whose length can be calculated. Suppose that the two components of G are given by u and v . Then the integral which describes the length of the curve is

$$\int_0^1 \|F_u u'(t) + F_v v'(t)\|_2 dt.$$

The integrand will have derivative discontinuities whenever the curve G crosses one of the knots for F as well as whenever G itself has a discontinuity.

As an example, consider the situation shown in Figure 24.7. Here, parameter space for a surface with four interior knots in the u direction and three knots in the v direction is shown. A curve through parameter space is shown, together with a collection of \bullet and a single \circ . The \circ represents a location on the curve in which the curve itself has a derivative discontinuity. Each \bullet represents a location at which the curve crosses one of the knot lines of the surface. Dependable schemes for calculating the length of that curve on the surface will break the curve at each \bullet or \circ so that the integrand is infinitely differentiable everywhere else. In this way, several smaller integrals are calculated, each of which comes with a reliable error estimate. When those values are summed, the result is an estimate of the value of the desired integral whose quality is known.

Of course, determining the locations of the \bullet is not easy. In particular, the two components u and v must each be set to each of the internal knot values, and the resulting nonlinear equation solved for all of its roots, each of which will determine the location of one \bullet . As discussed in the previous section, and as typical for much of geometry processing, the failure to find even a single root destroys the integrity of the computation. In this case, such failure means that the estimated error of the value of the curve length for the curve on the surface will not be valid. The estimated value of the integral may still be acceptable, but it's not possible to assert such a thing with any authority.

24.4. COMPUTING MASS PROPERTIES

This section describes how to compute mass properties of a boundary-represented (B-Rep) solid. The basic capability required in order to compute these properties is the ability to

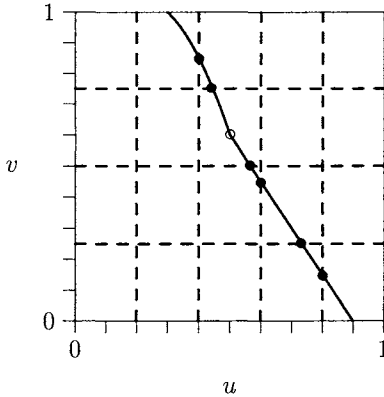


Figure 24.7. Location of derivative discontinuities for a curve on a surface

integrate a polynomial over the solid. A good method for accomplishing that is based on two applications of the divergence theorem.

Recall the divergence theorem:

$$\int_{\Omega} \nabla \cdot f = \int_{\partial\Omega} f \cdot n. \quad (24.8)$$

In words, this theorem says that the integral of the divergence of a vector valued function f over a region is equal to the integral over the boundary of the region of the inner product of f with the outward pointing unit normal to the region.

In the first application of this theorem in what follows, Ω will be treated as a solid defined in terms of the surfaces which bound it. For example, a cube would be defined in terms of the six rectangles which form its faces. Each face, in turn, can be defined using a parametric surface map $S : [0, 1]^2 \rightarrow \mathbb{R}^3$ whose outward pointing unit normal can be computed as

$$N(u, v) = \sigma \frac{S_u(u, v) \times S_v(u, v)}{\|S_u(u, v) \times S_v(u, v)\|}, \quad (24.9)$$

with σ a constant with value either -1 or 1 to settle orientation. In usual modeling situations, not all of the surface mapping will be used to describe a face of the solid. Such surfaces in which only a portion of the image of the surface mapping is used to define the boundary of a solid is called a trimmed surface. For example, the circular faces which form the ends of a bounded cylinder can be described by surface mappings which map the rectangular parameter space into a rectangular face properly positioned in space. Only a circular subset of that rectangle is required to bound the cylinder. A convenient means of specifying this subset is to keep track of that portion of parameter space which is its preimage. Interestingly, this preimage is itself a solid of one dimension lower than the one we started with which can be defined in terms of its boundary, in this case the curves in

parameter space which bound it. Each of these boundary curves can be defined using a parametric curve map $c : [0, 1] \rightarrow [0, 1]^2$. The two components of this map are typically labeled to correspond to parameter space for the surface, so

$$c(t) = \begin{pmatrix} u(t) \\ v(t) \end{pmatrix}. \quad (24.10)$$

If the curve map is oriented so that the active region lies to the left of the curve as it is followed, then the outward pointing unit normal to the active parametric region of the surface is given by

$$n(t) = \frac{\begin{pmatrix} v'(t) \\ -u'(t) \end{pmatrix}}{\|c'(t)\|}. \quad (24.11)$$

To describe an entire boundary of a solid, many boundary surfaces and boundary curves are usually needed. In what follows, assume that the number of boundary surfaces is given by n_s , and that the number of boundary curves for the i th boundary surface is given by n_i . The i th boundary surface map can be referred to as S_i , and the j th boundary curve of that map can be referred to as (u_{ij}, v_{ij}) . The normals to those maps can be similarly indexed. Lastly, assume that the entire solid is denoted by Ω , and that the union of the parametric preimages of all the pieces which make up its boundary are together denoted by $\partial\Omega$.

Computing mass properties of solids requires an ability to integrate polynomials over the solid. Thus, the crucial integral is

$$\int_{\Omega} x^{\alpha} y^{\beta} z^{\gamma}, \quad (24.12)$$

for non-negative integers α , β , and γ . Consider the vector valued function f defined by

$$f(x, y, z) = \begin{pmatrix} \frac{x^{\alpha+1} y^{\beta} z^{\gamma}}{3(\alpha+1)} \\ \frac{x^{\alpha} y^{\beta+1} z^{\gamma}}{3(\beta+1)} \\ \frac{x^{\alpha} y^{\beta} z^{\gamma+1}}{3(\gamma+1)} \end{pmatrix}. \quad (24.13)$$

The divergence of this function is the integrand in (24.12), so the divergence theorem can be applied to get

$$\sum_{i=1}^{n_s} \int_{\partial\Omega_i} \frac{1}{3} x_i(u, v)^{\alpha} y_i(u, v)^{\beta} z_i(u, v)^{\gamma} \left(\frac{x_i(u, v)}{\alpha+1}, \frac{y_i(u, v)}{\beta+1}, \frac{z_i(u, v)}{\gamma+1} \right) \cdot N_i(u, v) \|S_{iu}(u, v) \times S_{iv}(u, v)\|. \quad (24.14)$$

The scale factor at the end arises because of the parametrization of the boundary. This scale factor represents the measure of surface area with respect to the parametrization

of the boundary surface. Formally define a new function g_i to be the integrand of this integral, so that the integral (24.14) is just

$$\sum_{i=1}^{n_s} \int_{\partial\Omega_i} g_i, \quad (24.15)$$

where

$$g_i(u, v) = \frac{1}{3} x_i(u, v)^\alpha y_i(u, v)^\beta z_i(u, v)^\gamma \left(\frac{x_i(u, v)}{\alpha + 1}, \frac{y_i(u, v)}{\beta + 1}, \frac{z_i(u, v)}{\gamma + 1} \right) \cdot \sigma_i S_{iu}(u, v) \times S_{iv}(u, v). \quad (24.16)$$

Now consider a new function h_i which is defined to be the integral of g_i with respect to its first variable, i.e.

$$h_i(u, v) = \int_{u_0}^u g_i(\tau, v) d\tau. \quad (24.17)$$

Thus, g_i is the derivative of h_i with respect to its first variable, so the divergence theorem can be applied once again over the regions which are the parametric domains of each of the surfaces S_i to get integrals around the boundary of each of them, i.e. along the curves (u_{ij}, v_{ij}) . This transforms (24.15) into

$$\sum_{i=1}^{n_s} \sum_{j=1}^{n_i} \int_{t_{ij0}}^{t_{ij1}} \int_{u_{ij0}}^{u_{ij}(t)} g_i(\tau, v_{ij}(t)) v'_{ij}(t) d\tau dt. \quad (24.18)$$

This sum will produce the desired value of (24.12).

Equipped with an ability to evaluate (24.12), mass properties for any B-rep solid can be computed. The easiest mass property to compute is the volume, which is given by

$$v(\Omega) = \int_{\Omega} 1. \quad (24.19)$$

The mass is just the product of this volume times the density of the solid:

$$m(\Omega) = \rho(\Omega) v(\Omega). \quad (24.20)$$

Computing the center of mass is nearly as easy. The three coordinates of the center of mass are given by

$$\begin{aligned} \bar{x}(\Omega) &= \frac{\int_{\Omega} x}{v(\Omega)} \\ \bar{y}(\Omega) &= \frac{\int_{\Omega} y}{v(\Omega)} \\ \bar{z}(\Omega) &= \frac{\int_{\Omega} z}{v(\Omega)}. \end{aligned} \quad (24.21)$$

Now consider the moment of inertia about an axis w through the origin. If r is the distance of any point $x = (x_1, x_2, x_3)$ from this axis, then the inertia can be computed as

$$\begin{aligned} \int_{\Omega} r^2 &= \int_{\Omega} x \cdot x - (x \cdot w)^2 \\ &= w \cdot Jw, \end{aligned} \quad (24.22)$$

where J is the 3×3 matrix whose entries are given by

$$J_{ij} = \int_{\Omega} \delta_{ij} x \cdot x - x_i x_j. \quad (24.23)$$

In this expression, δ_{ij} is the usual Kronecker delta. The matrix J is called the inertia tensor, and its entries can be computed by evaluating the integrals

$$M_{ij} = \int_{\Omega} x_i x_j. \quad (24.24)$$

Given the inertia tensor J , it is possible to compute the moment of inertia $I(w, p)$ about any point p and any rotational axis w by making use of the parallel axis theorem [3], which states

$$I(w, p) = w \cdot Jw + v(\Omega)(p - 2\bar{x}(\Omega)) \cdot (p - (p \cdot w)w). \quad (24.25)$$

REFERENCES

1. C. de Boor. *Elementary Numerical Analysis: An Algorithmic Approach*. McGraw-Hill Book Company, New York: 1980.
2. Boeing Information and Support Services. *DT_NURBS Spline Geometry Subprogram Library Users' Manual, Version 3.4*. Carderock, Maryland: CARDEROCKDIV-94/000, Carderock Division, Naval Surface Warfare Center, 1997.
3. R. Feynman, R.B. Leighton, and M. Sands. *The Feynman Lectures on Physics*. Addison-Wesley, Reading, Massachusetts: 1963.
4. T.A. Grandine. Computing zeroes of spline functions. *Computer Aided Geometric Design*, 6:129-136, 1989.
5. T.A. Grandine and F.W. Klein IV. A new approach to the surface intersection problem. *Computer Aided Geometric Design* 14:111-134, 1997.
6. E.R. Hansen. Interval forms of Newton's method. *Computing*, 20:153-163, 1978.
7. J.M. Ortega and W.C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Classics in Applied Mathematics 30, Philadelphia: Society for Industrial and Applied Mathematics, 2000.
8. E.C. Sherbrooke and N.M. Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *Computer Aided Geometric Design*, 10:379-405, 1993.

Chapter 25

Intersection Problems

Nicholas M. Patrikalakis and Takashi Maekawa

25.1. INTRODUCTION

Intersections are fundamental in computational geometry, geometric modeling and design, analysis and manufacturing applications [4,33,67]. Examples of intersection problems include: (1) Contouring of surfaces through intersection with a series of parallel planes or coaxial cylinders for visualization. (2) Numerical control machining (milling) involving intersection of offset surfaces with a series of parallel planes, to create machining paths for ball (spherical) cutters. (3) Representation of complex geometries in the *Boundary Representation (B-rep)* scheme using a process called *boundary evaluation*, in which the Boundary Representation is created by evaluating a Constructive Solid Geometry (CSG) model of the object. During this process, intersections of the surfaces of primitives must be found during Boolean operations (union, intersection, difference) between primitives.

All above operations involve intersections of surfaces to surfaces. In order to solve general surface to surface (S/S) intersection problems, the following five auxiliary intersection problems need to be considered: point/point (P/P), point/curve (P/C), point/surface (P/S), curve/curve (C/C), curve/surface (C/S). All above six types of intersection problems are also useful in shape interrogation, robotics, collision avoidance, manufacturing simulation, scientific visualization, etc. When the geometric elements involved in intersections are nonlinear (curved), intersection problems typically reduce to solving systems of nonlinear equations, which may be either polynomial or more general functions.

Solution of nonlinear systems is a complex topic of numerical analysis and there are specialized textbooks on the topic [16,66]. However, geometric modeling applications pose severe robustness, accuracy, automation, and efficiency requirements on solvers of nonlinear systems. Therefore, geometric modeling researchers have developed specialized solvers to address these requirements explicitly using geometric formulations.

When studying intersection problems, the type of curves and surfaces that we consider can be classified into two types: (1) Rational polynomial parametric (RPP), (2) Implicit

algebraic (IA). Non-Uniform Rational B-Spline (NURBS) curves and surfaces can be subdivided into RPP curves and surfaces, and analyzed in a similar manner. Procedural curves and surfaces defined by means of an evaluation method without explicit use of the specific analytic properties of the defining formula are not treated here. Procedural curves and surfaces include offsets, evolutes, blends and generalized cylinders. A detailed treatment of intersection problems including procedural curves and surfaces can be found in a recent textbook [68]. In this handbook we only deal with S/S intersection problems and the auxiliary C/S problems involving RPP and IA geometries. P/P, P/C, P/S, C/C intersection problems are analyzed in [68] in some detail.

This chapter is organized as follows: In Section 25.2 we classify the intersection problems by dimension, type of geometric specification, and the number system used. Section 25.3 overviews solution methods for systems of nonlinear polynomial equations. Section 25.4 treats curve/surface intersection problems followed by surface/surface intersection problems in Section 25.5. Section 25.6 concludes this chapter and summarizes some current and future research directions.

25.2. CLASSIFICATION OF INTERSECTION PROBLEMS

The fundamental issue in intersection problems is the efficient discovery and description of *all* features of the solution with high precision commensurate with the tasks required from the underlying geometric modeler [67]. Reliability of intersection algorithms is a basic prerequisite for their effective use in any geometric modeling system and is closely associated with the way features of the solution such as constrictions (near singular or singular situations), small loops and partial surface overlap are handled. The solutions resulting from most present techniques, implemented in practical systems, are further complicated by imprecisions introduced by numerical errors present in finite precision computations. Intersection problems can be classified according to the dimension of the problems, the type of geometric equations involved in defining the various geometric elements, the number system in which the input is expressed, and the number system used in algorithm implementation. Such intersection problem classification is addressed in the next three subsections.

25.2.1. Classification by dimension

Using the abbreviation in Section 25.1, intersection problems can be classified in three subcategories, where one intersecting entity is a point or curve or surface as: (1) P/P, P/C, P/S, (2) C/C, C/S, (3) S/S.

25.2.2. Classification by type of geometric specification

In this subsection, we classify the various types of geometric specification of points, curves and surfaces that we will use in formulating various intersection problems:

1. Points

Explicit: $\mathbf{r}_0 = (x_0, y_0, z_0)^T$, where superscript T denotes transpose of a vector.

Implicit algebraic: Intersection of three implicit surfaces, or equivalently $f(\mathbf{r}) = g(\mathbf{r}) = h(\mathbf{r}) = 0$ where f, g, h are polynomial functions and $\mathbf{r} = (x, y, z)^T$.

2. *Curves*

Parametric: (Rational) (piecewise) polynomial: Bézier, rational Bézier, B-spline, NURBS, $\mathbf{r} = \mathbf{r}(t)$, $0 \leq t \leq 1$.

Implicit algebraic: A 2-D planar curve is given by $z = 0$, $f(x, y) = 0$, while 3-D space curve is given by intersection of two implicit surfaces $f(\mathbf{r}) = g(\mathbf{r}) = 0$ where f and g are polynomial functions.

3. *Surfaces*

Parametric: (Rational) (piecewise) polynomial: Bézier, rational Bézier, B-spline, NURBS, $\mathbf{r} = \mathbf{r}(u, v)$, $0 \leq u, v \leq 1$.

Implicit algebraic: $f(\mathbf{r}) = 0$ where f is a polynomial function.

25.2.3. Classification by number system

In our discussion of intersection problems, we will refer to various classes of numbers:

1. Rational numbers, m/n , $n \neq 0$, where m, n are integers.
2. Floating point (FP) numbers in a computer (which are a subset of rational numbers)
3. Algebraic numbers (roots of polynomials with integer coefficients).
4. Real numbers, e.g. transcendental numbers such as e , π , trigonometric, etc.
5. Interval numbers, $[a, b]$, where a, b are real numbers.
6. Rounded interval numbers, $[c, d]$, where c, d are FP numbers.

Issues relating to floating point and interval numbers affecting the robustness of intersection algorithms are addressed in [1,20,35,34,88] in the context of nonlinear solvers.

25.3. OVERVIEW OF NONLINEAR SOLVERS

Curves and surfaces in CAD/CAM systems are usually represented by piecewise polynomial equations of various types. Therefore the governing equations for intersection problems reduce to solving systems of nonlinear polynomial equations.

25.3.1. Brief review of local and global methods

One of the most popular local methods in solving nonlinear equation systems is the Newton-type method. Newton-type methods are based on local linearization and conceptually they are very simple. They are designed to compute roots based on initial approximations. Advantages of the Newton's method are its quadratic convergence and its easiness for implementation. Disadvantages are that for each root a good initial approximation is required, otherwise the method may diverge. Also the method cannot by itself provide full assurance that all roots have been found.

Global solution methods are designed to compute all roots in some area of interest. In recent computational algebraic geometry related research, three classes of methods for the computation of solutions of nonlinear polynomial systems can be distinguished [68]: (1) algebraic and hybrid techniques, (2) homotopy (continuation) methods, (3) subdivision methods. We will briefly review these three types of techniques.

Algebraic and hybrid techniques

Algebraic techniques for solving a nonlinear polynomial system are based on *elimination theory*. This theory deals with the problem of eliminating one or more variables from a system of polynomial equations, thus reducing the given problem to a problem of higher degree but in *fewer* variables. There are basically two fundamental approaches in elimination theory: (1) Resultants, and (2) Gröbner bases (see Chapter 15 of this handbook [85] for more details). Both of the above operate ideally in a symbolic algebra environment, and the coefficients of the polynomials involved are either rational or real algebraic numbers. There are several algorithms for solving nonlinear polynomial systems using the above approaches. All the algorithms are based on some fundamental algorithm that “finds” all the roots, real and complex, of a univariate polynomial. The word “finds” means, either the algorithm isolates the roots using intervals and rectangles, or encodes them as algebraic numbers, for further manipulation. Let $f(x)$ be a polynomial with integer coefficients of degree m , d be a bound for the size of the coefficients of $f(x)$, and $L(d)$ be the number of binary digits of d . Then, the (worst) running times of real root finding algorithms are functions of m , $L(d)$ and are given in [14]. On the other hand, bisection methods for finding all roots of f , real and complex with similar running times, can be found in [78,97]. As it can be seen from the computing times found in [14,78,97], there is an enormous coefficient growth of all the quantities involved along the way (requiring significant computer memory). The latter is one of the most serious problems that all the algorithms using these techniques suffer from.

Resultant type algorithms: A *resultant* is a function of the coefficients of a given system of polynomials and when it is zero it provides an algebraic criterion for determining when this polynomial system has a solution. Resultants can be classified as classical, like the Sylvester, Bezout, Macaulay and u resultants, and non-classical like the *sparse* resultants. A good introduction to resultants and applications can be found in [17,71,90,91] and Chapter 15 of this handbook [85].

Algorithms based on resultant computation have been presented in [9,10,36,57,92]. They work well on systems with a small number of solutions M . However, on systems with large M , these algorithms suffer from efficiency problems. The main reason for that is that finding roots of high degree univariate polynomials can be a very slow procedure, as discussed above, due to the use of exact arithmetic.

Gröbner bases-type algorithms: The theory of Gröbner bases was developed by Buchberger [7]. Gröbner bases are very special and useful bases (generator sets) for a special class of subsets of polynomial rings in l variables, called polynomial ideals. They are named after Gröbner who was Buchberger’s thesis advisor. Gröbner bases can be thought of as a generalization of Euclid’s algorithm for computing the greatest common divisor of two polynomials and of the Gauss triangularization algorithm for linear systems.

The usefulness of Gröbner basis for solving nonlinear polynomial systems comes from the fact that, whenever the system has a finite number of solutions, Gröbner basis provides an equivalent system of triangular form. Algorithms using Gröbner bases use the above fact, and appear in [8,22,43,47,99]. Using Gröbner bases, polynomial systems are converted to polynomial triangular systems, which can be solved by backward substitution, much in the manner of the Gauss triangularization algorithm for linear systems.

If the system has a finite number of solutions in the affine plane, as well as in the

projective plane, then a Gröbner basis can be computed in $O(m^l)$ time, where m is the highest degree among the polynomials and l is the number of variables. In case, however, that the system is not zero-dimensional at infinity, the time becomes $O(m^{l^2})$. These bounds do not take into account the coefficient growth. Gröbner basis algorithms work well on systems with few roots. This is one reason they have been considered seriously as a practical equation-solving tool. But when their complexity is measured as a function of the number of solutions, their performance is poor. As reported in [58], these algorithms frequently exhaust memory and computer resources even for low number of equations n and variables l (e.g. $n, l \leq 5$) and moderate degrees m . To overcome this difficulty, algorithms that combine resultant and linear algebra techniques are more promising concerning efficiency [2,58,59,65]. These algorithms are generally *hybrid* and are based on algebraic and numerical analysis methods. In particular, this approach based on resultants transforms the problem into a sequence of eigenvalue problems. This method has found extensive application in various types of intersection problems [42].

Homotopy (continuation) methods

Homotopy methods [23,44,104] are mathematically elegant, but unfortunately, investigation of such methods indicates that they tend to be numerically ill-conditioned. If we try to get around this problem by implementing the algorithm in rational arithmetic, we end up with enormous memory requirements because we have to solve large systems of complex initial value problems (IVP). Interval methods can be applied to the solution of these IVPs but they can be slow in practice [58].

Subdivision methods

Subdivision methods [46,64,74,88] are generally efficient (in finding simple intersections) and stable. Therefore, they are the most frequently used methods in practice. As we will see, they can be combined with interval methods to numerically guarantee that certain subdomains do not contain solutions. Interval Newton methods [6,24,28,29,37,62] are a promising class of subdivision methods. However, the subdivision methods are not as general as algebraic methods, since they are only capable of isolating zero-dimensional solutions. Furthermore, although the chances, that all roots have been found, increase as the resolution tolerance is lowered, there is no certainty that each root has been extracted/isolated. Subdivision methods typically do not provide a guarantee as to how many roots there may be in the remaining subdomains. However, if these subdomains are very small, the existence of a (single) root within these subdomains is a typical assumption. Lastly, subdivision techniques provide no explicit information about root multiplicities without additional computation. Despite these drawbacks, subdivision methods are very useful in practice and are further described below.

25.3.2. IPP algorithm

In this section we introduce an iterative global root-finding algorithm for n -dimensional nonlinear polynomial equation systems, which belongs to the subdivision class of methods, called Projected Polyhedron (PP) algorithm developed by Sherbrooke and Patrikalakis [88]. It is easy to visualize and simple in that it only requires two straightforward algorithms in order to implement it: one for subdividing multivariate polynomials in Bernstein

form, and one for finding the convex hull of a two-dimensional set of points. This algorithm is an extension and generalization of earlier adaptive subdivision algorithms: for $n = 1$ used in finding the real roots and extrema of a polynomial within an interval by Lane and Riesenfeld [46], and for $n = 2$ used in intersecting rays with trimmed rational polynomial surface patches by Nishita et al. [64], a method known as Bézier clipping. The PP algorithm has found many applications in shape interrogation problems and its convergence and complexity properties are developed in [88].

Suppose we are given a set of n nonlinear polynomials f_1, f_2, \dots, f_n , each of which is a function of x_1, x_2, \dots, x_l . Let $m_i^{(k)}$ denote the degree in x_i of polynomial f_k , so that the multi-index $M^{(k)} = (m_1^{(k)}, m_2^{(k)}, \dots, m_l^{(k)})$ describes all the degree information of f_k . Furthermore, suppose we are given an l -dimensional rectangular subset of \mathbf{R}^l

$$B = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_l, b_l]. \tag{25.1}$$

A priori knowledge of B is one of the main features of geometric modeling and shape interrogation problems. We wish to find all points $\mathbf{x} = (x_1, x_2, \dots, x_l) \in B$ such that

$$f_1(\mathbf{x}) = f_2(\mathbf{x}) = \dots = f_n(\mathbf{x}) = 0. \tag{25.2}$$

By making the *affine parameter transformation* [18] $x_i = a_i + u_i(b_i - a_i)$ for each i between 1 and l inclusive, we simplify the problem to one of determining all $\mathbf{u} \in [0, 1]^l$ such that

$$f_1(\mathbf{u}) = f_2(\mathbf{u}) = \dots = f_n(\mathbf{u}) = 0. \tag{25.3}$$

Since all of the f_k are polynomial in each of the l independent parameters, a simple *change of basis* [18] allows us to express them in the multivariate Bernstein basis, which has better numerical stability under perturbation of its coefficients than the power basis [20], and in addition permits transformation of an algebraic problem to a geometric problem. In other words, for each f_k there exists an l -dimensional array of real coefficients $w_{i_1 i_2 \dots i_l}^{(k)}$ such that for each $k \in \{1, \dots, n\}$

$$f_k(\mathbf{u}) = \sum_{i_1=0}^{m_1^{(k)}} \sum_{i_2=0}^{m_2^{(k)}} \dots \sum_{i_l=0}^{m_l^{(k)}} w_{i_1 i_2 \dots i_l}^{(k)} B_{i_1, m_1^{(k)}}(u_1) B_{i_2, m_2^{(k)}}(u_2) \dots B_{i_l, m_l^{(k)}}(u_l). \tag{25.4}$$

The notation in (25.4) may be simplified by letting $I = (i_1, i_2, \dots, i_l)$, $M^{(k)} = (m_1^{(k)}, m_2^{(k)}, \dots, m_l^{(k)})$ and writing (25.4) in the equivalent form

$$f_k(\mathbf{u}) = \sum_I w_I^{(k)} B_{I, M^{(k)}}(\mathbf{u}). \tag{25.5}$$

Provided that conversion of the problem to the Bernstein basis is exact or sufficiently accurate, the use of the Bernstein basis in conjunction with subdivision is known to be numerically stable [20]. The conversion process itself may be numerically ill-conditioned [21]; therefore, we recommend that the problem be formulated in the Bernstein basis from the very beginning and exactly, if possible. If necessary, polynomials may be converted

from the multivariate power basis to the multivariate Bernstein basis using the following formula

$$c_{i_1 i_2 \dots i_l}^B = \sum_{j_1=0}^{i_1} \sum_{j_2=0}^{i_2} \dots \sum_{j_l=0}^{i_l} \frac{\binom{i_1}{j_1} \binom{i_2}{j_2} \dots \binom{i_l}{j_l}}{\binom{m_1}{j_1} \binom{m_2}{j_2} \dots \binom{m_l}{j_l}} c_{j_1 j_2 \dots j_l}^M \tag{25.6}$$

where $c_{i_1 i_2 \dots i_l}^B$ and $c_{j_1 j_2 \dots j_l}^M$ are the coefficients of polynomials in Bernstein and power bases, respectively.

We now restate the problem as the intersection of the *graphs* of the f_k (each of which is a hypersurface in \mathbf{R}^{l+1}) and the hyperplane $u_{l+1} = 0$ of \mathbf{R}^{l+1} . This idea is designed to impart geometrical significance to the coefficients of the polynomials and to the solution process. Let us build a graph \mathbf{f}_k for each f_k :

$$\mathbf{f}_k(\mathbf{u}) = (u_1, u_2, \dots, u_l, f_k(\mathbf{u})) = (\mathbf{u}, f_k(\mathbf{u})) \tag{25.7}$$

Clearly, (25.3) is satisfied by the point \mathbf{u} if and only if

$$\mathbf{f}_1(\mathbf{u}) = \mathbf{f}_2(\mathbf{u}) = \dots = \mathbf{f}_n(\mathbf{u}) = (\mathbf{u}, 0) \tag{25.8}$$

Using the linear precision property of the Bernstein basis [18], we obtain an equivalent expression for each of the u_j in (25.7):

$$u_j = \sum_I^{M^{(k)}} \frac{i_j}{m_j^{(k)}} B_{I, M^{(k)}}(\mathbf{u}) \tag{25.9}$$

Substituting (25.9) into (25.7) gives a more useful representation for the \mathbf{f}_k :

$$\mathbf{f}_k(\mathbf{u}) = \sum_I^{M^{(k)}} \mathbf{v}_I^{(k)} B_{I, M^{(k)}}(\mathbf{u}) \tag{25.10}$$

where

$$\mathbf{v}_I^{(k)} = \left(\frac{i_1}{m_1^{(k)}}, \frac{i_2}{m_2^{(k)}}, \dots, \frac{i_l}{m_l^{(k)}}, w_I^{(k)} \right) \tag{25.11}$$

The $\mathbf{v}_I^{(k)}$ are called the *control points* of \mathbf{f}_k . Using the parametric hypersurfaces \mathbf{f}_k instead of the real-valued f_k permits use of the powerful *convex-hull property* of the multivariate Bernstein basis.

We assume we are given n nonlinear polynomial equations with l variables in the power basis, where $n \geq l$, and a box $B = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_l, b_l]$, in which we need to determine the roots of the given system. In this case we first scale the box by performing an appropriate affine parameter transformation described above to the functions f_k , so that the box becomes $[0, 1]^l$. Next we express the transformed nonlinear polynomial equations in the multivariate Bernstein basis using (25.6). Now we summarize the PP algorithm.

1. Using the convex hull property, find a sub-box of $[0, 1]^l$ which contains all the roots. The essential idea behind the box generation scheme in this algorithm is to transform a complicated $l + 1$ -dimensional problem into a series of l two-dimensional problems. Suppose \mathbf{R}^{l+1} can be coordinatized with the u_1, u_2, \dots, u_{l+1} axes; we can then employ these steps:
 - (a) Project the $\mathbf{v}_I^{(k)}$ of all of the \mathbf{f}_k into l different coordinate planes; specifically, the (u_1, u_{l+1}) -plane, the (u_2, u_{l+1}) -plane, and so on, up to the (u_l, u_{l+1}) plane.
 - (b) In each one of these planes,
 - i. Construct n two-dimensional convex hulls. The first is the convex hull of the projected control points of \mathbf{f}_1 , the second is from \mathbf{f}_2 and so on.
 - ii. Intersect each convex hull with the horizontal axis (that is, $u_{l+1} = 0$). Because the polygon is convex, the intersection may be either a closed interval (which may degenerate to a point) or empty. If it is empty, then no root of the system exists within the given search box.
 - iii. Intersect the intervals with one another. Again, if the result is empty, no root exists within the given search box.
 - (c) Construct an l -dimensional box by taking the Cartesian product of each one of these intervals in order. In other words, the u_1 side of the box is the interval resulting from the intersection in the (u_1, u_{l+1}) -plane, and so forth.
2. Using the scaling relationship between our current box and the initial box of search, see if the new sub-box represents a sufficiently small box in \mathbf{R}^l . If it does not, then go to step 3. If it does, then check the convex hulls of the hypersurface in the new box. If the convex hulls cross each variable axis, conclude that there is a root or at least an approximate root in the new box, and put the new box into a root list. Otherwise the new box is discarded.
3. If any dimension of this sub-box is not much smaller than 1 unit in length (i.e., the box has not decreased much in size along one or more sides), split the box evenly along each dimension which is causing trouble (not reducing in size). Continue on to the next iteration with several independent sub-problems.
4. If none of the boxes is left, then the root-finding process is over. Otherwise, perform an appropriate affine parameter transformation to the functions f_k , so that the box becomes $[0, 1]^l$, and go back to step 1 for each new box. This transformation can be performed with the multivariate de Casteljau subdivision algorithm which is an extension of similar algorithms for 1 and 2 dimensions given in [18]. However, keep track of the scaling relationship between this box and the initial box of search.

If we assume that each equation in (25.2) is of degree m in each variable and the system is n -dimensional, then the total asymptotic time per step is of $O(nlm^{l+1})$. The number of steps depends primarily on the accuracy required [88]. The PP algorithm achieves quadratic convergence in one dimension, while for higher dimensions, it exhibits at best linear convergence [88]. Once roots have been isolated via the PP algorithm, local

quadratically convergent Newton-type algorithms can be used to compute the roots to high precision more efficiently. An extension of the algorithm described above for a set of simultaneous piecewise polynomial nonlinear equations expressed in terms of tensor product B-splines can be found in Chapter 24 of this handbook [25]. A novel feature of this extension is the normalization of the equations in the range $[-1,1]$ and normalization of the knot vector in each subdomain in the range $[0,1]$ at each iteration step of the process to capitalize on the higher density of floating point numbers in this range, thereby improving numerical robustness of the algorithm.

Geometric modeling systems for curved objects typically operate in *floating point arithmetic* (FPA). Arithmetic operations, especially division, in FPA lead to significant numerical errors. Division operation can be avoided by four-dimensional homogeneous processing proposed by Yamaguchi [63,102]. CAD systems frequently fail as a result of the *limited precision* that is inherent to the internal representation of floating point numbers [31]. To remedy such problems interval arithmetic research in geometric modeling has become quite active. An *interval* is a set of real numbers defined below [62]:

$$[a, b] = \{x | a \leq x \leq b\}. \tag{25.12}$$

If floating point arithmetic is used to evaluate these interval numbers there is no guarantee that the roundings of the bounds are performed conservatively.¹ Rounded interval arithmetic (RIA) [1,55] ensures that the computed end points always contain the exact interval as follows:

$$\begin{aligned} [a, b] + [c, d] &= [(a + c) - \epsilon_\ell, (b + d) + \epsilon_u], \\ [a, b] - [c, d] &= [(a - d) - \epsilon_\ell, (b - c) + \epsilon_u], \\ [a, b] \cdot [c, d] &= [\min(a \cdot c, a \cdot d, b \cdot c, b \cdot d) - \epsilon_\ell, \max(a \cdot c, a \cdot d, b \cdot c, b \cdot d) + \epsilon_u], \\ [a, b] / [c, d] &= [\min(a/c, a/d, b/c, b/d) - \epsilon_\ell, \max(a/c, a/d, b/c, b/d) + \epsilon_u], \end{aligned} \tag{25.13}$$

where ϵ_ℓ and ϵ_u are the units-in-last-place denoted by ulp_ℓ and ulp_u for each separate floating point number resulting from the floating point operations giving the lower and upper bounds in (25.13). When performing standard operations for interval numbers using RIA, the lower bound is extended to include its previous consecutive floating-point number, which is smaller than the lower bound by ulp_ℓ . Similarly, the upper bound is extended by ulp_u to include its next consecutive number. Thus, the width of the result is enlarged by $ulp_\ell + ulp_u$ and the result will be reliable in subsequent operations.

Maekawa and Patrikalakis [55] extended the PP algorithm to operate in rounded interval arithmetic in order to find all the roots of a polynomial system *robustly*, which we refer to as *Interval Projected Polyhedron (IPP) algorithm*. Rounded interval arithmetic can be implemented effectively in object-oriented languages such as C++ [1]. Other than overloading the arithmetic operations from FP to interval, we need to pay attention in intersecting each convex hull with the horizontal axis. The computed parametric values result in interval numbers $u_{low} = [u_a, u_b]$ and $u_{up} = [u_c, u_d]$. We simply replace them by $u_{low} = [u_a, u_a]$ and $u_{up} = [u_d, u_d]$ to keep the parameter as real numbers or in other words degenerate interval numbers.

¹This statement is true only for the default IEEE-754 rounding mode of *round towards nearest*. The subject of hardware rounding modes is discussed in [1,68].

Errors are first introduced during the formulation of the governing equations for intersection problems. Formulation of the governing nonlinear polynomial equation systems in multivariate Bernstein form for shape interrogation usually involves arithmetic operations in Bernstein form [21,72] starting from the given input Bézier curve or surface. To eliminate or control such errors, we suggest [55]:

- Use of *rational arithmetic* (RA) or *rounded interval arithmetic* (RIA), if the control points of the given curve or surface are FP numbers to maintain a pristine or guaranteed precision statement of the problem, respectively.
- Use of RIA if the control points of the given curve or surface are real numbers to avoid any numerical contamination by standard FP arithmetic. This happens, for example, when the curve or surface is rotated, so that the control points may involve transcendental numbers (e.g. e , π , values of trigonometric function, radicals of rational numbers, etc.), which cannot be processed by RA. In this case RIA can be used to maintain a guaranteed precision statement of the problem.
- Conversion of the coefficients of the nonlinear equations in Bernstein form into intervals with FP number boundaries, if rational arithmetic is used in the formulation.

25.4. CURVE/SURFACE INTERSECTION

Curve to surface intersections are very useful as auxiliary problems in solving surface to surface intersections. When the curve is a straight line, the curve/surface intersection is also useful in ray tracing for visualization, and in point classification in solid modeling. In Sections 25.4.1 to 25.4.4 several of the most frequent curve to surface intersection problems namely, RPP/IA, RPP/RPP, IA/IA, IA/RPP are analyzed. The remaining cases are not discussed, but the reader should be able to address them via the methods of this section (see also [68]). We will start with RPP curve to IA surface intersection (RPP/IA), which is quite representative of the complexities of intersection problems.

25.4.1. RPP curve/IA surface intersection

This intersection problem is defined as:

$$\mathbf{r}(t) = \left(\frac{X(t)}{W(t)}, \frac{Y(t)}{W(t)}, \frac{Z(t)}{W(t)} \right)^T \cap f(\mathbf{r}) = 0, \quad (25.14)$$

where $X(t)$, $Y(t)$, $Z(t)$, $W(t)$ are polynomials of degree n . Let us consider an implicit algebraic surface of total degree m

$$f(x, y, z) = \sum_{i=0}^m \sum_{j=0}^{m-i} \sum_{k=0}^{m-i-j} c_{ijk} x^i y^j z^k = 0. \quad (25.15)$$

We substitute $x = \frac{X(t)}{W(t)}$, $y = \frac{Y(t)}{W(t)}$ and $z = \frac{Z(t)}{W(t)}$ into the implicit equation and multiply by $W^m(t)$ leading to

$$F(t) = \sum_{i=0}^m \sum_{j=0}^{m-i} \sum_{k=0}^{m-i-j} c_{ijk} X^i(t) Y^j(t) Z^k(t) W^{m-i-j-k}(t) = 0, \quad (25.16)$$

of degree $\leq mn$ in t . Therefore the problem of intersection is equivalent to finding the real roots of $F(t)$ in $0 \leq t \leq 1$. The most usual form of $F(t)$ is the power basis. The coefficients can be evaluated symbolically by substitution and collection of terms. This can be readily done in a standard symbolic manipulation program such as MATHEMATICA [100], MAPLE [11] etc. Such programs are oriented to processing rational numbers exactly.

An alternate basis for the representation of $F(t) = 0$ is the Bernstein basis, which leads to better stability for its real roots under perturbations of its coefficients than the power form [20]. Here the conversion is done exactly using rational arithmetic (given that the conversion itself is not in general numerically well-conditioned [21]). By the use of linear precision property

$$t = \sum_{i=0}^{mn} \frac{i}{mn} B_{i,mn}(t), \tag{25.17}$$

we can construct a graph which is a degree mn Bézier curve

$$\mathbf{f}(t) = (t, F(t))^T = \sum_{i=0}^{mn} \binom{mn}{c_i} B_{i,mn}(t). \tag{25.18}$$

Now we can apply the IPP algorithm introduced in Section 25.3 which converts the problem of finding roots of polynomials into a problem of finding the intersection of the Bézier curve with the parameter axis.

25.4.2. RPP curve/RPP surface intersection

The intersection problem between a rational polynomial parametric curve and a rational polynomial parametric surface is defined as:

$$\begin{aligned} \mathbf{r} = \mathbf{r}_1(t) &= \left(\frac{X_1(t)}{W_1(t)}, \frac{Y_1(t)}{W_1(t)}, \frac{Z_1(t)}{W_1(t)} \right)^T, \quad 0 \leq t \leq 1, \\ \cap \quad \mathbf{r} = \mathbf{r}_2(u, v) &= \left(\frac{X_2(u, v)}{W_2(u, v)}, \frac{Y_2(u, v)}{W_2(u, v)}, \frac{Z_2(u, v)}{W_2(u, v)} \right)^T, \quad 0 \leq u, v \leq 1. \end{aligned} \tag{25.19}$$

The equation consists of three nonlinear equations $\mathbf{r}_1(t) = \mathbf{r}_2(u, v)$ in three unknowns t, u, v , which can be converted to a nonlinear polynomial system and solved via the IPP algorithm. A preprocessing step of checking bounding boxes for absence of intersection is helpful. Implicitization [83] of $\mathbf{r}_2(u, v)$ in rational arithmetic, when possible, (which is recommended for low degree surfaces) reduces this problem to the RPP/IA curve to surface intersection problem described in Section 25.4.1.

25.4.3. IA curve/IA surface intersection

Implicit algebraic curve and implicit algebraic surface intersection problem is defined as:

$$\underbrace{f(\mathbf{r}) = g(\mathbf{r})}_{curve} = \underbrace{h(\mathbf{r})}_{surface} = 0. \tag{25.20}$$

The formulation comprises three nonlinear polynomial equations in three unknowns, the components of \mathbf{r} . Possible solution approaches include elimination methods [83], Newton's and minimization methods with objective function $F(\mathbf{r}) = f^2 + g^2 + h^2$, and the IPP algorithm.

25.4.4. IA curve/RPP surface intersection

The implicit algebraic curve and rational polynomial parametric surface intersection is defined as:

$$f(\mathbf{r}) = g(\mathbf{r}) = 0 \cap \mathbf{r} = \mathbf{r}(u, v) = \left(\frac{X(u, v)}{W(u, v)}, \frac{Y(u, v)}{W(u, v)}, \frac{Z(u, v)}{W(u, v)} \right)^T, 0 \leq u, v \leq 1. \quad (25.21)$$

By substituting $\mathbf{r} = \mathbf{r}(u, v)$ into $f(\mathbf{r}) = 0$ and $g(\mathbf{r}) = 0$ we obtain two algebraic curves $F(u, v) = 0$ and $G(u, v) = 0$. This formulation reduces to IA/IA curve intersection problem which can be solved by the IPP algorithm (see also [68] for more detail). A discussion of algebraic curve properties is given in Section 25.5.1.

25.5. SURFACE/SURFACE INTERSECTIONS

The solution of a surface/surface intersection problem may be empty, or include a curve (possibly made of several branches), a surface patch, or a point. In Sections 25.5.1 to 25.5.3 several of the most frequent surface to surface intersection problems, namely RPP/IA, RPP/RPP, and IA/IA are analyzed (see also [68] for a more complete discussion). Conceptually, RPP/IA surface intersection is the simplest of the above cases and may serve to illustrate the general difficulties of surface to surface intersection problems.

25.5.1. RPP/IA surface intersection

We start with a rational polynomial parametric surface to implicit algebraic surface intersection problem defined as:

$$\mathbf{r} = \mathbf{r}(u, v) = \left(\frac{X(u, v)}{W(u, v)}, \frac{Y(u, v)}{W(u, v)}, \frac{Z(u, v)}{W(u, v)} \right)^T \cap f(\mathbf{r}) = 0, \quad 0 \leq u, v \leq 1. \quad (25.22)$$

This leads to four algebraic equations in five unknowns $\mathbf{r} = (x, y, z), u, v$. For the usual low degree surfaces $f(\mathbf{r})$ and low degree patches $\mathbf{r}(u, v)$, we can substitute $\mathbf{r}(u, v)$ into $f(\mathbf{r}) = 0$ to obtain an implicit algebraic curve in u, v [41,42,69,74]. Examples of low order implicit algebraic surfaces in practical use are planes (degree 1), the natural quadrics (cylinder, sphere, cone) (degree 2), and torii (degree 4). In fact in a survey of mechanical parts (mechanical elements), over 90% of all surfaces involved are of these types [27,81,93]. It is also well known that these low order implicit algebraic surfaces have a low degree rational polynomial parametric representation, which can be obtained effectively using exact arithmetic methods (whenever possible) so that when two such low order implicit algebraic surfaces are intersected, the methods of this section may be also used.

Formulation

Now let us denote the implicit algebraic surface $f(x, y, z) = 0$ of total degree m by

$$f(x, y, z) = \sum_{i=0}^m \sum_{j=0}^{m-i} \sum_{k=0}^{m-i-j} c_{ijk} x^i y^j z^k. \quad (25.23)$$

By substituting $x = \frac{X}{W}$, $y = \frac{Y}{W}$, $z = \frac{Z}{W}$, where X, Y, Z and W are all of degree p in u and q in v , into (25.23) and multiplying by W^m leads to an algebraic curve

$$F(u, v) = \sum_{i=0}^m \sum_{j=0}^{m-i} \sum_{k=0}^{m-i-j} c_{ijk} X^i(u, v) Y^j(u, v) Z^k(u, v) W^{m-i-j-k}(u, v) = 0, \quad (25.24)$$

of degree $M = mp$ and $N = mq$ in u, v , respectively. Consequently, the problem of intersection reduces to the problem of tracing $F(u, v) = 0$ without omitting any special features of the curve, e.g. small loops, singularities, and accurately computing all its branches. This is a fundamental problem in *algebraic geometry* [96] and much work has been done to understand its solution [79]. In the context of algebraic geometry the coefficients of $F(u, v) = 0$ are integers. In the context of CAD, the coefficients of $F = 0$, and $\mathbf{r} = \mathbf{r}(u, v)$ are FP numbers. Therefore, if the above substitution is performed in FP arithmetic the coefficients of $F(u, v) = 0$ involve error, which may considerably modify the problem being solved. To avoid such error, rational arithmetic should be used for robustness (whenever possible) as discussed in Section 25.3.

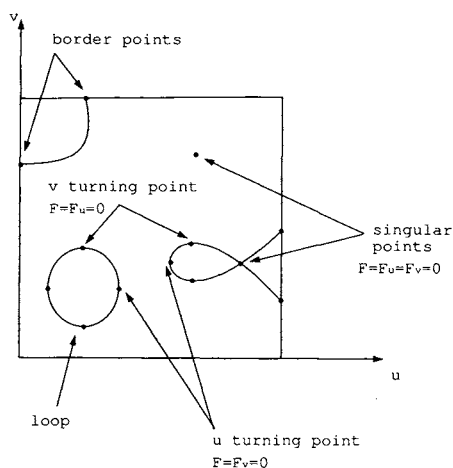


Figure 25.1. Parameter space of $\mathbf{r}(u, v)$ and resulting algebraic curve $F(u, v) = 0$.

The algebraic curve

$$F(u, v) = \sum_{i=0}^M \sum_{j=0}^N c_{ij}^M u^i v^j = 0, \tag{25.25}$$

can be reformulated in terms of Bernstein polynomials

$$F(u, v) = \sum_{i=0}^M \sum_{j=0}^N c_{ij}^B B_{i,M}(u) B_{j,N}(v) = 0, \tag{25.26}$$

where $(u, v) \in [0, 1]^2$. In fact the power basis form of $F(u, v) = 0$ need not be computed at all, if polynomial arithmetic for Bernstein polynomials [21] is used.

The advantage of the Bernstein form is its numerical stability and convex hull property. If $c_{ij}^B > 0$ or $c_{ij}^B < 0$ for all i, j , there is no solution and the two surfaces do not intersect.

More precisely, the entire algebraic surface $f(\mathbf{r}) = 0$ does not intersect the surface patch $\mathbf{r} = \mathbf{r}(u, v)$ for $(u, v) \in [0, 1]^2$. Obviously, when all $c_{ij}^B = 0$ the two surfaces coincide in their entirety. A somewhat complex algebraic curve $F(u, v) = 0$ is shown in Figure 25.1 involving various branches (from border to border), internal loops, and singularities.

Tracing method

Given a point on every branch of an algebraic curve, the curve can be traced using differential curve properties. We now consider the intersection curve which lies on the surface represented by the parametric form $\mathbf{r}(t) = \mathbf{r}(u(t), v(t))$. Differentiating (25.26) with respect to t yields

$$F_u \dot{u} + F_v \dot{v} = 0, \quad (25.27)$$

where u, v are considered as functions of a parameter t . The solution to the differential equation is given by

$$\dot{u} = \xi F_v(u, v), \quad \dot{v} = -\xi F_u(u, v), \quad (25.28)$$

where ξ is an arbitrary nonzero factor. For example, ξ can be chosen to provide arc length parametrization using the first fundamental form of the surface as a normalization condition

$$\xi = \pm \frac{1}{\sqrt{EF_v^2 - 2FF_uF_v + GF_u^2}}, \quad (25.29)$$

where E, F and G are first fundamental form coefficients of the parametric surface following standard differential geometry terminology. Note that F in Equation (25.29) has no relation with function F that appears in Sections 25.4 and 25.5. Equations (25.28) form a system of two first order nonlinear differential equations which can be solved by the Runge-Kutta or other methods with adaptive step size [15]. For a tracing method to work properly, we must provide all the starting points of all branches in advance. Step size selection is complex and too large a step size may lead to straying or looping. Tracing through singularities ($F_u^2 + F_v^2 = 0$) is also problematic.

Characteristic points

Starting points for tracing algebraic curves are identified by looking for characteristic points defined below:

1. *Border points:* The intersections of $F(u, v) = 0$ with all four boundary edges of the parameter space $[0, 1]^2$, e.g. $F(0, v) = 0, 0 \leq v \leq 1$.
2. *Turning points:* The u -turning points are the points where the tangent of $F(u, v) = 0$ is parallel to the $u = 0$ axis, which satisfies the simultaneous equations $F = F_v = 0$ (with $F_u \neq 0$). On the other hand the v -turning points are the points where the tangent of $F(u, v) = 0$ is parallel to the $v = 0$ axis, which satisfies the simultaneous equations $F = F_u = 0$ (with $F_v \neq 0$). Both types of turning points are shown in Fig. 25.1. If F has a degree of (M, N) in (u, v) , then the degrees of F_u and F_v will be $(M - 1, N)$ and $(M, N - 1)$, respectively. It can be shown that the total number of roots of two simultaneous polynomial equations in two variables whose degree are

(m, n) and (p, q) , respectively, is $mq + np$ [19]. Therefore the number of u -turning points and v -turning points can be at most $2MN - M$ and $2MN - N$, respectively.

3. *Singular points:* The points on the curve which satisfy the following three simultaneous equations $F = F_u = F_v = 0$ are called singular points. Noting that $f(x, y, z) = 0$, and $F(u, v) = W^m(u, v)f(x, y, z) = 0$, we deduce

$$F_u = mW^{m-1}W_u f + W^m \left(\frac{\partial f}{\partial x} \frac{\partial x}{\partial u} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial u} + \frac{\partial f}{\partial z} \frac{\partial z}{\partial u} \right) = W^m \nabla f \cdot \mathbf{r}_u, \quad (25.30)$$

and hence at singular points $\nabla f \cdot \mathbf{r}_u = \nabla f \cdot \mathbf{r}_v = 0$. This means that $\nabla f \parallel \mathbf{r}_u \times \mathbf{r}_v$ or that the normals of two surfaces are parallel and since $F(u, v) = 0$ at these points the two surfaces intersect tangentially. If F has a degree of (M, N) in (u, v) , the degrees of F_u and F_v will be $(M - 1, N)$ and $(M, N - 1)$, respectively, thus the number of singular points can be at most $2MN - M - N + 1$ [19]. However, singular points require little additional computation, since they are merely common roots of the u -turning points and v -turning points.

Table 25.1
Number of turning and singular points in various cases.

S_1	S_2	algebraic curve $F(u, v)$ degree M, N	max number u -turning pts $2MN - M$	max number v -turning pts $2MN - N$	max number singular points $2MN - M - N + 1$
plane	biquadratic	2, 2	6	6	5
plane	bicubic	3, 3	15	15	13
quadric	biquadratic	4, 4	28	28	25
quadric	bicubic	6, 6	66	66	61
torus	biquadratic	8, 8	120	120	113
torus	bicubic	12, 12	276	276	265
biquadratic	biquadratic	16, 16	496	496	481
bicubic	biquadratic	36, 36	2556	2556	2521
bicubic	bicubic	54, 54	5778	5778	5725

From the above discussions we can get upper bounds for the maximum number of u -, v -turning points and singular points [68] as listed in Table 25.1. These bounds refer to the maximum possible number of solutions (u, v) in the entire complex plane. Biquadratic and bicubic surfaces in the first column of Table 25.1 are degree 8 and 18 implicit algebraic surfaces. It turns out that the number of such points in the real square $[0, 1]^2$ is much smaller, but can still be quite large. Consequently methods which focus only on the real solutions in $[0, 1]^2$ are advantageous, such as the IPP algorithm described in Section 25.3.

Analysis of singular points

Let us construct a parametric equation of a straight line L through a point (u_0, v_0) on the algebraic curve $F(u, v) = 0$

$$u = u_0 + \alpha t, \quad v = v_0 + \beta t, \quad (25.31)$$

where α and β are constants and t is the parameter [80,96]. We find its intersections between L and the algebraic curve $F(u, v) = 0$ by determining the roots of $F(u_0 + \alpha t, v_0 + \beta t) = 0$. Taylor expansion of the left hand side gives

$$(\alpha F_u + \beta F_v)t + \frac{1}{2}(\alpha^2 F_{uu} + 2\alpha\beta F_{uv} + \beta^2 F_{vv})t^2 + \dots = 0, \quad (25.32)$$

where partial derivatives of F are evaluated at (u_0, v_0) and $F(u_0, v_0) = 0$ is used.

When F_u and F_v are not both zero ($F_u^2 + F_v^2 > 0$) at (u_0, v_0) , (25.32) has a single root $t = 0$ and every line through (u_0, v_0) has a single intersection with the algebraic curve at (u_0, v_0) except for one case where $\alpha F_u + \beta F_v = 0$ for certain values of α and β . In such cases (25.32) has a double root $t = 0$, provided at least one of the second order partial derivatives is not zero ($F_{uu}^2 + F_{uv}^2 + F_{vv}^2 > 0$), and L is tangent to the curve at (u_0, v_0) .

When (u_0, v_0) is a singular point ($F_u(u_0, v_0) = F_v(u_0, v_0) = F(u_0, v_0) = 0$), and at least one of F_{uu} , F_{uv} , F_{vv} is not zero ($F_{uu}^2 + F_{uv}^2 + F_{vv}^2 > 0$), then $t = 0$ is a double root and has at least two intersections at (u_0, v_0) except for the values of α and β which satisfy

$$\alpha^2 F_{uu} + 2\alpha\beta F_{uv} + \beta^2 F_{vv} = 0. \quad (25.33)$$

In such cases, $t = 0$ is a triple root, provided at least one of the third order partial derivatives is not zero ($F_{uuu}^2 + F_{uuv}^2 + F_{uvv}^2 + F_{vvv}^2 > 0$). We can solve the quadratic equation (25.33) for $\frac{\alpha}{\beta}$ or $\frac{\beta}{\alpha}$ which leads to the following three possibilities: (1) Two real distinct roots: These values correspond to two distinct tangent directions at the singular point, which implies the algebraic curve has a self-intersection. (2) One real double root: This value corresponds to one tangent direction at the singular point, which implies a cusp. (3) Two complex roots: No real tangents at the singular point implies an isolated point.

Computing starting points for all branches

Starting points for tracing algebraic curves could be border points, turning points and singular points. Border points involve solution of a univariate polynomial equation, e.g. for border along $u = 0$, using (25.26)

$$F(0, v) = \sum_{j=0}^N c_{0j}^B B_{j,N}(v) = 0. \quad (25.34)$$

Turning and singular point computation involve the first order partial derivatives:

$$F_u(u, v) = M \sum_{i=0}^{M-1} \sum_{j=0}^N (c_{i+1,j}^B - c_{ij}^B) B_{i,M-1}(u) B_{j,N}(v), \quad (25.35)$$

$$F_v(u, v) = N \sum_{i=0}^M \sum_{j=0}^{N-1} (c_{i,j+1}^B - c_{ij}^B) B_{i,M}(u) B_{j,N-1}(v). \quad (25.36)$$

Consequently, computing turning points ($F = F_u = 0$ and $F = F_v = 0$) is equivalent to solving a system of two nonlinear polynomial equations in two variables, and computing singularities $F = F_u = F_v = 0$ is equivalent to solving an overconstrained system of three nonlinear polynomial equations in two variables. Solution of these nonlinear polynomial systems is addressed in Section 25.3.

25.5.2. RPP/RPP surface intersection

Rational polynomial parametric surface to rational polynomial parametric surface intersection is defined as:

$$\begin{aligned} \mathbf{r} &= \mathbf{r}_1(\sigma, t) = \left(\frac{X_1(\sigma, t)}{W_1(\sigma, t)}, \frac{Y_1(\sigma, t)}{W_1(\sigma, t)}, \frac{Z_1(\sigma, t)}{W_1(\sigma, t)} \right)^T, \quad 0 \leq \sigma, t \leq 1, \\ \cap \mathbf{r} &= \mathbf{r}_2(u, v) = \left(\frac{X_2(u, v)}{W_2(u, v)}, \frac{Y_2(u, v)}{W_2(u, v)}, \frac{Z_2(u, v)}{W_2(u, v)} \right)^T, \quad 0 \leq u, v \leq 1. \end{aligned} \quad (25.37)$$

Formulation can be provided by setting $\mathbf{r}_1(\sigma, t) = \mathbf{r}_2(u, v)$ which leads to three nonlinear polynomial equations for four unknowns σ, t, u, v . It is an underconstrained system with 3 equations and 4 unknowns. This system can be solved by the IPP algorithm of Section 25.3. However, as the solutions are typically not isolated points but curves, such approach is inefficient when small tolerances are used. Another method involves implicitization of $\mathbf{r}_1(\sigma, t)$ to the form $f(\mathbf{r}) = 0$ and substitution of $\mathbf{r} = \mathbf{r}_2(u, v)$ into f to reduce the problem to RPP/IA case for a low degree surface [42]. Heo et al. [30] developed an intersection algorithm for two ruled surfaces which performs more efficiently than those for general parametric surfaces.

There are three major techniques for solving RPR/RPP surface intersections. A review as of 1993 can be found in [67] and a more up-to-date and detailed treatment in [68].

Lattice methods

Lattice method reduces the dimensionality of surface intersections by computing intersections of a number of isoparametric curves of one surface with the other surface followed by connection of the resulting discrete intersection points to form different solution branches [75]. For intersections of parametric patches, the method reduces to the solution of a large number of independent systems of nonlinear equations. The reduction of problem dimensionality in lattice methods involves an initial choice of grid resolution, which, in turn, may lead the method to miss important features of the solution, such as small loops and isolated points which reflect near tangency or tangency of intersecting surfaces, and to provide incorrect connectivity. Appropriate methods for the solution of the resulting nonlinear equations in the present context are identified in Section 25.3.

Subdivision methods

Subdivision methods in their most basic form, involve recursive decomposition of the problem into simpler similar problems until a level of simplicity is reached, which allows simple direct solution, (e.g. plane/plane intersection). This is followed by a connection phase of the individual solutions to form the complete solution. Initially conceived in the context of intersections of polynomial parametric surfaces [45], they can be extended to the computation of RPP/IA and IA/IA surface intersections [69]. A simple subdivision al-

gorithm employs uniform subdivision which leads to a uniform quadtree data structure [33]. Subdivision techniques do not require starting points as marching methods, an important advantage. General non-uniform subdivision [13] allows selective refinement of the solution providing the basis for an adaptive intersection technique. A disadvantage of subdivision techniques used in the evaluation of the entire intersection set is that, in actual implementations with finite subdivision steps, correct connectivity of solution branches in the vicinity of singular or near-singular points is difficult to guarantee, small loops may be missed (in methods with polyhedral surface approximations) or extraneous loops may be present in the approximation of the solution. Furthermore, if subdivision methods are used for high precision evaluation of the entire intersection set, they lead to data proliferation and are consequently slow, and, therefore, unattractive. There are many applications in CAD/CAM, that require high accuracy, for which pure subdivision methods are impractical. However, adaptive subdivision methods coupled with efficient local techniques to get high accuracy, offer an attractive approach for the computation of characteristic points. These points can then be used in initiating efficient marching methods for tracing intersection curves.

As can be seen from the above review, common problems of intersection methods include the difficulty in handling singularities, surface overlap and efficiently identifying closely spaced features and small loops. These algorithmic difficulties are further compounded by numerical error present in finite precision computations.

Marching methods

Marching methods involve generation of sequences of points of an intersection curve branch by stepping from a given point on the required curve in a direction prescribed by the local differential geometry [3,5,40,101], similar to tracing a planar algebraic curve $F(u, v) = 0$ in Section 25.5.1. However, such methods are by themselves *incomplete* in that they require *starting points* for every branch of the solution. In order to identify all connected components of the intersection curve, a set of characteristic points on the intersection curve can be defined. As seen in Section 25.5.1, such a set may include *border, turning and singular points* of the intersection and provides at least one point on any connected intersection segment and identifies all singularities. For RPP/RPP surface intersections a more convenient set of such points sufficient to discover all connected components of the intersection, includes *border and collinear normal points* between the two surfaces. Collinear normal points provide points inside all intersection loops and all singular points [34].

Border points are points of the intersection at which at least one of the parametric variables σ, t, u, v takes a value equal to the border of the σ - t or u - v parametric domain. To compute border points, a piecewise rational polynomial curve to piecewise rational polynomial surface intersection capability is required, e.g., $\mathbf{r}_1(0, t) = \mathbf{r}_2(u, v)$, which we discussed in Section 25.4.2.

Sederberg et al. [84] first recognized the importance of collinear normal points in detecting the existence of closed intersection loops in intersection problems of two distinct parametric surface patches. These are points on the two parametric surfaces at which the normal vectors are collinear. Collinear normal points are a subset of parallel normal points first used by [89] in surface intersection loop detection methods.

To simplify the notation, we replace $\mathbf{r}_1(\sigma, t)$ by $\mathbf{p}(\sigma, t)$ and $\mathbf{r}_2(u, v)$ by $\mathbf{q}(u, v)$. Then the collinear normal points satisfy the following equations [34]:

$$(\mathbf{p}_\sigma \times \mathbf{p}_t) \cdot \mathbf{q}_u = 0, \quad (\mathbf{p}_\sigma \times \mathbf{p}_t) \cdot \mathbf{q}_v = 0, \quad (\mathbf{p} - \mathbf{q}) \cdot \mathbf{p}_\sigma = 0, \quad (\mathbf{p} - \mathbf{q}) \cdot \mathbf{p}_t = 0. \quad (25.38)$$

Equations (25.38) form a system of four nonlinear polynomial equations that can be solved using the methods of Section 25.3 (see also [34] for more details on interval methods coupled with subdivision to solve the system (25.38)). Now we split the patches in (at least) one parametric direction at these collinear normal points. Consequently, starting points are only border points on the boundaries of all subdomains created. Grandine and Klein [26] follow a systematic approach for topology resolution of B-spline surface intersections. In this process, they determine the structure of the intersection curves including closed loops prior to numerical tracing (following a marching method based on numerical integration of a differential algebraic system of equations). Topology resolution in this context relies on an extension of the PP algorithm (see Section 25.3.2) to the B-spline case implemented in floating point (with normalization of the equations in the range $[-1, 1]$ and normalization of the knot vector in each subdomain in the range $[0, 1]$ at each iteration step of the process to capitalize on the higher density of floating point numbers in this range, thereby improving numerical robustness of the algorithm). An alternate way to detect closed intersection loops is to use topological methods [12,40,49,51,52,60,95,94]. Also bounding pyramids [39,86] can be used effectively to assure the nonexistence of closed surface to surface intersection loops. These earlier methods need to be implemented in exact or RIA for robustness.

In order to trace the intersection curve, starting points must be located prior to tracing. An intersection curve branch can be traced if its pre-image starts from the parametric domain boundary in either parameter domain [4]. The marching direction coincides with the tangential direction of the intersection curve $\mathbf{c}(s)$ which is perpendicular to the normal vectors of both surfaces. Therefore, the marching direction can be obtained as follows:

$$\mathbf{c}'(s) = \frac{\mathbf{P}(\sigma, t) \times \mathbf{Q}(u, v)}{|\mathbf{P}(\sigma, t) \times \mathbf{Q}(u, v)|}, \quad (25.39)$$

where the normalization forces $\mathbf{c}(s)$ to be arc length parametrized and

$$\mathbf{P}(\sigma, t) = \mathbf{p}_\sigma \times \mathbf{p}_t, \quad \mathbf{Q}(u, v) = \mathbf{q}_u \times \mathbf{q}_v, \quad (25.40)$$

are the normal vectors of \mathbf{p} and \mathbf{q} , respectively. When the two surfaces intersect tangentially, we cannot use Equation (25.39) since the denominator vanishes. In such cases we must find the marching direction in an alternate way [103].

The intersection curve can also be viewed as a curve on the two intersecting surfaces. A curve $\sigma = \sigma(s)$, $t = t(s)$ in the σt -plane defines a curve $\mathbf{r} = \mathbf{c}(s) = \mathbf{p}(\sigma(s), t(s))$ on a parametric surface $\mathbf{p}(\sigma, t)$, as well as a curve $u = u(s)$ $v = v(s)$ in the uv -plane defines a curve $\mathbf{r} = \mathbf{c}(s) = \mathbf{q}(u(s), v(s))$ on a parametric surface $\mathbf{q}(u, v)$. We can derive the first derivative of the intersection curve as a curve on the parametric surface using the chain rule:

$$\mathbf{c}'(s) = \mathbf{p}_\sigma \sigma' + \mathbf{p}_t t', \quad \mathbf{c}'(s) = \mathbf{q}_u u' + \mathbf{q}_v v'. \quad (25.41)$$

Since we know the unit tangent vector of the intersection curve from Equation (25.39), we can find σ' and t' as well as u' and v' by taking the dot product on both sides of the first equation of (25.41) with \mathbf{p}_σ and \mathbf{p}_t and the second equation with \mathbf{q}_u and \mathbf{q}_v , which leads to two linear systems [34]. The solutions are immediately obtained as

$$\sigma' = \frac{\det(\mathbf{c}', \mathbf{p}_t, \mathbf{P}(\sigma, t))}{\mathbf{P}(\sigma, t) \cdot \mathbf{P}(\sigma, t)}, \quad t' = \frac{\det(\mathbf{p}_\sigma, \mathbf{c}', \mathbf{P}(\sigma, t))}{\mathbf{P}(\sigma, t) \cdot \mathbf{P}(\sigma, t)}, \quad (25.42)$$

$$u' = \frac{\det(\mathbf{c}', \mathbf{q}_v, \mathbf{Q}(u, v))}{\mathbf{Q}(u, v) \cdot \mathbf{Q}(u, v)}, \quad v' = \frac{\det(\mathbf{q}_u, \mathbf{c}', \mathbf{Q}(u, v))}{\mathbf{Q}(u, v) \cdot \mathbf{Q}(u, v)}, \quad (25.43)$$

where \det denotes the determinant (see also [26]).

The points of the intersection curves are computed successively by integrating the initial value problem for a system of nonlinear ordinary differential equations (25.42) and (25.43) using standard numerical techniques [15].

25.5.3. IA/IA surface intersection

Implicit algebraic surface to implicit algebraic surface intersection is defined as follows:

$$f(\mathbf{r}) = 0 \cap g(\mathbf{r}) = 0, \quad (25.44)$$

where f, g are polynomial functions. Here we have two equations in three unknowns \mathbf{r} . Bajaj et al. [3] developed a marching method for IA/IA surface intersection (as well as for parametric surfaces).

A method for low order f, g is to eliminate one variable (e.g. z) to find projection of intersection curves on the plane of other two variables (e.g. x, y), then trace the algebraic curve and use the inversion algorithm to find z . Intersections of low degree implicit algebraic surfaces are of special interest in the boundary evaluation of the Constructive Solid Geometry models. A more complete analysis of the special intersections of two quadric surfaces can be found in [48,61,82,87,98].

25.6. CONCLUSION

Some important outstanding issues in the area of intersection problems are summarized below. While solving nonlinear polynomial systems, as a preliminary step in computing characteristic points of surface intersections, it is frequently necessary to deal with solution sets that are not zero-dimensional (e.g. the solution sets are one-dimensional, two-dimensional etc.). Most of the methods experience serious numerical and efficiency difficulties in those cases. Methods to deal effectively with these problems need to be developed, including methods to identify and, if possible, parameterize these higher-dimensional solution sets.

Extension of current intersection methods applied on rational B-spline surfaces, to more general and complex surfaces requires further study. Such surfaces include offset, generalized cylinder (pipe or canal surfaces in particular), blending, and medial surfaces and surfaces arising from the solution of partial differential equations or via recursion techniques (subdivision surfaces, see Chapter 12 of this handbook [76]). Intersections of such surfaces with the basic low order algebraic and rational B-spline surfaces, commonly

used in CAD need to be explored. However, a basic element of a solution of many of these problems is the auxiliary variable method described in [32,54,68], where the problem is reduced to a higher dimension nonlinear polynomial system. In some cases, recent research has indicated that some special instances of these general surfaces can be exactly expressed as rational polynomial surfaces [50,53,56,73] of higher degree and therefore these problems are reducible at least in principle to the problems addressed in this paper. Further research is needed to implement this idea in a practical setting and examine the relative efficiency of competing approaches.

Investigating the effects of floating point arithmetic on the implementation of intersection algorithms has been an important area for basic research during the last decade. Ways to enhance the precision of intersection computation, to monitor numerical error contamination and alternate means of performing arithmetic, not relying on imprecise floating point computation alone, have been explored in some detail. Researchers in surface intersection problems during the last decade have already obtained a good understanding of robustness problems when employing floating point arithmetic and of methods to mitigate these problems based on normalization of the system [26] and rounded interval arithmetic [34]. However, these methods are not a panacea since they cannot resolve effectively non-zero-dimensional solution sets of nonlinear systems or achieve very high precision in reasonable computation times. A related active problem area has been the rectification of solid models expressed in the boundary representation form, which attempts to resolve intersection inconsistencies in such models and create topologically and geometrically consistent models [70].

As a result of these deficiencies, recent research tends to focus on exact methods involving rational arithmetic [38,77,79]. Much research remains to be done in bringing such methods to the CAD practice, generalizing the arithmetic to go beyond rational and algebraic numbers (eg. involving transcendental numbers of trigonometric form), and to explore more efficient alternatives that are generally applicable in low and high degree problems alike. Finally, a general and comprehensive comparison and mapping of the efficiency properties of all available methods for solving nonlinear systems robustly would be valuable as a guide for future research.

Acknowledgements

We thank T. Sakkalis, M. N. Vrahatis and T. A. Grandine for their expert input and comments on an earlier version of this manuscript.

REFERENCES

1. S.L. Abrams, W. Cho, C.-Y. Hu, T. Maekawa, N.M. Patrikalakis, E.C. Sherbrooke, and X. Ye. Efficient and reliable methods for rounded-interval arithmetic. *Computer-Aided Design*, 30(8):657–665, July 1998.
2. W. Auzinger and H.J. Stetter. An elimination algorithm for the computation of zeros of a system of multivariate polynomial equations. In R.P. Agarwal, Y.M. Chow, and S.J. Wilson, editors, *Numerical Mathematics, Singapore, 1988, International Series of Numerical Mathematics, Volume 86*, pages 11–30. Birkhäuser Verlag, Boston Basel Berlin, 1988.

3. C.L. Bajaj, C.M. Hoffmann, J.E. Hopcroft, and R.E. Lynch. Tracing surface intersections. *Computer Aided Geometric Design*, 5(4):285–307, November 1988.
4. R.E. Barnhill, G. Farin, M. Jordan, and B.R. Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4(1-2):3–16, July 1987.
5. R.E. Barnhill and S.N. Kersey. A marching method for parametric surface / surface intersection. *Computer Aided Geometric Design*, 7(1-4):257–280, June 1990.
6. C. Blik. *Computer Methods for Design Automation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, July 1992.
7. B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, Innsbruck, Austria, 1965.
8. B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N.K. Bose, editor, *Multidimensional Systems Theory: Progress, Directions and Open Problems in Multidimensional Systems*, pages 184–232. Dordrecht, Holland: D. Reidel Publishing Company, 1985.
9. J.F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.
10. J.F. Canny and I.Z. Emiris. An efficient algorithm for the sparse mixed resultant. In G. Cohen, T. Mora, and O. Moreno, editors, *Proceedings of 10th International Symposium, Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 89–104. Springer-Verlag, 1993.
11. B.W.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, and S.M. Watt. *First Leaves: A Tutorial Introduction to Maple V*. Springer-Verlag, 1992.
12. K.-P. Cheng. Using plane vector fields to obtain all the intersection curves of two general surfaces. In W. Strasser and H. Seidel, editors, *Theory and Practice of Geometric Modeling*, pages 187–204. Springer-Verlag, New York, 1989.
13. E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing*, 14(2):87–111, October 1980.
14. G.E. Collins and R. Loos. Real zeros of polynomials. In B. Buchberger, G.E. Collins, and R. Loos, editors, *Computer Algebra: Symbolic and Algebraic Computation*, pages 83–94. Springer-Verlag, Vienna, 1982.
15. G. Dahlquist and Å. Björck. *Numerical Methods*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974.
16. J.E. Dennis and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, Philadelphia, 1996.
17. I.Z. Emiris. *Sparse Elimination and Applications in Kinematics*. PhD thesis, University of California at Berkeley, Berkeley, CA, 1994.
18. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, Boston, MA, 3rd edition, 1993.
19. R.T. Farouki. The characterization of parametric surface sections. *Computer Vision, Graphics and Image Processing*, 33(2):209–236, February 1986.
20. R.T. Farouki and V.T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, November 1987.

21. R.T. Farouki and V.T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5(1):1–26, June 1988.
22. J.C. Faugere, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero-dimensional Gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.
23. C.B. Garcia and W.I. Zangwill. Global continuation methods for finding all solutions to polynomial systems of equations in n variables. In A.V. Fiacco and K.O. Kortanek, editors, *Extremal Methods and Systems Analysis*, pages 481–497. Springer-Verlag, New York, NY, 1980.
24. T.A. Grandine. Computing zeroes of spline functions. *Computer Aided Geometric Design*, 6(2):129–136, May 1989.
25. T.A. Grandine. Geometry processing. In G. Farin, J. Hoschek, and M.-S. Kim, editors, *Handbook of Computer Aided Geometric Design*. Elsevier, 2002.
26. T.A. Grandine and F.W. Klein. A new approach to the surface intersection problem. *Computer Aided Geometric Design*, 14(2):111–134, 1997.
27. D.G. Hakala, R.C. Hillyard, B.E. Nourse, and P.J. Malraison. Natural quadrics in mechanical design. In *Proceedings of the Autofact West 1, Anaheim, CA*, pages 363–378, November 1980.
28. P. Van Hentenryck, D. McAllester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, April 1997.
29. P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica: A Modeling Language for Global Optimization*. MIT Press, Cambridge, MA, 1997.
30. H.-S. Heo, M.-S. Kim, and G. Elber. The intersection of two ruled surfaces. *Computer-Aided Design*, 31(1):33–50, January 1999.
31. C.M. Hoffmann. The problems of accuracy and robustness in geometric computation. *Computer*, 22(3):31–41, March 1989.
32. C.M. Hoffmann. A dimensionality paradigm for surface interrogations. *Computer Aided Geometric Design*, 7(6):517–532, November 1990.
33. J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. A.K. Peters, Wellesley, MA, 1993. Translated by L.L. Schumaker.
34. C.Y. Hu, T. Maekawa, N.M. Patrikalakis, and X. Ye. Robust interval algorithm for surface intersections. *Computer-Aided Design*, 29(9):617–627, September 1997.
35. C.Y. Hu, T. Maekawa, E.C. Sherbrooke, and N.M. Patrikalakis. Robust interval algorithm for curve intersections. *Computer-Aided Design*, 28(6/7):495–506, June/July 1996.
36. J.P. Jouanolou. Le formalisme du resultant. *Advances in Mathematics*, 90(2):117–263, 1991.
37. R.B. Kearfott. Interval Newton/generalized bisection when there are singularities near roots. *Annals of Operations Research*, 25:181–196, 1990.
38. J. Keyser, T. Culver, D. Manocha, and S. Krishnan. Efficient and exact manipulation of algebraic points and curves. *Computer-Aided Design*, 32(11):649–662, September 2000.
39. G.A. Kriezis and N.M. Patrikalakis. Rational polynomial surface intersections. In G.A. Gabriele, editor, *Proceedings of the 17th ASME Design Automation Conference*,

- Vol. II*, pages 43–53, Miami, September 1991. ASME, New York, 1991.
40. G.A. Kriezis, N.M. Patrikalakis, and F.-E. Wolter. Topological and differential-equation methods for surface intersections. *Computer-Aided Design*, 24(1):41–55, January 1992.
 41. G.A. Kriezis, P.V. Prakash, and N.M. Patrikalakis. Method for intersecting algebraic surfaces with rational polynomial patches. *Computer-Aided Design*, 22(10):645–654, December 1990.
 42. S. Krishnan and D. Manocha. Efficient surface intersection algorithm based on lower-dimensional formulation. *ACM Transactions on Graphics*, 16(1):74–106, January 1997.
 43. Y.N. Lakshman. *On the complexity of computing Gröbner bases for zero dimensional ideals*. PhD thesis, Rennselaer Polytechnic Institute, Troy, NY, 1992.
 44. H. Lamure and D. Michelucci. Solving geometric constraints by homotopy. In C. Hoffmann and J. Rossignac, editors, *Proceedings of the Third ACM Solid Modeling Symposium*, pages 263–269, Salt Lake City, Utah, May 1995. ACM, NY.
 45. J.M. Lane and R.F. Riesenfeld. A theoretical development for the computer display and generation of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):35–46, January 1980.
 46. J.M. Lane and R.F. Riesenfeld. Bounds on a polynomial. *BIT: Nordisk Tidsskrift for Informations-Behandling*, 21(1):112–117, 1981.
 47. D. Lazard. Solving zero-dimensional algebraic systems. *Journal of Symbolic Computation*, 13(2):117–131, 1992.
 48. J.Z. Levin. Mathematical models for determining the intersections of quadric surfaces. *Computer Vision, Graphics and Image Processing*, 11:73–87, 1979.
 49. N.G. Lloyd. *Degree Theory*. Cambridge University Press, Cambridge, 1978.
 50. W. Lü and H. Pottmann. Pipe surfaces with rational spine curve are rational. *Computer Aided Geometric Design*, 13(7):621–628, October 1996.
 51. Y. Ma and Y.-S. Lee. Detection of loops and singularities of surface intersections. *Computer-Aided Design*, 30(14):1059–1067, December 1998.
 52. Y. Ma and R.C. Luo. Topological method for loop detection of surface intersection problems. *Computer-Aided Design*, 27(11):811–820, November 1995.
 53. T. Maekawa. An overview of offset curves and surfaces. *Computer-Aided Design*, 31(3):165–173, March 1999.
 54. T. Maekawa, W. Cho, and N.M. Patrikalakis. Computation of self-intersections of offsets of Bézier surface patches. *Journal of Mechanical Design, Transactions of the ASME*, 119(2):275–283, June 1997.
 55. T. Maekawa and N.M. Patrikalakis. Computation of singularities and intersections of offsets of planar curves. *Computer Aided Geometric Design*, 10(5):407–429, October 1993.
 56. T. Maekawa, N.M. Patrikalakis, T. Sakkalis, and G. Yu. Analysis and applications of pipe surfaces. *Computer Aided Geometric Design*, 15(5):437–458, May 1998.
 57. D. Manocha. Solving polynomial systems for curve, surface and solid modeling. In J. Rossignac, J. Turner, and G. Allen, editors, *Proceedings of 2nd ACM/IEEE Symposium on Solid Modeling and Applications*, pages 169–178, Montreal, May 1993. New York: ACM Press, 1993.

58. D. Manocha. Numerical methods for solving polynomial equations. In D.A. Cox and B. Sturmfels, editors, *Proceedings of Symposia in Applied Mathematics Volume 53, Applications of Computational Algebraic Geometry: American Mathematical Society short course*, pages 41–66, January 6–7, 1997, San Diego, California. American Mathematical Society, 1998.
59. D. Manocha and S. Krishnan. Solving algebraic systems using matrix computations. *Sigsam Bulletin: Communications in Computer Algebra*, 30(4):4–21, December 1996.
60. R.P. Markot and R.L. Magedson. Solutions of tangential surface and curve intersections. *Computer-Aided Design*, 21(7):421–429, September 1989.
61. J.R. Miller and R.N. Goldman. Geometric algorithms for detecting and calculating all conic sections in the intersection of any two natural quadratic surfaces. *Graphical Models and Image Processing*, 57(1):55–66, January 1995.
62. R.E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.
63. M. Niizeki and F. Yamaguchi. Projectively invariant intersection detections for solid modeling. *ACM Transactions on Graphics*, 13(3):277–299, July 1994.
64. T. Nishita, T.W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface patches. *ACM Computer Graphics*, 24(4):337–345, August 1990.
65. M. Noro, T. Takeshima, and K. Yokoyama. Solution of systems of algebraic equations and linear maps on residue class ring. *Journal of Symbolic Computation*, 14:399–417, 1992.
66. J.M. Ortega and W.C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970.
67. N.M. Patrikalakis. Surface-to-surface intersections. *IEEE Computer Graphics and Applications*, 13(1):89–95, January 1993.
68. N.M. Patrikalakis and T. Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer-Verlag, Heidelberg, 2002.
69. N.M. Patrikalakis and P.V. Prakash. Surface intersections for geometric modeling. *Journal of Mechanical Design, Transactions of the ASME*, 112(1):100–107, March 1990.
70. N.M. Patrikalakis, T. Sakkalis, and G. Shen. Boundary representation models: Validity and rectification. In R. Cipolla and R. Martin, editors, *The Mathematics of Surfaces IX*, pages 389–409, University of Cambridge, UK., September 2000. London: Springer.
71. S. Petitjean. Algebraic geometry and computer vision: Polynomial systems, real and complex roots. *Journal of Mathematical Imaging and Vision*, 10(3):191–220, 1999.
72. L.A. Piegl and W. Tiller. Symbolic operators for NURBS. *Computer-Aided Design*, 29(5):361–368, May 1997.
73. H. Pottmann. Rational curves and surfaces with rational offsets. *Computer Aided Geometric Design*, 12(2):175–192, March 1995.
74. M.J. Pratt and A.D. Geisow. Surface/surface intersection problems. In J.A. Gregory, editor, *The Mathematics of Surfaces*, pages 117–142. Clarendon Press, 1986.
75. J.R. Rossignac and A.A.G. Requicha. Piecewise-circular curves for geometric modeling. *IBM Journal of Research and Development*, 31(3):296–313, 1987.
76. M. Sabin. Subdivision surfaces. In G. Farin, J. Hoschek, and M.-S. Kim, editors, *Handbook of Computer Aided Geometric Design*. Elsevier, 2002.

77. T. Sakkalis. On the zeros of a polynomial vector field. Research Report RC-13303, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1987.
78. T. Sakkalis. The Euclidean algorithm and the degree of the Gauss map. *SIAM Journal on Computing*, 19(3):538–543, June 1990.
79. T. Sakkalis. The topological configuration of a real algebraic curve. *Bulletin of the Australian Mathematical Society*, 43:37–50, 1991.
80. T. Sakkalis and R. Farouki. Singular points of algebraic curves. *Journal of Symbolic Computation*, 9:405–421, June 1990.
81. N.M. Samuel, A.A.G. Requicha, and S.A. Elkind. Methodology and results of an industrial part survey. Technical Report Tech. Memo. No. 21, Production Automation Project, University of Rochester, Rochester, NY, 1976.
82. R.F. Sarraga. Algebraic methods for intersections of quadric surfaces in GMSOLID. *Computer Vision, Graphics and Image Processing*, 22(2):222–238, May 1983.
83. T.W. Sederberg, D.C. Anderson, and R.N. Goldman. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics and Image Processing*, 28(1):72–84, October 1984.
84. T.W. Sederberg, H.N. Christiansen, and S. Katz. Improved test for closed loops in surface intersections. *Computer-Aided Design*, 21(8):505–508, October 1989.
85. T.W. Sederberg and J. Zheng. Algebraic methods for computer aided geometric design. In G. Farin, J. Hoschek, and M.-S. Kim, editors, *Handbook of Computer Aided Geometric Design*. Elsevier, 2002.
86. T.W. Sederberg and A.K. Zundel. Pyramids that bound surface patches. *Graphical Models and Image Processing*, 58(1):75–81, January 1996.
87. C.-K. Shene and J.K. Johnstone. On the lower degree intersections of two natural quadrics. *ACM Transactions on Graphics*, 13(4):400–424, October 1994.
88. E.C. Sherbrooke and N.M. Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *Computer Aided Geometric Design*, 10(5):379–405, October 1993.
89. P. Sinha, E. Klassen, and K.K. Wang. Exploiting topological and geometric properties for selective subdivision. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 39–45. New York: ACM, 1985.
90. P. Stiller. Sparse resultants. Technical Report ISC-96-01-MATH, Texas A & M University, Institute for Scientific Computation, 1996.
91. B. Sturmfels. Introduction to resultants. In D.A. Cox and B. Sturmfels, editors, *Proceedings of Symposia in Applied Mathematics Volume 53, Applications of Computational Algebraic Geometry: American Mathematical Society short course*, pages 25–39, January 6-7, 1997, San Diego, California. American Mathematical Society, 1998.
92. B. Sturmfels and A. Zelevinsky. Multigraded resultants of Sylvester type. *Journal of Algebra*, 163(1):115–127, January 1994.
93. H.B. Voelcker et al. An introduction to PADL: Characteristics, status, and rationale. Technical Report Tech. Memo. No. 22, Production Automation Project, University of Rochester, Rochester, NY, December 1974.
94. M.N. Vrahatis. CHABIS: A mathematical software package for locating and evaluating roots of systems of nonlinear equations. *ACM Transactions on Mathematical*

- Software*, 14(4):330–336, December 1988.
95. M.N. Vrahatis. Solving systems of nonlinear equations using the nonzero value of the topological degree. *ACM Transactions on Mathematical Software*, 14(4):312–329, December 1988.
 96. R.J. Walker. *Algebraic Curves*. Princeton University Press, Princeton, New Jersey, 1950.
 97. H.S. Wilf. A global bisection algorithm for computing the zeros of polynomials in the complex plane. *Journal of the Association for Computing Machinery*, 25(3):415–420, July 1978.
 98. I. Wilf and Y. Manor. Quadric-surface intersection curves: shape and structure. *Computer-Aided Design*, 25(10):633–643, October 1993.
 99. F. Winkler. *Polynomial Algorithms in Computer Algebra*. Springer-Verlag, New York, 1996.
 100. S. Wolfram. *The Mathematica Book*. Wolfram Media, Champaign, IL, 3rd edition, 1996.
 101. S.-T. Wu and L.N. Andrade. Marching along a regular surface/surface intersection with circular steps. *Computer Aided Geometric Design*, 16(4):249–268, May 1999.
 102. F. Yamaguchi. A shift of playground for geometric processing from Euclidean to homogeneous. *The Visual Computer*, 14(7):315–327, 1998.
 103. X. Ye and T. Maekawa. Differential geometry of intersection curves of two surfaces. *Computer Aided Geometric Design*, 16(8):767–788, September 1999.
 104. W.I. Zangwill and C.B. Garcia. *Pathways to Solutions, Fixed Points, and Equilibria*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

Chapter 26

Reverse Engineering

Tamas Varady and Ralph Martin

Reverse engineering, as described in this Chapter, is the process of converting dense 3D point data captured from the surface of an object into a boundary representation CAD model.

26.1. INTRODUCTION

There are many applications of reverse engineering. It is often necessary to reproduce a part for which no original drawings or machining data are available. We may wish to re-engineer an existing part, by modifying it to construct an improved product. In aesthetic design and industrial styling, real-scale wood or clay models are still essential tools for evaluating 3D objects; nevertheless a CAD model is eventually needed. Generating custom smooth surfaces for prosthesis fitting is another example.

Note that the aim of reverse engineering is not simply to copy 3D objects, nor simply to obtain a triangular mesh for visualization purposes. Instead, we want models of captured objects which can be analyzed and modified. This requires each boundary surface to be suitably represented within the CAD model.

Different approaches to reverse engineering are necessary according to the type of object being scanned. The approaches given here are not appropriate for complex natural or artistic objects—we are only concerned with engineering shapes. These may be classified as *conventional engineering objects* or as *free-form objects*. The former typically have many surfaces of simple geometric form (planes, natural quadrics, tori) which meet in sharp edges or smooth blends. The latter have few surfaces of high geometric complexity, mostly meeting with G^1 or higher continuity. This classification is clearly artificial, but clarifies different approaches in the exposition which follows.

The aim of this Chapter is to summarise a representative set of algorithms which explain how reverse engineering can be carried out. These draw strongly on our own personal experience. Various interrelated steps are needed, relying on ideas from many areas of geometric modelling, some of which are adequately covered in other Chapters of

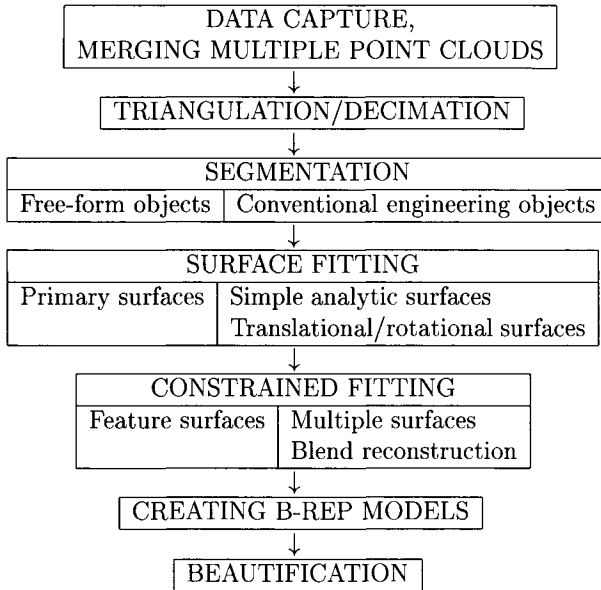


Figure 26.1. Steps in reverse engineering

this handbook. We cannot give full details of the algorithms, but try to present the key concepts, and show where further information can be obtained.

26.2. THE BASIC PHASES OF REVERSE ENGINEERING

This Section summarises the steps necessary to go from a scanned object to a final CAD model, as outlined in the flowchart in Figure 26.1. As noted, different approaches are needed for different types of object—the left track in the chart shows the steps for free-form objects, while the right track is for conventional objects. This chart is an idealisation of the process, which in practice may not be linear as shown, and may require iteration as well.

The first step is to *capture* dense 3D point data with a suitable device. Such data from several viewpoints then needs to be merged into a single data file in the same coordinate system.

The next step is to determine the nearest neighbours of each point, and link all points into a *triangular surface mesh*. This gives both the *local* topology of the point set, and the *global* topology of the skin of the object. These are needed in many further steps, particularly for efficiency reasons. For example, estimates of surface normal direction are computed from the points in a neighbourhood of a given point. Also, surfaces are fitted to contiguous regions of points in the triangulation.

Because the point cloud may contain a huge number of points, it is necessary to reduce the data to manageable proportions by removing redundant points in a *decimation*

process, either before triangulation, or after, or both. This process should not alter the global topology, or significantly change the local geometry.

One of the most crucial elements of reverse engineering is *segmentation* [38,74], i.e. marking out connected regions within the whole point cloud which form pre-images of single faces (or possibly multiple faces) of the B-rep model. Different methods are needed for free-form and conventional engineering objects. Automatic segmentation is possible in the latter case because of the limited number of surface types, because of translational and rotational symmetry, and because the surfaces are often delimited by sharp edges or small-radius blends. However, user assistance seems to be needed for free-form objects, because the faces have higher geometric complexity, and because it is not trivial to decide where one face ends and another starts.

Note that while image processing applications often use edge detection for segmentation, such an approach is *not* suitable for reverse engineering, because (i) data captured in the vicinity of sharp edges is particularly unreliable for at least some sensing devices, and (ii) boundaries between regions may be smooth edges. During segmentation, points in the vicinity of edges are temporarily discarded, leaving well-defined regions of points providing a firm basis for surface fitting. The discarded points *are* used later for finding blends and other connecting features between primary surfaces.

For each region of points, a single surface is *fitted* within a given tolerance. Thus, each segmented portion of the object's boundary is now represented by a continuous mathematical expression. Quite different fitting methods are needed for free-form faces and simple analytical faces. Further methods can also fit extrusions and surfaces of revolution.

Once the *primary surfaces* have been determined, subsequent surface fitting phases are also required. For free-form objects, various connecting *feature surfaces* are generated which are dependent on the primary surfaces, usually meeting them with tangential continuity. For conventional objects, we may wish to recognise and impose constraints between surfaces: for example, perpendicularity, coaxiality, and so on. For both classes of object, some form of *constrained* surface fitting is used: in the former case, for a single surface, in the latter case, for multiple surfaces. *Blend reconstruction* can also be considered to be a special case of constrained fitting.

Having obtained the individual surfaces, the geometry of the edges and vertices bounding each face must also be found, together with details of adjacency relationships. This information is necessary to construct a *boundary representation* data structure.

Due to gaps and noise in the point cloud data, and numerical errors in algorithms, the resulting B-rep model is unlikely to show all the regularities expected by an engineer. A *beautification* step may be used to improve the model, by first detecting, and then imposing, symmetries and other constraints on the model. This may be done both during the constrained fitting step mentioned earlier, and by a post-process which adjusts the parameters of the geometric elements found.

26.3. DATA CAPTURE

In this section we give a brief summary of how point data may be captured from an object which we wish to reverse engineer, and how point data from multiple views may

be merged to form a single point cloud.

26.3.1. Laser scanners

For data capture, typically laser scanners are used, which produce measured 3D points with high density (10^4 – 10^8 data points) and reasonable accuracy (10–100 μm). Some—of higher accuracy—are based on gantry or NC-machine-like arrangements, while others use similar technology to robot arms, with joint sensors. Other devices such as coordinate-measuring machines, which produce few measured points, are not really suitable for automatic reverse engineering. Typically, objects to be scanned must be prepared by coating them with a matt paint, but even so, the data may contain outliers, particularly near silhouette curves and concave edges. In single views, missing data may occur due to occlusion; this may hold for merged point clouds too if insufficient views are used. The data is usually a series of points along successive scanlines, but occlusion may cause steps and gaps along a given scanline. Thus, unlike 2D image data, 3D point data even in a single view may be unevenly distributed and is typically not organised in any regular structure.

26.3.2. Multiple view registration

A single view represents only a part of an object. To obtain a complete model, points from different views need to be *merged*. This is less important for arm-based scanners, which have the flexibility to capture data from a wide envelope of viewpoints in a single coordinate system, although even they cannot see the whole object in one setup. Determining appropriate transformations to put the different data sets into a common coordinate system is called *registration* [7,20].

Straightforward approaches to registration are to use (i) calibrating balls attached to the object, whose positions can readily be found in each data set—but they hide part of the object, and (ii) high-precision rotary tables, directly giving the registration—but in this case the bottom face and other downward pointing faces are not visible.

Automatic registration is often based on *iterative closest point* methods, which match closest points in several overlapping datasets. Such methods are slow, and require reasonable initial estimates of the registration. Some progress has been made towards overcoming these difficulties [51] using a hierarchical method based on computing characteristic curves in the point data and then special points on those curves. Registration is first done using the special points, then refined using the curves, and finally made accurate using the whole point set. This approach is even effective for obtaining an initial registration. Iterative *reciprocal* closest point methods [62] are more robust when there is relatively little overlap between the data sets.

26.4. TRIANGULATION AND DECIMATION

Triangulation aims to connect the measured data points into a structure which gives their adjacency relationships, and the correct global topology for the final model. A triangulation algorithm for reverse engineering must take into account the following. Data points are subject to noise, the point density is often uneven (because of merged views), and outliers may occur. The data is typically an unorganized point cloud, possibly with open boundaries, holes or multiple components. The triangulation formed should be one or more 2-manifolds. The method should be computationally efficient and robust.

Decimation aims to replace the collection of triangles with a much smaller set which is still suitable for downstream algorithms. In reverse engineering applications, vertices of deleted triangles are typically not discarded, but assigned to a remaining triangle for later use when the full data *is* required. Decimation is a large topic in its own right [35], and will not be discussed further here.

26.4.1. Triangulation overview

Many algorithms are based on the Delaunay tessellation [2,12,77] of the sample point set, or an α -shape of the points [24]. The α -shape is a subset of the 3-, 2-, 1-, and 0-dimensional simplices for which those elements lying on a sphere of radius less than α contains no points in its interior. In favourable cases, a value for α can be globally chosen so that the only triangles retained are those forming the external surface of the object. This is rarely possible in practice, and building a triangulation from an α -shape is difficult [32]. Improvements using local α -values have been suggested [3,8,25].

An early triangulation method is due to Hoppe [39]. A piecewise linear function is created estimating the signed distance from the boundary of the object, and the zero-set of this function is extracted by a *marching cubes* algorithm. This works for surfaces of arbitrary topology or with an open boundary. However, it is computationally expensive.

26.4.2. Kós's method

Kós's algorithm [49] attempts to avoid the limitations of the above approaches. The basic principle is to merge local triangulations, leading to a consistent global triangular mesh. The algorithm has three phases. Preprocessing organises the data points and estimates certain local quantities. The main phase builds a consistent triangulation. Finally, a postprocessing step optimises the triangulation by smoothing it.

Preprocessing consist of the following steps: (i) all sample points are put into an octree-like structure for speed, (ii) any coincident points are detected and removed, (iii) a selective neighbourhood graph is computed around each point, (iv) triangles adjacent to any boundary are flagged, (v) estimates of surface normal are computed at each point by locally fitting a quadric surface.

The main phase of the algorithm uses the idea of a generalised, curved Voronoi diagram of the sample points on the surface. It constructs a local Delaunay triangulation around each sample point A ; these triangles are candidates to become elements of the final triangulation. This is done by building a list B_1, \dots, B_n of neighbours of A such that the points B_i are ordered counter-clockwise, and that for any point C in the angle sector B_iAB_{i+1} , the points A and C should not be connectable in the quadrilateral AB_iCB_{i+1} . Given a quadrilateral $ABCD$, we define A and C to be connectable if, using angles projected onto a local tangent plane, $\angle ABC + \angle CDA < \angle BCD + \angle DAB$; otherwise, B and D are connectable.

To generate points B_i , the algorithm works by inserting and deleting points dynamically into a list, initially empty. A queue stores candidate points, initially containing A 's neighbours in the neighbourhood graph. At each step we take the point C closest to A from the queue. If C lies in the (projected) angle sector B_iAB_{i+1} , we test whether A and C are connectable in quadrilateral AB_iCB_{i+1} . If not, we discard the point C , otherwise we insert C into the list between B_i and B_{i+1} . After inserting C , some points may need

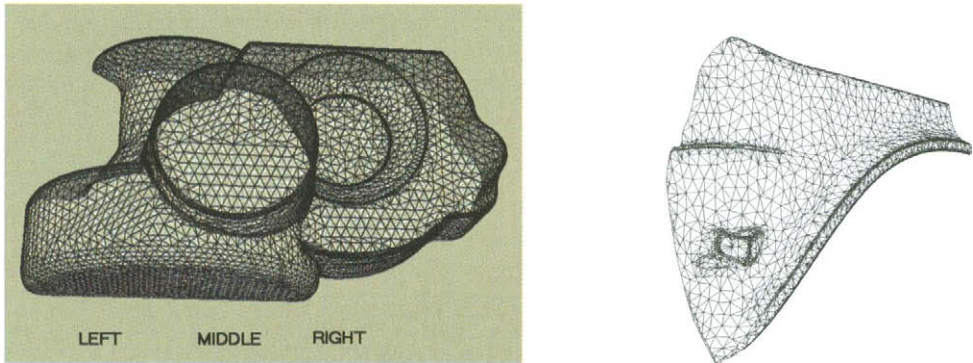


Figure 26.2. Triangulated point clouds from a conventional and a free-form object

to be deleted: B_j must be deleted if the points A and B_j are not connectable in the quadrilaterals $AB_{j-1}B_jC$ or ACB_jB_{j+1} . If C is inserted into the list, its neighbours are inserted into the queue if not already present. This iteration is repeated until the queue becomes empty.

After building the point list, we take all the triangles AB_iB_{i+1} which contain A , and the point A awards a *vote* for these triangles.

In most cases the set of candidate triangles does not form a manifold structure, and so a consistent triangle set must be selected. Each triangle is given a priority based on (i) the number of votes it has obtained, and (ii) an error computed from the difference between the estimated normals at the vertices and the normal of the triangle. Then, using the priority order, each triangle is considered for insertion into the final triangulation: it is added provided that the triangle mesh remains manifold and oriented, and the triangle does not overlap any previously added triangle.

After adding topologically consistent triangles, some polyhedral holes may remain to be filled. This is done by triangulating the hole into triangles of good shape using a heuristic method.

The postprocessing phase is now used to optimise the triangulation, based on an edge swapping method which keeps the original vertices. We iterate until there is no remaining edge swap which decreases the sum of triangle normal errors described above.

The results of triangulating point clouds captured from a conventional engineering object and a free-form object are shown in Figure 26.2. (In both cases the triangulations have been greatly decimated for clarity of illustration in this book.)

26.5. RECONSTRUCTING FREE-FORM OBJECTS

26.5.1. Segmentation strategies

We assume that the triangulated data represents a composite free-form object, which is decomposable into meaningful surface portions, each formed by one or more patches of an original CAD design, for example. The composite surface is internally smooth and is

bounded by edge loops. The external edges may be partly or entirely the boundaries of the patch structure; they may also be trimming curves cutting across the structure, such as might have arisen from intersection, Boolean operations or blending in a CAD system.

The most widely used parametric surfaces, such as Bézier and NURBS surfaces, map a rectangular parametric domain into three dimensions, resulting in a surface patch with four boundary curves. However, the majority of complex free-form shapes cannot be adequately represented by a single surface of this type.

We wish to reconstruct the object according to its original design. The key issue here is how to identify point regions within the triangulated mesh which can then be individually approximated by free-form surfaces. Four different approaches have been identified for reverse engineering free-form shapes [74], using:

- a global approximating surface (i.e. without segmentation)
- a segmentation based on the triangulation
- a segmentation based on a user defined curve network
- functional decomposition

In the first two approaches, surfaces are created without user interaction, and no attempt is made to recognize the underlying design structure. This is likely to consequentially result in a generated surface with poor overall quality, and which will not be suitable for demanding applications such as car body design. The first approach fits a single four-sided surface to the whole dataset, and at the same time, tries to accurately reproduce small details. These two requirements conflict, and although attempts have been made to reconcile them using special scalar weighting functions [19] or a hierarchical approach [31], only partial success has been achieved. The control point structure produced is a grid which cannot match arbitrarily oriented features in the source data.

The second approach can help to resolve this problem, since it can produce a more general control point structure, better adapted to the details of the object. It works by performing a drastic decimation of the triangulation, and uses this to produce quadrilaterals which are then covered by smooth patches [23,33]. However, in practice, such segmentation methods rarely divide the object into surface patches which are the ones expected or desired by an engineering user.

The third and fourth approaches rely on user assistance, and practical experience shows that much higher quality surfaces can be obtained in this way. In the *curve network* approach, the user draws an explicit segmentation over the triangulation, using his experience and understanding of the design to guide placement of polylines on the surface. Smooth curves are fitted to these polylines; these form boundaries of surfaces which are subsequently fitted. The curve network method has several deficiencies. First of all, it is very hard to determine suitable positions for smooth edges, for example the trimlines of a blend. Minor mistakes in edge positions will lead to the inclusion of points which logically belong to an adjacent surface patch, and consequent problems during surface fitting. Another problem is that the user may have to create artificial boundaries between regions to satisfy topological requirements for patching, even though the regions may naturally

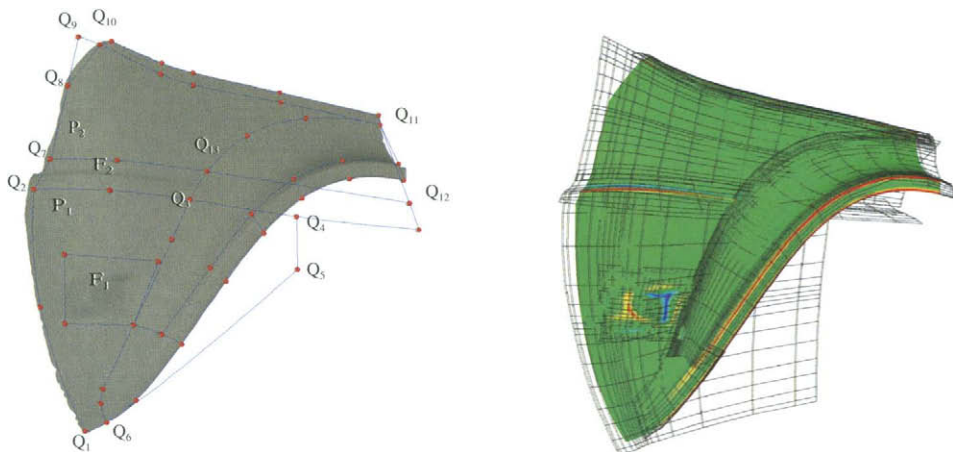


Figure 26.3. Functional decomposition and reconstructed surfaces

belong together. In such a case, global fairing of what logically should be a single region cannot readily be performed.

While curve network segmentation is probably the most widely applied approach in current industrial practice, *functional decomposition* methods are attracting attention because they overcome most of the difficulties mentioned above. They assume that the user has in mind an abstract model of the object, with a hierarchy of *primary surfaces*, and *dependent features* which are smooth transitions between the primary surface. Each primary surface is normally fitted by a four-sided patch, which may be extended, trimmed, or intersected as necessary.

Instead of drawing an explicit curve network over the point cloud, a set of curves is chosen whose sole purpose is to delimit regions of the triangulation, each of which definitely belongs to one primary surface. A usually rectangular primary surface, of greater extent than the region it contains, is fitted to the data points of the region. The position of the boundary of the rectangular surface is determined algorithmically, starting from a framework given by the user which decides the u and v parametric directions, and an initial parametrisation. A given primary surface may be based on a region with holes, or even several disjoint regions.

We illustrate the above concept in the left of Figure 26.3, which shows part of a car body panel. The primary surface P_2 has a rectangular frame defined by the loop Q_7 - Q_9 - Q_{11} - Q_{12} - Q_7 , and is fitted to data from the region defined by the loop Q_7 - Q_8 - Q_{10} - Q_{11} - Q_{13} - Q_7 . Primary surface P_1 is based on the region Q_1 - Q_2 - Q_3 - Q_6 - Q_1 . However, within P_1 is a feature element F_1 . Data points belonging F_1 must be excluded when fitting P_1 . Such an excluded region is called an *ignore area* for P_1 . This approach allows the required smooth surface over region P_1 to be reconstructed with high quality; the feature with its complex detail is added subsequently. Once the surface representations for P_1 and P_2

have been created, it is now possible to reconstruct a free-form step feature F_2 , which is clearly dependent on P_1 and P_2 . (We have ignored other primary surfaces and dependent features which are also present in this object.) The right hand side of Figure 26.3 shows a curvature map of the trimmed patches together with the control nets of the primary surfaces and dependent features. Although only a limited region was used to fit P_1 , a complete rectangular patch has been reconstructed and is available, should the user wish to move the position of the dependent features, for example. This is a major advantage of functional decomposition.

26.5.2. Fitting free-form surfaces

As noted above, we wish to fit a parametric surface to each segmented primary region, based on a frame which extends beyond the region. NURBS surfaces are most commonly used [44]. Assuming that for each data point \mathbf{p}_i there is an associated point on the surface defined by the parameter pair (u_i, v_i) and a scalar weight ω_i , surface fitting can be formulated as finding the surface which minimises the following weighted least-squares expression:

$$F_{\text{lsq}}(\mathbf{S}) = \sum_i \omega_i^2 \|\mathbf{S}(u_i, v_i) - \mathbf{p}_i\|^2.$$

If we fix the knot vectors, the parameter values, and the weights, this system can be solved for the unknown quantities: the control points of the surface $\mathbf{S}(u, v)$. However, there are various problems. Firstly, appropriate parameter values u_i, v_i and weights ω_i must be chosen. Secondly, there is nothing in this approach to guarantee that the resulting shape will be smooth (fair). Thirdly, inadequate choice of the number of knots, or their values, will lead to a surface which poorly approximates the original data. Note that the requirements of smoothness and accuracy are inherently opposed.

Two basic approaches are taken to solving this problem [22,26,31,34,36,41,47,68]. The first utilises an iterative process: the region is initially approximated by a surface to within a certain precision, and then the surface is gradually smoothed by a *fairing* procedure. The second *variational approach* minimises a hybrid functional which simultaneously enforces good approximation and smoothness.

Several of these issues are now discussed further below.

Parametrisation

We now consider how to choose the parameter values at each data point. There are two distinct sub-problems: (i) a reasonably good *initial parametrisation* must be found, and (ii) as the surface fit is iteratively improved, we have to optimise not only the surface, but also the parameter values of each data point. Such *parameter correction* is particularly important for surface fitting to tight tolerances.

To find an initial parametrisation for irregularly distributed data points, usually a simple surface, a *base surface*, is created, which approximately follows the global shape of the point cloud and is determined from it. This might be a plane, a cylinder, or a bilinearly blended Coons patch [36]. For example, Ma [58] uses interactively defined section curves together with the four boundary curves to obtain a base surface. Initial parameter values are then computed by projecting data points onto the base surface. Such procedures may fail if the data points cannot be projected in an unambiguous way.

Other methods for finding an initial parametrisation use the 3D triangulation of the point cloud to find a topologically equivalent 2D triangulation. The vertex positions in 2D give the desired parameter values. These methods differ in how they perform the mapping from 3D to 2D. Harmonic parametrisations minimise the squared distances of the edge lengths [22], while Floater [29] gives a shape preserving approach locating the new vertices of the interior triangles using barycentric mappings. Greiner [31] projects the boundary points onto a best fit plane and minimises the energy of springs associated with internal edges of the triangulation; this approach was improved in [40]. Such approaches, however, are slow for large numbers of data points, and somewhat limited if the point cloud contains concavities and holes.

In summary, there is no safe, universal method for obtaining a good initial parametrisation. However, the following approach works well in many cases. After obtaining some initial parametrisation, we incrementally generate a suitable B-spline reference surface as a base surface, and use its parametrisation. Note that positional accuracy is not an issue at this stage. We start with a simple surface with a few control points (a Bézier patch), and gradually increase the smoothness or the number of control points until (i) the control net of the base surface is reasonably fair and even, without cross-overs, while (ii) the orthogonal projection of the triangles onto the reference surface keeps their original orientation [79].

The standard method for *parameter correction* is as follows. For each data point, the closest surface point is found, using Newton-Raphson steps [36,42], and its parameter values are taken as the new parameter values for the data point. If the initial parametrisation and the current approximating surface are reasonably good, a few iterations are sufficient to obtain the corrected parameter values, but if these conditions do not hold, reparametrisation can actually make things worse.

A different approach is to take the parameter values of the data points as unknown variables of the least-squares minimisation in addition to the control points of the surface, this results in a nonlinear problem. This does not seem to be an efficient or stable approach.

Smoothness terms

Besides minimising the squared distances between the data points and the surface to obtain an accurate fit, various smoothness functionals must be added to improve the quality of the shape. These may minimise the spanned surface area, the overall curvature, or variation of the curvature. In most cases, these quantities are—for simplicity—approximated using quadratic functionals of the parametric derivatives, in order to preserve linearity of the system of equations [18]. Another useful quadratic smoothness term is based on difference vectors between adjacent control points, which minimises the ‘tension’ of the control net [18]. Many such functionals are derived by simplifying physical energy terms, such as the thin plate energy [34,69].

In [30] Greiner introduced so-called *data dependent functionals*, which give better approximation than the simplified functionals. To compute the new functional he estimates differential quantities from a reference surface and the minimisation remains linear. The process works iteratively and the reference surface is always the approximating surface from the previous iteration.

To illustrate the idea of smoothing functionals, we give here a very commonly used one:

$$F_{\text{smooth}}(\mathbf{S}) = \iint (\mathbf{S}_{uu}^2 + 2\mathbf{S}_{uv}^2 + \mathbf{S}_{vv}^2) du dv.$$

This is put together with the least-squares term, to obtain the composite functional

$$F_{\text{composite}}(\mathbf{S}) = F_{\text{lsq}}(\mathbf{S}) + \lambda F_{\text{smooth}}(\mathbf{S}).$$

A crucial issue in surface fitting, which is poorly discussed in the literature, is to find an appropriate value for the *smoothness weighting factor* λ , which determines the relative importance of closeness of fit and smoothness of the final surface. A bad choice of λ may result in over-smoothing, in which case the data points will not be properly approximated, or under-smoothing, whereupon the surface will not be fair. A method for adaptively setting λ is given by Weiß [79].

A different problem also affecting smoothness is the following. When fitting a surface to a region containing large internal areas with few underlying data points, the least-squares system becomes almost singular. Control points corresponding to such areas are *weakly defined* by the data points, and large changes in their position will result in relatively small modification of the least-squares residual. These control points have very small coefficients in the linear system, so the determination of their positions is not stable. They may end up extremely far from the data points, causing serious problems in the final surface shape. Again, Weiß presents a solution to this problem [79].

Knot determination

An important subproblem in surface fitting is to determine appropriate knot vectors; this choice affects both the surface quality (more knots give greater freedom of shape) and the efficiency of fitting (more knots increase computation time).

A general approach is to add more knots iteratively as needed while the surface does not meet the required tolerance. Two main options exist for computing new knot vectors: either inserting new knots while keeping the existing ones, or inserting new knots and modifying the existing ones. We prefer the first option.

Standard knot insertion algorithms choose to insert a knot into the interval where the largest deviation between the data and the fitted surface occurs, measured either absolutely or in terms of average deviation over the interval. Typically a single knot is inserted in the middle of the selected interval. For one-dimensional problems this works well, but for B-spline *surfaces* it is often unclear whether it is better to subdivide in the u - or v -direction.

Dierkx [17] suggests a trial and error algorithm, which inserts a candidate knot in the u -direction and another in v , and accepts the one which produces the smaller residual. Unfortunately, this approach is slow as it must solve the composite system twice to insert a single knot.

We prefer an alternative approach, where more than one knot is inserted at a time. Suitable intervals for new knots are chosen according to the fitting errors and the shape of the current fitted surface. Optimal placement of the new knots within the intervals is also determined.

Our knot insertion strategy uses information about the derivatives of the normal deviation function, which measures distance in the normal direction between the current fitted

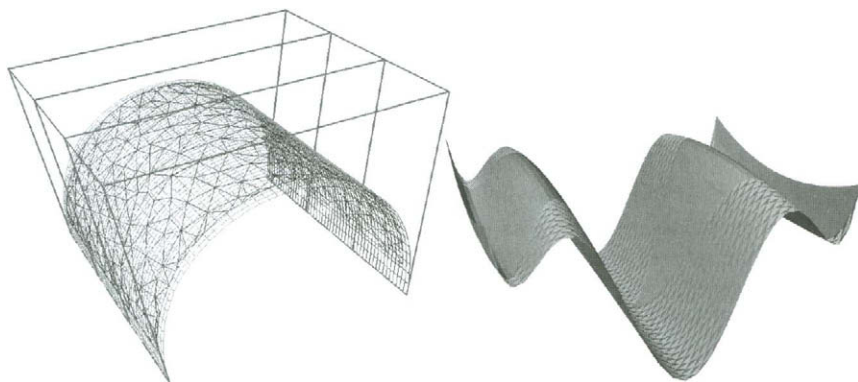


Figure 26.4. Bézier surface fitted to a half cylinder and normal deviations

surface, and the ideal surface through the data points. As an example, see Figure 26.4, where data points from a half cylinder have been approximated using a bicubic Bézier patch; the parameter directions are axial and circumferential. Since the surface is not within the desired tolerance, new knots must be introduced. The normal deviation function is shown on the right; we can see that its partial derivative in the axial direction is close to zero. This means that the parameter lines along the axial direction already follow the *shape* of the data points quite well, and are at about the same *distance* from the data points. In the circumferential direction, the situation is different. The normal deviation varies by a large amount along each parameter line: these parameter lines do not approximate the shape well. This indicates that the knot vector for the circumferential direction should be refined; new knots in the axial direction would not lead to significantly smaller residuals. Full details of this procedure can be found in [79].

26.5.3. Fitting feature surfaces

After the primary surfaces have been fitted, derived features dependent on the primary surfaces such as free-form blends, smooth steps, slots, ribs, and pockets are added to the model where necessary. These generally meet the adjacent primary surfaces with tangential continuity. Much better results can be obtained if features are treated separately than if global surfaces are fitted across the whole data set. Figure 26.5 shows an example of the poor curvature distributions obtained from global fitting in contrast to functional decomposition with feature fitting.

In general, there are many types of feature one could wish to fit. Weiß [78] considers fitting of a particular set of features characterised by a varying swept planar profile curve. Traditional surface fitting techniques can be extended to handle constraints in addition to the data points; for example, we may also wish to interpolate a discrete set of normal vectors [19,36]. Constraining a row of control points to lie in the same plane is useful for producing approximating feature surfaces of the above type. This can be done in such a way that the equations to be solved remain linear, with a special banded structure.

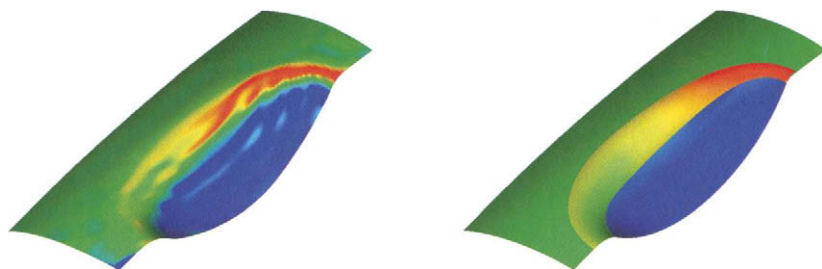


Figure 26.5. Global fitting and feature fitting between primary surfaces

Having determined a suitable set of profile curves, further optimisation is done to locate the best positions of the trimlines on the primary surfaces [78].

26.6. RECONSTRUCTING CONVENTIONAL ENGINEERING OBJECTS

This section discusses the reconstruction of conventional engineering objects, bounded by simple surfaces. Throughout this section, we illustrate the procedure using a well-known object suggested as a benchmark by Hoschek [43]—see Figure 26.2. From a functional point of view, this part is the union of three components. LEFT has rotational symmetry, MIDDLE is a truncated cone, and RIGHT is bounded by two planar faces and a cone, as well as a sequence of extruded side faces. All intersection edges are blended by constant radius rolling-ball blends. Clearly, applying free-form techniques to this case would not adequately reflect this underlying structure.

26.6.1. Segmentation

The first stage is to segment the triangulated point cloud into regions belonging to disjoint subsets which are internally smooth and to which one (or more) simple analytic surfaces can locally be fitted.

Overview

In general, segmentation is a difficult problem [74]. A standard technique is to analyse the signs of estimated mean and Gaussian curvatures [1,9,16]. Alternatively, iterative *region growing* techniques may be applied [10,56,67]. Each region is constructed by starting at a seed point in the triangulation, and adding contiguous points which are consistent with the hypothesis that all points belong to a surface of the same type with the same parameter values. While it is possible to apply the region growing paradigm to segment conventional engineering objects [59], we prefer another solution called *direct segmentation* [4,75], which is an efficient, non-iterative approach.

Direct segmentation

The structure of the boundary of an object is determined by its primary surfaces. Here we assume that each pair of adjacent primary surfaces meets in a sharp edge, a smooth edge,

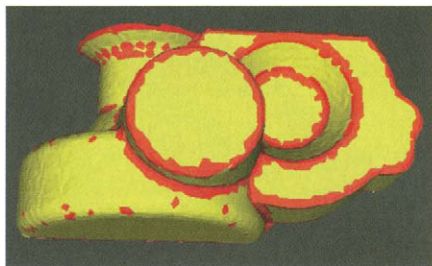


Figure 26.6. Planarity filtering



Figure 26.7. Segmented object

or a small-radius blend. The first stage of direct segmentation separates the triangulation into disjoint regions bounded by sharp edges and small blends by removing triangles at such boundaries. Each such region can be classified either as *simple*, i.e. it can be approximated by a single analytic surface, or as *multiple*. Regions of the latter type must be further subdivided using smooth internal edges.

This first stage is carried out using surface normal estimates and a *planarity* value at each point. Planarity can be determined by least-square plane fitting through the point and its neighbours [75]. The normalised fitting errors within a smooth region will be small, while for highly curved areas (at a sharp edge or small-radius blend) will be greater. Results of using this approach on the benchmark object are shown in Figure 26.6. Other methods also exist for identifying highly curved areas, based on angular variation of normals [45], or curvature estimates.

Having found each region, a set of increasingly complex tests is performed to decide whether it is a simple or multiple region. Such a sequence works well because simpler surfaces (i) occur more frequently in mechanical engineering, and (ii) can be detected in a more reliable and efficient manner. Surface detection is carried out by attempting to fit a surface of the chosen type (see Section 26.6.2), and checking if the residual errors are sufficiently small. In detail, we attempt to fit surfaces in the following order: plane, sphere, linear extrusion (subcases in order: cylinder, straight line / circular arc profile, free-form profile), cone, surface of revolution (subcases: torus, straight line / circular arc profile, free-form profile). Any remaining regions must contain multiple primary surfaces joined by smooth edges, and are handled in a second stage.

In cases where extrusions or surfaces of revolution with straight line / circular arc profiles are detected, further segmentation must be done based on the 2D profile curve, as will be explained in Section 26.6.3. Results of this process for the benchmark object are shown in Figure 26.7.

The second stage of direct segmentation decomposes multiple regions. Curvature estimates are unreliable for noisy data, so we prefer to apply *dimensionality filtering*. The normal vectors of each multiple region are mapped onto the Gaussian sphere. Normals of a planar subregion are mapped into a point cluster; those of a linear extrusion or conical region onto a circular arc. By classifying the normal vector distributions on the Gaussian

sphere as 0- or 1-dimensional, planar and extrusion surface subregions can be separated and detected [75]. Dimensionality of a set of normals is estimated by choosing an appropriate radius ρ on the Gaussian sphere, and computing the logarithm of the ratio of the number of normals within a radius 2ρ to those within a radius ρ .

The above steps identify any planar or linear extrusion subregions (as well as cones) within this multiple region. The only problem left is to check whether one or more surfaces of revolution remain. We compute candidate rotational axes using an approach suggested by Lukács [57], and cluster them to select the correct solutions.

We conclude by noting that while our idealised category of *conventional engineering object* includes only simple surfaces (and blends), in reality, such objects may also contain one or more free-form surfaces. Any regions which have not been decomposed by direct segmentation can be inferred to belong to free-form surfaces, and the techniques of Section 26.5 can be applied.

26.6.2. Fitting analytic surfaces

Overview

We now consider how to fit analytic surfaces to segmented point data. For a general overview of fitting surfaces, see [13]. We assume that a set of points lies close to a particular member of a family of surfaces parametrised by a vector of parameters \mathbf{s} . Let $d(\mathbf{s}, \mathbf{p}_i)$ be the “true” distance of the point \mathbf{p}_i from that surface. A surface going through all points is that member of the family for which the simultaneous system of equations $d(\mathbf{s}, \mathbf{p}_i) = 0$ is satisfied. Since the number of points is much greater than the number of parameters in \mathbf{s} , this system of equations is overdetermined. Furthermore, the data are noisy, so we must apply least-squares minimisation to the sum $\sum d(\mathbf{s}, \mathbf{p}_i)^2$. We use the term *geometric fitting* when the true (geometric) distance is used.

Additional non-linear constraints of the form $\mathbf{c}(\mathbf{s}) = 0$ may be needed. For example, if the family comprises the quadric surfaces, we may wish to restrict the surface being fitted to a cylinder [28]. Lagrangian multipliers may be used, leading to a difficult non-linear generalized eigenvalue problem. In some cases, the constraints may be used to reduce the problem to an unconstrained optimization problem in a lower dimensional space.

Points \mathbf{p} lying exactly on a member of the family of surfaces satisfy an implicit equation $f(\mathbf{s}, \mathbf{p}) = 0$. For a fixed \mathbf{s} , the functions f and d have the same solution set in space, but they usually behave quite differently for points which do not lie on the surface. Thus, if instead of $\sum d^2$, one minimizes $\sum f^2$, referred to as *algebraic fitting*, quite different results are usually obtained. When \mathbf{c} is quadratic and f is linear in terms of the parameters, linear generalised eigenvalue techniques work [27]; if f is non-linear, Taubin’s generalized eigenvector fit may be used [72]. However, Rosin [66] shows that choosing f carelessly can lead to severely biased estimates for \mathbf{s} .

To avoid the bias of algebraic fitting, and to overcome the complexity inherent in geometric fitting, various alternative approaches are useful. The first approximates the true distance using a *faithful representation*, generally giving a simpler and more robust non-linear problem. The second *sequential least-squares* approach finds a solution by using a series of linear steps to find first some, then other, surface parameters. Finally, by separating the point data terms from the surface parameter terms in a modification to the faithful representation, it is possible to obtain a solution more efficiently, if less

accurately. The efficiency of the last approach makes it particularly useful in *constrained fitting*, as discussed in Section 26.6.4.

Faithful representation

We say that an approximation to the true distance function $d(\mathbf{p}_i, \mathbf{s})$ is *faithful* if, firstly, the function is zero where the true distance is zero, and secondly, *at these points*, its first derivatives with respect to the surface parameters are the same as those for the true distance function.

Faithful distance functions can be obtained in various ways. A fairly general method approximates square roots in $d(\mathbf{p}_i, \mathbf{s})$: suppose the distance function is of the form $d(\mathbf{p}_i, \mathbf{s}) = \sqrt{g} - h$. We approximate d by

$$\tilde{d} = \frac{g - h^2}{2h} = d + \frac{d^2}{2h}.$$

Under very general assumptions this function is faithful to the Euclidean distance.

The corresponding quantity to be minimised for surface fitting is now

$$\sum \tilde{d}^2(\mathbf{p}_i, \mathbf{s}) = \sum \frac{(g(\mathbf{p}_i, \mathbf{s}) - h^2(\mathbf{p}_i, \mathbf{s}))^2}{4h^2(\mathbf{p}_i, \mathbf{s})}.$$

As an example, consider cylinder fitting. Let the closest point of the cylinder to the origin be $\varrho \hat{\mathbf{n}}$, let the direction of the axis of the cylinder be $\hat{\mathbf{a}}$, and let its radius be $1/k$. Let \mathbf{p} be an arbitrary point in space. Its distance from the cylinder is

$$d(\mathbf{p}, \mathbf{s}) = \left| \left(\mathbf{p} - \left(\varrho + \frac{1}{k} \right) \hat{\mathbf{n}} \right) \wedge \hat{\mathbf{a}} \right| - \frac{1}{k} = \sqrt{|\mathbf{p} - \left(\varrho + \frac{1}{k} \right) \hat{\mathbf{n}}|^2 - \langle \mathbf{p} - \left(\varrho + \frac{1}{k} \right) \hat{\mathbf{n}}, \hat{\mathbf{a}} \rangle^2} - \frac{1}{k}.$$

The corresponding faithful approximation found using the above approach is:

$$\tilde{d}(\mathbf{p}, \mathbf{s}) = \frac{k}{2} (|\mathbf{p}|^2 - 2\varrho \langle \mathbf{p}, \hat{\mathbf{n}} \rangle - \langle \mathbf{p}, \hat{\mathbf{a}} \rangle^2 + \varrho^2) + \varrho - \langle \mathbf{p}, \hat{\mathbf{n}} \rangle = \frac{k}{2} |\bar{\mathbf{p}} \wedge \hat{\mathbf{a}}|^2 - \langle \bar{\mathbf{p}}, \hat{\mathbf{n}} \rangle,$$

where $\bar{\mathbf{p}} = \mathbf{p} - \varrho \hat{\mathbf{n}}$. Note that this is linear in the curvature k if all other parameters are fixed, giving a *separable* non-linear least squares problem (see e.g. [11]). Furthermore, this distance approximation is well-behaved: in the limit that $k \rightarrow 0$, we see that $\tilde{d} \rightarrow \varrho - \langle \mathbf{p}, \hat{\mathbf{n}} \rangle$ and so the problem reduces to least-squares plane fitting.

By using trigonometric functions to parameterise $\hat{\mathbf{n}}$ and $\hat{\mathbf{a}}$, we can avoid having to separately enforce the constraints $|\hat{\mathbf{n}}| = |\hat{\mathbf{a}}| = 1$ and $\langle \hat{\mathbf{n}}, \hat{\mathbf{a}} \rangle = 0$.

Further details of faithful representations for cylinder fitting, as well as for spheres, cones and tori, may be found in Lukács [57].

Sequential least-squares methods

Assuming that we have normal vector estimates at each point, simple linear least-squares methods exist (see Section 26.6.3) for determining the translational direction for linear extrusions, and the axis for surfaces of revolution.

Thus, the parameters of a cylinder can be found by first estimating its translational direction, and then projecting the data points onto a plane perpendicular to this direction. Pratt's formulation [64] is used to fit the best circle through the projected points,

giving the radius of the cylinder via a simple eigenvalue problem. For cones, first the apex position is estimated by finding that point which minimises the sum of the squared distances from the normal planes at each data point. Next, the apex angle and axis are found simultaneously by fitting a plane on the Gaussian sphere to the ruling directions. For a torus, first the rotational axis is determined, and then all data points are rotated into a single plane containing the axis, whereupon circle fitting gives the major and minor radii.

While these non-iterative methods do not provide the same solution as correctly minimising the sum of squares of the true distances, the differences are found in practice to be small. If necessary, these solutions can be refined by using them as initial estimates for the non-linear methods given here.

26.6.3. Fitting extruded and rotational surfaces

Algorithm

Linear extrusions and surfaces of revolution are two special categories of surface whose recovery is important for conventional engineering objects. We use a three-step process due to Benkő [6]. Firstly, the direction of extrusion or the axis of revolution is determined, as appropriate. Secondly, all of the data points are put into a single plane, either by linear projection into a plane perpendicular to the extrusion direction, or by rotation into a plane containing the axis of revolution. Finally, a planar profile curve is fitted to the 2D points with the aid of a so-called *guiding polygon*. This profile curve may be composed of a series of tangentially continuous straight line segments and circular arcs, or it may be a free-form curve. In the former case, the guiding polygon helps to determine a preliminary segmentation of the 2D points [6], while in the latter case, it provides a parametrisation needed for curve fitting.

Determining the extrusion direction

The normal vectors \mathbf{n}_i to a linear extrusion surface are perpendicular to the direction of extrusion. Thus, we seek the unit vector \mathbf{d} which minimises $\sum \langle \mathbf{n}_i, \mathbf{d} \rangle^2$, i.e. the sum of squares of the cosines of angular deviations from $\pi/2$. (Using cosines rather than the angular deviations themselves makes the problem linear.) This is a well-known three dimensional eigenvalue problem.

Determining the axis of revolution

The normals to a surface of revolution intersect the axis of revolution (in a projective sense, i.e. the lines may also be parallel to the axis). As an error measure for least squares minimisation, we would ideally like to use the distance of these two lines, but this has two problems: (i) a normal parallel to the axis does not have zero error, and (ii) for a given angular deviation in normal, a greater error will result for the normal through a point further from the axis than a point nearer the axis. An alternative error measure would be to use the angle between the normal, and the plane containing the axis and the corresponding data point. This eliminates the first problem, but produces the opposite of the second problem, giving higher weighting to errors in position of points *nearer* the axis. We use a solution suggested by Pottmann and Randrup [63], and define the error to be the product of the distance and the sine of the angle between the normal line, and

the plane of the axis and the data point.

Lines are represented using Plücker coordinates. A line through a point \mathbf{p}_i in the direction \mathbf{d}_i is represented by a sextuple $(\mathbf{d}_i, \mathbf{d}_i \wedge \mathbf{p}_i)$. The latter term, denoted by $\bar{\mathbf{d}}_i$, is independent of the choice of \mathbf{p}_i . In this form, the axis may be denoted by $(\mathbf{d}_a, \bar{\mathbf{d}}_a)$.

Using this formalism, the error function is linear in the coordinates of the unknown axis. We minimise

$$\sum [\langle \bar{\mathbf{d}}_i, \mathbf{d}_a \rangle + \langle \mathbf{d}_i, \bar{\mathbf{d}}_a \rangle]^2$$

for $(\mathbf{d}_a, \bar{\mathbf{d}}_a)$ under the constraints $\|\mathbf{d}_a\| = 1$, $\langle \mathbf{d}_a, \bar{\mathbf{d}}_a \rangle = 0$. At first we ignore the second constraint, and solve the remaining system via a generalised eigenvalue problem. If greater accuracy is required, the full system is solved iteratively using this solution as an initial value. Generally only 3 or 4 iterations are needed.

Guiding polygons

Due to inaccuracies in estimating the extrusion direction, or the axis of revolution, the 2D points of the profile may form a ‘thick’ curve. Ordering the points, a necessary step for most curve fitting methods, is tricky in such a case. One possible solution to this problem is to thin the points [53]. Instead, we prefer to generate a *guiding polygon* from the original 3D triangulation. This is a coarse approximation of the profile curve which makes it possible to order the 2D points.

In principle, the guiding polygon is constructed by intersecting the triangulation with a plane orthogonal to the extrusion direction, or containing the axis of revolution respectively.

In practice, unfortunately it may not be possible to find a suitable plane which contains the *entire* profile. Thus, the guiding polygon must be constructed stepwise. We first find an initial segment of the polygon by intersecting an arbitrary triangle with a suitable plane through its barycentre. Then we extend the guiding polygon stepwise by considering the neighbours of the triangles already processed, then their neighbours in turn, and so on. Each new triangle provides *one* vertex which has not already been considered. This is rotated (or projected, as appropriate) into the profile plane. A suitable test is performed to determine whether this point overlaps the existing guiding polygon, or should be used to extend it.

If the distance between the two endpoints of the guiding polygon becomes smaller than the average distance between adjacent vertices, we stop the building process, as a closed profile curve has been found.

26.6.4. Constrained fitting for multiple curves and surfaces

Overview

A variety of approaches have been utilised for solving geometric constraint problems; see also Chapter 21. These may be broadly summarised as (i) analytical solvers, which rely on numerical methods [52], or symbolic algebra (ii) graph-theoretical based methods [37], and (iii) rule-based methods [46]; for an extensive survey see [21]. A general framework is provided by Triggs [73]. More specific work on simultaneous multiple fitting of surfaces under constraints has been given by the Edinburgh group [80–82,65]. However, none of this work adequately addresses all the specific needs of constrained surface fitting for

reverse engineering, when the constraints may be inconsistent, and huge amounts of data may be present, as we will now discuss.

Assumptions

We suppose that we wish to simultaneously fit multiple geometric entities each of known type (for example, a set of surfaces, or elements of a profile curve) to an already segmented set of data points. Furthermore, various constraints link the parameters of the geometric entities. See [5,60] for relevant constraint types, which include tangential continuity, concentricity, and perpendicularity. Fitting is done simultaneously rather than sequentially to avoid accumulation of errors.

Because they have been determined automatically, some constraints may be mutually incompatible. However, each constraint is prioritised: the goal is to satisfy as many constraints as possible which do not conflict with already satisfied higher priority constraints. The priority comes from the type of constraint, how well it is initially numerically satisfied, and so on. Contrast this with the usual case in geometric modelling [14], where, in principle, the correct number of constraints is given.

Note that the number of data points is large, and that constrained fitting methods are iterative. For efficiency reasons, it is important to keep the amount of computation per iteration to a minimum.

Auxiliary elements

To help set up the problem, *auxiliary* geometric elements are used. For example, suppose a set of planes must be concurrent through a common point. We could prescribe that all subsets of four planes each have a common point. Instead, the common point through which *all* planes must pass is defined as a new auxiliary object. Such auxiliary objects help to reduce the size of the constraint problem. They also often form part of the required output geometry needed for model building later. A list of useful auxiliary elements is given by Benkő [5].

Problem statement

The n parameter values to be found, describing both primary and auxiliary objects, are collected into a vector \mathbf{s} . The function F , which depends on the parameters \mathbf{s} , and on the distances of the data points from the objects, is to be minimized:

$$F(\mathbf{s}) = \sum_i \alpha_i \sum_j d(\mathbf{p}_{ij}, g_i)^2.$$

Here $\{g_i\}$ is the i th surface (or curve, as appropriate) defined by parameters in \mathbf{s} , \mathbf{p}_{ij} is the j th point assigned to the i th surface, and $d(\mathbf{p}_{ij}, g_i)$ is the distance of point \mathbf{p}_{ij} from surface g_i . α_i is an optional weighting in the sum.

Linking the parameter values of the primary and auxiliary objects are k constraint equations $\{c_k\}$; c_1 is the highest priority constraint. These can be written in the form $\mathbf{c}(\mathbf{s}) = 0$.

We wish to find those parameter values which minimise F subject to $\mathbf{c} = 0$, or at least, the best consistent subset of the constraint equations: we find a minimum for F by sequentially attempting to satisfy the constraints in priority order, so that any constraint consistent with those previously satisfied is also imposed.

Method

Here we outline our solution method described in detail in [5]. Initially, an unconstrained fit is made to each primary geometric entity to give the starting point for iteratively solving the constrained fitting problem. Estimates for auxiliary objects are determined from the corresponding primary objects.

The two problems: $\mathbf{c}(\mathbf{s}) = 0$, and $F(\mathbf{s})$ is minimal, are solved simultaneously by iteration. At each iteration step we use a linear approximation for \mathbf{c} , and a quadratic one for F , to compute an update \mathbf{d} to be added to the current parameter vector \mathbf{s}_0 . Taylor series expansions are used to compute \mathbf{d} :

$$\mathbf{c}(\mathbf{s}_0 + \mathbf{d}) \approx \mathbf{c}(\mathbf{s}_0) + \mathbf{c}'(\mathbf{s}_0)\mathbf{d}, \quad F(\mathbf{s}_0 + \mathbf{d}) \approx F(\mathbf{s}_0) + F'(\mathbf{s}_0)\mathbf{d} + \frac{1}{2}\mathbf{d}^T F''(\mathbf{s}_0)\mathbf{d}$$

Using these expansions, the problem to be solved may be written in the following form:

$$C\tilde{\mathbf{d}} = 0, \quad \tilde{\mathbf{d}}^T A \tilde{\mathbf{d}} \text{ to be minimised,}$$

where $\tilde{\mathbf{d}} = (d_1, \dots, d_n, 1)^T$, C and A is formed by concatenation: $C = [\mathbf{c}'(\mathbf{s}_0)|\mathbf{c}(\mathbf{s}_0)]$, while

$$A = \begin{array}{|c|c|} \hline f''(\mathbf{s}_0) & f'(\mathbf{s}_0) \\ \hline f'(\mathbf{s}_0)^T & 0 \\ \hline \end{array}.$$

To compute $\tilde{\mathbf{d}}$, we first reduce $\tilde{\mathbf{d}}$ to a lower dimensional vector \mathbf{d}^* of *independent* variables (the constrained variables are expressed in terms of unconstrained ones): $\tilde{\mathbf{d}} = M\mathbf{d}^*$. M is a matrix chosen so that $CM = 0$ (see later). Using this, we now have an unconstrained minimisation problem: minimise $\mathbf{d}^{*T} A^* \mathbf{d}^*$, where $A^* = M^T A M$. The solution for \mathbf{d}^* , and thus \mathbf{d} , is found by straightforward Newton iteration.

M is found by working through the rows of C from top to bottom using a process similar to Gaussian elimination. Suppose firstly that all constraints are independent of the previous ones, and consistent; we explain later what happens if not. Starting with the first row, we choose the variable d_l whose coefficient has the largest absolute value (for stability reasons) and express the corresponding variable in terms of the others. We may use this to eliminate d_l by writing $\tilde{\mathbf{d}} = M_1 \mathbf{d}_1$, where M_1 is a unit matrix except for row l , which takes the form

$$M_{l,*} = \left(-\frac{C_1}{C_l}, \dots, -\frac{C_{l-1}}{C_l}, -\frac{C_{l+1}}{C_l}, \dots, -\frac{C_{k+1}}{C_l} \right).$$

\mathbf{d}_1 is of one dimension less than $\tilde{\mathbf{d}}$, and is given by $\mathbf{d}_1 = (d_1, \dots, d_{l-1}, d_{l+1}, \dots, d_n, 1)$. We now construct CM_1 , and consider row 2 of (CM_1) in the analogous equation $(CM_1)\mathbf{d}_1 = 0$ to eliminate another variable, and so on.

In practice, the constraint equations may not be independent. If all the entries in a given row m are zero, then the constraint equation follows from the previous ones, and elimination cannot be performed. If the constant term in a given row m is not zero but all the others are, then the constraint contradicts the previous ones, and so this constraint equation is discarded. In either case, we proceed to the next row without further processing.

A minor complication is that a single *geometric* constraint may require more than one constraint *equation*. We must remove *all* constraint equations involved when a geometric constraint is found to be inconsistent and back-up the computation to the row after the highest row which was discarded.

Efficient representation

The function being minimised is a weighted sum of squared distances between points and geometric objects. Derivatives of the error function must be evaluated during each iteration. For efficiency, we wish to evaluate as much as possible in a once-only preprocessing step before iteration commences. This may be achieved by using suitable object and constraint representations combined with appropriate approximations to distance functions.

Our objective is to write the approximate signed distance between an object (described by parameter vector \mathbf{s}) and a point \mathbf{p} in the form

$$d(\mathbf{p}, \mathbf{s}) = S(\mathbf{s})^T P(\mathbf{p}) = P(\mathbf{p})^T S(\mathbf{s}),$$

where S and P are vector valued functions. If we do this, the function to be minimised (for a single object) is

$$F(\mathbf{s}) = \sum d(\mathbf{p}, \mathbf{s})^2 = S(\mathbf{s})^T \left(\sum P(\mathbf{p})P(\mathbf{p})^T \right) S(\mathbf{s}).$$

Using such a separable form, we need to compute the matrix $M_{\mathbf{p}} = \sum P(\mathbf{p})P(\mathbf{p})^T$ only once, as it depends only on the data points, and not on the parameters of the object.

The first and second derivatives of $F(\mathbf{s})$ are then $F' = 2S^T M_{\mathbf{p}} S'$, $F'' = 2(S'^T M_{\mathbf{p}} S' + S^T M_{\mathbf{p}} S'')$. These may still be time consuming to compute, so we attempt to find descriptions where S is *simple*. If S is just a row vector of parameters, i.e. $S(\mathbf{s}) = (s_1, s_2, s_3, \dots)$, rather than a vector containing more complicated functions of the parameters, then the derivative formulae are simple, and take the form $F'(\mathbf{s}) = 2M_{\mathbf{p}} \mathbf{s}$, $F''(\mathbf{s}) = 2M_{\mathbf{p}}$. In fact, one may always choose a new representation $\tilde{\mathbf{s}} = S(\mathbf{s})$ in this way, but it may contain more variables, linked by potentially complicated constraints.

We illustrate these ideas with the concrete example of fitting points to a circle. The distance between a point \mathbf{p} and a circle with centre \mathbf{o} and radius r is $\|\mathbf{p} - \mathbf{o}\| - r$. Because this function contains a square root, it is not in separable form. However, if we use instead the faithful approximation $((\mathbf{p} - \mathbf{o})^2 - r^2) / 2r = (\mathbf{p}^2 - 2\langle \mathbf{p}, \mathbf{o} \rangle + \mathbf{o}^2 - r^2) / 2r$, and now choose $S(\mathbf{s}) = S(\mathbf{o}, r) = (1/2r, -o_x/r, -o_y/r, (\mathbf{o}^2 - r^2)/2r)$, $P(\mathbf{p}) = (\mathbf{p}^2, p_x, p_y, 1)$, we now *do* have a separable form.

We may do better if we change the circle parameters to $S(\mathbf{s}) = (\kappa, o'_x, o'_y, A)$; the extra variable is linked by the equations $\mathbf{o}' = -2\kappa\mathbf{o}$ and $A = (\mathbf{o}^2 - r^2)/2r$, giving $\mathbf{o}'^2 - 4\kappa A = 1$. Computing derivatives is now simpler: no divisions are required. This representation also has the advantage that it is still valid when the circle degenerates to a straight line.

A detailed list of object representations, constraint equations, and efficient approximate distance functions is given by Benkő in [5] for all the simple analytic surfaces we consider in this Chapter. He also formulates appropriate equations for a wide variety of common engineering constraints, which makes it possible to solve the constrained fitting problem in practice.

A simple example taken from the benchmark object is shown in Figure 26.8, which shows the profile curve of the LEFT part of the object. Note that tangential continuity constraints have been enforced on the straight line segments and circular arcs involved.

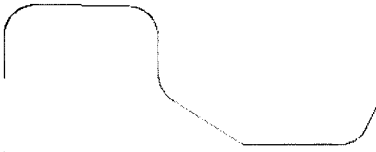


Figure 26.8. Reconstructed rotational profile

26.6.5. Reconstructing blend surfaces

Problem statement

We now describe how blending surfaces may be recovered and incorporated into solid models. We assume that the primary surfaces adjacent to each blend have already been reconstructed, and meet in an edge. We assume that each blend is a fixed-radius rolling-ball blend. Our approach is to determine its radius, and then call upon a solid modelling kernel to construct an appropriate blend surface for incorporation into the B-rep model.

Data points belonging to blending surfaces are identified during segmentation: they are the points not belonging to any primary surface. We discard points in the neighbourhood of vertices, leaving a separate point set for each edge blend. We do not explicitly reconstruct vertex blends, since for small blends these are represented by only a few data points. Vertex blends instead are constructed by the solid modeller as by-products automatically connecting edge blends. Setback vertex blends provide a reasonable transition surface in the case of arbitrary edge blend configurations [76].

Approaches

Various approaches for estimating blend radii are thoroughly analysed in [48]. In summary, these may be divided into methods which work for general types of blend, and methods which work for particular types of blend surface. In the former category are: (i) averaging discrete estimates of principal curvature at each blend point, (ii) the iterative spine method, and (iii) the maximum ball method, while in the latter category are: (iv) direct surface fitting when the blend is a simple analytic surface, and (v) fitting cross-section circles when the blend is a swept profile, with or without tangency constraints to the adjacent primary surfaces. We describe approach (iii) further below, as it is generally the best method.

A different approach uses medial-axis-transforms [15] (see Chapter 19). The medial surface of the object is constructed from the triangulation of the point cloud, and each Delaunay tetrahedron is classified according to its type. This allows medial edges to be constructed, each of which correspond to the centre line of a blended pair of faces. The average circumspherical radius of the associated tetrahedra approximates the blend radius.

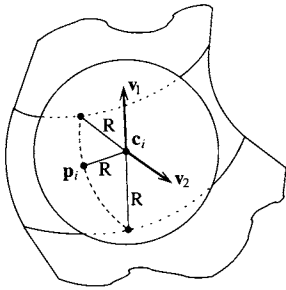


Figure 26.9. The maximum ball for point \mathbf{p}_i

The maximum ball method

This method works by computing an estimate for the blend radius at each measured point of the blend. For each such point \mathbf{p}_i , we attempt to reconstruct the position of the rolling ball which generates the point \mathbf{p}_i . This is the largest sphere which contains \mathbf{p}_i and is tangent to both primary surfaces. We average the radii of these largest spheres (or take the median value) to give an estimate of the blend radius.

We denote the centre of the current ball by \mathbf{c}_i . It must satisfy the following two conditions (see Figure 26.9). The distances $d_1(\mathbf{c}_i)$, $d_2(\mathbf{c}_i)$ to the two primary surfaces and $|\mathbf{c}_i - \mathbf{p}_i|$ must all be equal to R , the radius of the ball. The vectors $\mathbf{v}_1(\mathbf{c}_i)$, $\mathbf{v}_2(\mathbf{c}_i)$ from \mathbf{c}_i to the corresponding points on the primary surfaces, and $\mathbf{c}_i - \mathbf{p}_i$ must be in the same plane. Note that there are two centre points with these properties. As well as the one we seek, the other is the centre of the *smallest* sphere which contains \mathbf{p}_i and is tangent to the two primary surfaces.

This method has been generalised to reproduce varying radius rolling ball blends [50].

26.6.6. Building solid models

In order to build the final B-rep model, we have to join together the fitted surfaces at explicit edges and vertices. For regions with tangentially adjacent surfaces, suitable edges and vertices have been computed by the constrained fitting process. In other cases, they must now be found by intersection of adjacent surfaces. Further details of our approach, and examples, can be found in [4].

Fattening

Model building starts by creating a *region adjacency* data structure, which gives the topology of the final B-rep model. During segmentation, triangles not classified as belonging to a particular region were ignored. Now, during *fattening*, we classify these triangles to remove the gaps between the disjoint regions—these are extended outwards until all triangles belong to some region.

In each pass, unclassified triangles are considered which have at least one common edge with the classified triangles. Such triangles, if within a distance tolerance, are allocated to the nearest region owning a classified neighbour. If no further triangles can be added which

satisfy the tolerance criterion, triangles are added based only on adjacency information. In each pass, all regions are simultaneously grown in a reasonably even manner.

After fattening, each edge of the triangulation shared by two adjacent regions forms a polyline, which topologically represents an edge in the final model. Vertices of the triangulation shared by at least three regions, at the ends of the polylines, correspond to vertices in the final model. Connected sets of triangles which logically all belong to the same region correspond to faces.

Edges and vertices

Sharp edge curves are computed by calling a surface-surface intersection routine of the modelling kernel. For efficiency reasons, and to avoid ambiguities, we compute a bounding box for each edge, based on the related approximating polyline. We always leave processing of very short and silhouette edges (see below) to as late as possible as they are ill-defined.

Vertices where three surfaces meet are well-determined and are computed by curve-surface intersection; care must be taken if one of the edges meets the opposite face with tangential continuity. Silhouette vertices are computed only approximately, constrained by the underlying edge curve.

If constraints were not enforced during surface fitting, vertices with more than three edges need special care: the intersection geometry may produce a topology which is inconsistent with the topology deduced by the fattening procedure. We must update the topology accordingly. Careful ordering of the intersections helps to establish the correct topology.

For *incomplete* models, i.e. whose surface does not form a closed shell, we must terminate the model using silhouette edges at the edge of the triangulation. Edge loops of the corresponding faces are completed by constructing approximate edges from the border polylines of the triangulation. A piecewise linear approximation to the polyline (with larger linear pieces) is determined and projected onto the underlying surface, giving a sequence of small edges lying on this face. If the face involved is a plane, a better result may be obtained by constructing a composite curve made up of straight and circular segments using the same methods as for smooth profile curve reconstruction.

Stitching

We now create the complete topological structure by stitching together the faces, edges and vertices. This is straightforward, since the previous phases assure the consistency of the geometrical and topological entities, and is performed by algorithms of the underlying solid modelling kernel. Blends are then added as appropriate. The final model produced for the benchmark object before blending is shown in Figure 26.10; after blending in Figure 26.11.

Alternative method

A completely different approach may also be taken to combine the data from different views into a single B-rep model. Point clouds from different views are *not* merged, and a prismatic B-rep model is formed from each view, reproducing the faces visible in that view. Such prismatic models can then in principle be *unioned* together. However, to avoid many

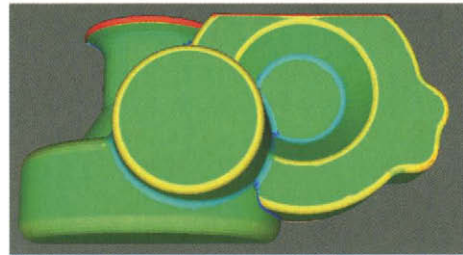
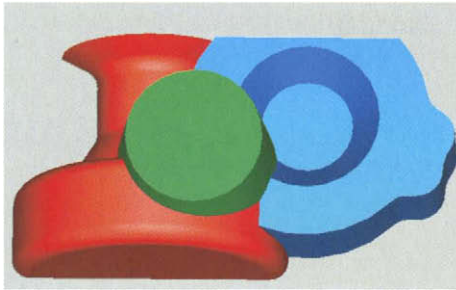


Figure 26.10. Test object without blends

Figure 26.11. Test object with blends

small facets and edges, the individual models must be carefully beautified and features matched before combination. This approach has the advantage that it always produces closed volumes (assuming each prism has a limiting back face).

26.6.7. Beautifying solid models

Beautification is performed by firstly automatically identifying the constraints required and secondly, imposing them. Enforcing constraints between geometric elements can either be done at the surface fitting stage, or as a post-process after model building. The rationale for doing it as a post-process is that an explicit topological and geometric data structure now exist; also the original data points can now be discarded, so much less data has to be processed. In some sense, discarding the original data does not matter: we may often be able to hypothesise correct parameter values for geometric entities from the approximate ones present in the model. For example, an angle of 88° should probably be 90° . Even if draft angles are present, it is not obvious whether we should recover the actual model including the draft angles, or the ideal model before draft angles were added!

Finding constraints

We may classify constraints into three broad categories. The most local constraints are ones affecting a single geometric entity (e.g. the radius of a cylinder should be an integer constant), or pairs of geometric entities (e.g. coaxiality of two cylinders). At the other end of the scale, the whole geometric object may possess a symmetry, e.g. a plane of mirror symmetry. In between these extremes are patterns of repeated features, such as a ring of circular through-holes of equal radii. An analysis of a range of engineering shapes has been carried out to determine which constraints and symmetries are of common occurrence in typical conventional engineering components [60]. An efficient algorithm has been developed to detect symmetries [61,71] in a solid model of such an object, by analysing a suitable set of sample points taken from its boundary. This algorithm has the advantage of not explicitly requiring an input tolerance to work to, but decides on suitable tolerance levels as it operates. Another algorithm [54,55] has been developed which finds constraints on geometric elements and groups of geometric elements, including repeated patterns of simple features. However, in general, feature recognition is a vast

subject—see Chapter 21, and finding constraints between arbitrary features which are only approximately represented is a tricky problem.

Imposing constraints

Imposing constraints is a much more difficult problem than identifying them, particularly when the constraints may be inconsistent. This is still a topic of current research. Imposing symmetry on a complete object seems to be an easier problem, if only because it does not involve any inconsistent constraints.

When beautifying a model, it is generally not sufficient to modify parameters of geometric entities one at a time. Vertices at each end of any very short edge should also be merged, for example. The question then arises as to whether *any* consistent configuration of new geometry exists which satisfies the requirements, and also how to construct it if it does. Related work on resolvable representations [70] goes some way to answering this question, but only for polyhedra of genus zero.

26.7. CONCLUSION

Reverse engineering is a vast subject, and we have only been able to present a conceptual overview of many of the topics in this chapter, together with a little more detail in some places to give insight into a few problems of particular significance. Interested readers will certainly need to consult the cited literature.

Many advances in the subject have been made in the last few years, although at present there are two clearly separate tracks for free-form and conventional engineering objects.

The research state of the art is that boundary representation models of moderately complex conventional engineering objects can be constructed automatically, and high-quality surfaces representing free-form objects can also be recovered with user-guided functional decomposition. However, careful setting of various system parameters is necessary to obtain satisfactory results.

Many of the algorithms described here will enable more powerful commercial systems to be developed over the next few years, but some algorithms need to be made more robust and general, and other areas are still the subject of research.

REFERENCES

1. L. Aloul and R. van Damme. Polyhedral metrics in surface reconstruction: tight triangulations. In T. Goodman and R.R. Martin, editors, *The Mathematics of Surfaces VII*, pages 309–336, Information Geometers, Winchester, 1997.
2. N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. *Computer Graphics (SIGGRAPH '98)*, pages 415–421, 1998.
3. C. Bajaj, F. Bernardini, J. Chen, and D. Schikore. Automatic reconstruction of 3D CAD models. In W. Strasser, R. Klein, and R. Rau, editors, *Proc. Int. Conf. on Theory and Practice of Geometric Modeling II*, Blaubeuren, Germany, October 1996.
4. P. Benkő, R.R. Martin, and T. Várady. Algorithms for reverse engineering boundary representation models. *Computer-Aided Design*, 33(11):839–851, 2001.
5. P. Benkő, G. Kós, T. Várady, L. Andor, and R. Martin. Constrained fitting in reverse engineering. *Computer Aided Geometric Design*, to appear 2001.

6. P. Benkő and T. Várady. Best fit translational and rotational surfaces for reverse engineering shapes. In R. Cipolla and R. Martin, editors, *The Mathematics of Surfaces IX*, pages 70–81, Springer-Verlag, 2000.
7. R. Bergevin, M. Soucy, H. Gagnon, and D. Laurendeau. Towards a general multi-view registration technique. *IEEE PAMI*, 18(5):540–547, 1996.
8. F. Bernardini. *Automatic Reconstruction of CAD Models and Properties from Digital Scans*, PhD thesis, Purdue University, 1997.
9. P.J. Besl. *Surfaces in Range Image Understanding*, Springer-Verlag, 1988.
10. P.J. Besl and R.C. Jain. Segmentation through variable-order surface fitting. *IEEE PAMI*, 10(2):167–192, 1988.
11. Å. Björk. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, Philadelphia, 1996.
12. J-D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.
13. R.M. Bolle and B.C. Vemuri. On three-dimensional surface reconstruction methods. *IEEE PAMI*, 13(1):1–13, 1991.
14. B. Brüderlin and D. Roller. *Geometric Constraint Solving and Applications*. Springer-Verlag, 1998.
15. A. Casement, C. Armstrong and A. Middleditch. *Identifying Blends and Axisymmetric Profiles in Model Reconstruction Using the Appropriate Medial Axis*. Technical Report, Dept. of Mechanical Engineering, Queen's University, Belfast, UK, 2000.
16. P. Csákány and A.M. Wallace. Computation of local differential parameters on irregular meshes. In R. Cipolla and R. Martin, editors, *The Mathematics of Surfaces IX*, pages 19–33, Springer-Verlag, 2000.
17. P. Dierckx. *Curve and Surface Fitting with Splines*. Clarendon Press, Oxford, 1995.
18. U. Dietz. *Erzeugung glatter Flächen aus Meßpunkten*. Fachbereich Mathematik, Technische Hochschule Darmstadt, Preprint-Nr. 1717, 1995.
19. U. Dietz. Fair surface reconstruction from point clouds. In M. Daehlen, T. Lyche, and L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces II*, pages 79–86, Vanderbilt University Press, 1998.
20. C. Dorai, G. Wang, A.K. Jain, and C. Mercer. Registration and integration of multiple object views for 3D model construction. *IEEE PAMI*, 20(1):83–89, 1998.
21. C. Durand. *Symbolic and Numerical Techniques for Constraint Solving*. PhD thesis, Purdue University, Computer Science Department, 1998.
22. M. Eck and J. Hadenfeld. Local energy fairing of B-spline curves. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Computing 10*, Springer-Verlag, 1995.
23. M. Eck and H. Hoppe. Automatic reconstruction of B-spline surfaces of arbitrary topological type. *Computer Graphics (SIGGRAPH 96)*, 30:325–334, 1996.
24. H. Edelsbrunner and E. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.
25. H. Edelsbrunner. *Weighted Alpha Shapes*. Technical Report UIUCDCD-R-92-1760, Comp. Sci. Dept., Univ. Illinois, Urbana, Ill, 1992.
26. G. Farin and N. Sapidis. Curvature and the fairness of curves and surfaces. *IEEE Comp. Graph. Applic.*, 3:52–57, 1989.
27. A.W. Fitzgibbon, M. Pilu, and R.B. Fisher. Direct least-square fitting of ellipses. In

- Proceedings of the 13th ICPR Conference*, Vienna Austria, August 1996.
28. A.W. Fitzgibbon, D.W. Eggert, and R.B. Fisher. High-level CAD model acquisition from range images. *Computer-Aided Design*, 29(4):321–330, 1997.
 29. M.S. Floater. Parameterization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14:231–250, 1997.
 30. G. Greiner. Variational design and fairing of spline surfaces. *Computer Graphics Forum*, 13(3):143–154, 1994.
 31. G. Greiner and K. Hormann. Interpolating and approximating scattered 3D data with hierarchical tensor product B-splines. In A. Le Méhauté, C. Rabut and L.L. Schumaker, editors, *Surface Fitting and Multiresolution Methods*, pages 163–172 Vanderbilt University Press, 1997.
 32. B. Guo, J. Menon, and B. Willette. Surface reconstruction using alpha shapes. *Computer Graphics Forum*, 16(4):177–190, 1997.
 33. B. Guo. Surface reconstruction: from points to splines. *Computer-Aided Design*, 29:269–277, 1997.
 34. H. Hagen and P. Santarelli. Variational design of smooth B-spline surfaces. In H. Hagen, editor, *Topics in Surface Modeling*, pages 85–94. SIAM, 1992.
 35. P.S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. *Course Notes, Course 25, SIGGRAPH 97*, 1997.
 36. T. Hermann, Z. Kovács, and T. Várady. Special applications in surface fitting, In W. Strasser, R. Klein, and R. Rau, editors, *Geometric Modeling: Theory and Practice*, pages 14–31, University of Tübingen, 1997.
 37. C.M. Hoffmann and R. Joan-Arinyo. Symbolic constraints in constructive geometry. *Journal of Symbolic Computation*, 23:287–300, 1997.
 38. A. Hoover et al.. An experimental comparison of range image segmentation algorithms. *IEEE PAMI*, 18(7):673–689, 1996.
 39. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganised points. *Computer Graphics (SIGGRAPH 92)*, 26:71–78, 1992.
 40. K. Hormann and G. Greiner. MIPS: an efficient global parametrization method. In P.-J. Laurent, P. Sablonniere, and L.L. Schumaker, editors, *Curve and Surface Design, Saint-Malo 1999*, pages 153–162. Vanderbilt University Press, 1999.
 41. J. Hoschek. Smoothing of curves and surfaces. *Computer-Aided Design*, 2:97–105, 1985.
 42. J. Hoschek. Intrinsic parametrization for approximation. *Computer Aided Geometric Design*, 5:27–31, 1988.
 43. J. Hoschek and W. Dankwort, Eds., *Reverse Engineering*. B.G. Teubner, Stuttgart, 1996.
 44. J. Hoschek and P. Kaklis, Eds.. *Advanced Course on FAIRSHAPE*. B.G. Teubner, Stuttgart, 1996.
 45. J. Hoschek, U. Dietz and W. Wilke. A geometric concept of reverse engineering of shape: approximation and feature lines. In M. Dæhlen, T. Lyche, and L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces II*, pages 253–262. Vanderbilt University Press, 1998.
 46. R. Joan-Arinyo and A. Soto. A correct rule-based geometric constraint solver. *Com-*

- puters and Graphics*, 21(5):599–609, 1997.
47. E. Kaufmann and R. Klass. Smoothing surfaces using reflection lines for families of splines. *Computer-Aided Design*, 20:312–316, 1988.
 48. G. Kós, R.R. Martin, and T. Várady. Recovery of blend surfaces in reverse engineering. *Computer Aided Geometric Design*, 17:127–160, 2000.
 49. G. Kós. An algorithm to triangulate surfaces in 3D using unorganised point clouds, *Computing Suppl*, 14:219–232, 2001.
 50. G. Kós. Recovering variable radius rolling ball blends in reverse engineering. In M. Shpitalni, editor, *2000 Int. CIRP Design Seminar*, (Haifa, 16–18 May, 2000), pages 469–474, 2000.
 51. P. Krsek, T. Pajdla, V. Hlavác, and R.R. Martin. Range image registration driven by a hierarchy of surface differential features. *Proc. 22nd Workshop of the Austrian Association for Pattern Recognition*, pages 175–183, May 1998.
 52. H. Lamure and D. Michelucci. Solving geometric constraint systems by homotopy. In *Third ACM Symposium on Solid Modeling and its Applications*, pages 263–269, ACM Press, 1995.
 53. I.K. Lee. *Curve reconstruction from unorganised points*. Technical Report No. 55, Institute for Geometry, Technical University of Vienna, 1998.
 54. F.C. Langbein, B.I. Mills, A.D. Marshall, and R.R. Martin. Finding Approximate Shape Regularities in Reverse Engineered Solid Models Bounded by Simple Surfaces. In *Proc. Sixth ACM Symposium on Solid Modeling and Applications*, pages 206–215, Ann Arbor, Michigan, June 6–8, 2001.
 55. F.C. Langbein, B.I. Mills, A.D. Marshall, and R.R. Martin. Recognizing geometric patterns for beautification of reconstructed solid models. In *Proc. Shape Modelling International*, pages 10–19, Genova, Italy, May 2001.
 56. A. Leonardis, A. Jaklic, and F. Solina. Superquadrics for segmenting and modeling range data. *IEEE PAMI* 19(11):1289–1295, 1997.
 57. G. Lukács, A.D. Marshall, and R.R. Martin. Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation. In H. Burkhardt and B. Neumann, editors, *Computer Vision–ECCV98*, Lecture Notes in Computer Science, Springer-Verlag, 1998.
 58. W. Ma and J.P. Kruth. Parameterization of randomly measured points for least squares fitting of B-spline curves and surfaces. *Computer-Aided Design*, 27(9):663–675, 1995.
 59. A.D. Marshall, G. Lukács, and R.R. Martin. Robust segmentation of primitives from range data in the presence of geometric degeneracy. *IEEE PAMI* 23(3):304–314, 2001.
 60. B.I. Mills, F.C. Langbein, A.D. Marshall, and R.R. Martin. *Estimate of Frequencies of Geometric Regularities for Use in Reverse Engineering of Simple Mechanical Components*. <http://ralph.cs.cf.ac.uk/papers/Geometry/survey.pdf>
 61. B.I. Mills, F.C. Langbein, A.D. Marshall, and R.R. Martin. Approximate Symmetry Detection for Reverse Engineering. In *Proc. of Sixth ACM Symposium on Solid Modeling and Applications*, pages 241–248, Ann Arbor, Michigan, June 6–8, 2001.
 62. T. Pajdla and L. Van Gool. Matching of 3-D curves using semi-differential invariants. In *5th International Conference on Computer Vision*, ppages 390–395, IEEE Computer Society Press, 1995.

63. H. Pottmann and T. Randrup. Rotational and helical surface approximation for reverse engineering. 60(4):307–322, *Computing*, 1998.
64. V. Pratt. Direct least-squares fitting of algebraic surfaces. *Computer Graphics (SIGGRAPH 87)*, 21(4):145–152, 1987.
65. C. Robertson, R.B. Fisher, D. Corne, N. Werghe, and A.P. Ashbrook. Investigating evolutionary optimisation of constrained functions to capture shape descriptions from range data. In R. Roy, T. Furuhashi, and P.K. Chawdhry, editors, *Advances in Soft Computing—Engineering Design and Manufacturing*, pp. 455–466, Springer-Verlag, 1999.
66. P.L. Rosin. Analysing error of fit functions for ellipses. *Pattern Recognition Letters* 17:1461–1470, 1996.
67. N.S. Sapidis and P.J. Besl. Direct construction of polynomial surfaces from dense range images through region growing. *ACM Transactions on Graphics*, 14(2):171–200, 1995.
68. B. Sarkar and C.-H. Menq. Smooth-surface approximation and reverse engineering. *Computer-Aided Design*, 23(9):623–628, 1991.
69. V. Skytt. Two methods for surface reconstruction from given point distributions. In J. Hoschek and W. Dankwort, editors, *Reverse Engineering*, pages 159–168, Teubner, Stuttgart, 1996.
70. K. Sugihara. Resolvable representation of polyhedra. *Discrete and Computational Geometry*, 21:243–255, 1999.
71. S.J. Tate, G.E.M. Jared, and K.G. Swift. Detection of symmetry and primary axes in support of proactive design for assembly, In W.F. Bronsvort and D.C. Anderson, editors, *Proc. 5th ACM Symposium on Solid Modeling and Applications*, pages 151–158, ACM Press, 1999.
72. G. Taubin. Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation. *IEEE PAMI*, 13(11):1115–1138, 1991.
73. B. Triggs. Optimal estimation of matching constraints. In R. Koch and L. Van Gool, editors, *3D Structure from Multiple Images of Large-scale Environments (SMILE 98)*, Lecture Notes in Computer Science, Springer-Verlag, 1998.
74. T. Várady, R.R. Martin, and J. Cox. Reverse engineering of geometric models—an introduction. *Computer-Aided Design*, 29(4):255–268, 1997.
75. T. Várady, P. Benkó, and G. Kós. Reverse engineering regular objects: simple segmentation and surface fitting procedures. *Int. Journal of Shape Modeling*, 4:127–141, 1998.
76. T. Várady and C.M. Hoffmann. Vertex blending: problems and solutions. In M. Dæhlen, T. Lyche, and L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces II*, pages 501–528, Vanderbilt University Press, 1998.
77. R.C. Veltkamp. Boundaries through scattered points of unknown density. *Graphical Models and Image Processing*, 57(6):441–452, 1995.
78. V. Weiß, G. Renner, and T. Várady. Reconstruction of swept free-form features from measured data points. In *Proc. 32nd CIRP International Seminar on Manufacturing Systems*, pp. 33–41, 1999.
79. V. Weiß, L. Andor, G. Renner, and T. Várady. Advanced surface fitting techniques.

Computer Aided Geometric Design, to appear 2001.

80. N. Werghi, R.B. Fisher, C. Robertson, and A.P. Ashbrook. Modelling objects having quadric surfaces incorporating geometric constraints, In *Proc. 5th European Conference on Computer Vision*, Vol. II, pages 185–201, 1998.
81. N. Werghi, R.B. Fisher, C. Robertson, and A. Ashbrook. Object reconstruction by incorporating geometric constraints in reverse engineering. *Computer-Aided Design*, 31:363–399, 1999.
82. N. Werghi, R.B. Fisher, C. Robertson, and A. Ashbrook. Improvement of quadric surface estimation by global fitting. *International Journal of Shape Modelling*, 6(1): 65–78, 2000.

Chapter 27

Vector and Tensor Field Visualization

Gerik Scheuermann and Hans Hagen

27.1. INTRODUCTION

“Visualization transforms data into images that efficiently and accurately represent information about the data” [49, p.83]. This definition describes the visualization task as a transformation process like the one in Figure 27.1.

It is obvious on one side that one needs knowledge about the data and its underlying context to do the job. On the other side, it is necessary to use the whole repertoire of image generation to be efficient and accurate as the definition demands. In this chapter, we understand visualization as transformation of digital, numerical data into digital images by using computer graphics. This kind of visualization has been formed into a scientific discipline by a report of McCormick and DeFanti [34]. If you are looking for an example of classical visualization techniques, Merzkirch [35] provides an excellent survey on experimental flow visualization. The data in our applications stems from natural scientists or engineers doing measurements during experiments, observing natural or technical phenomena or simulating experiments by numerical calculations. The information about the data serves three main purposes [45], namely

- to get an impression of the experiment or simulation;
- to inspire the development of new and better theories;
- to verify a new theory or model.

All three purposes are directed towards the natural scientist or engineer, so visualization is an inherent interdisciplinary endeavour from the perspective of a computer scientist. In the natural and engineering sciences, three types of numerical variables make up nearly

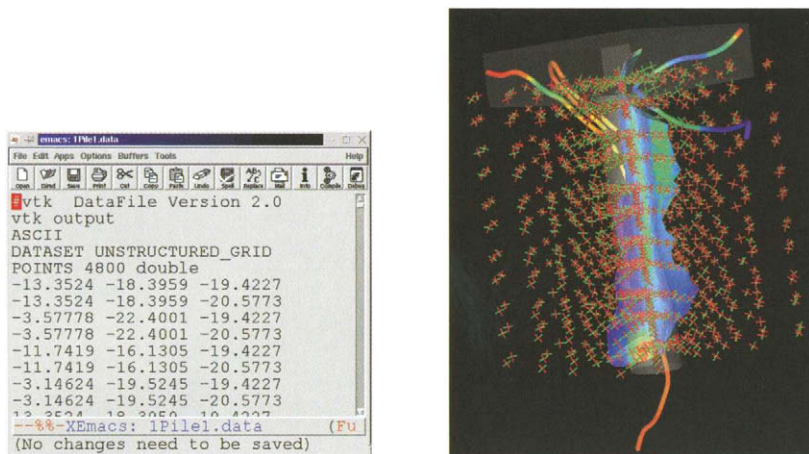


Figure 27.1. Visualization transforms data into images. Here, tensor data together with geometry is transformed into a computer generated image.

all numerical data: scalars, vectors, and tensors. Typical examples for scalar variables are temperature, pressure, density, and electrical charge. Vector variables are velocity, vorticity, magnetic field, electric field, force, and any kind of scalar gradient like temperature gradient. Tensor variables describe for example stress, strain or rate of deformation (in fluids). There are a lot of good visualization concepts for scalar data like coloring, height fields, contouring, and volume visualization, but we like to concentrate in this chapter on vector and tensor data. A practical description of all kinds of visualization algorithms can be found in [49].

27.2. VISUALIZATION PROCESS

The transformation of data into images is a process with three steps, usually described as a visualization pipeline. A typical model is given by the three intermediate steps in Figure 27.2. The **data generation phase** stands outside the visualization process. It

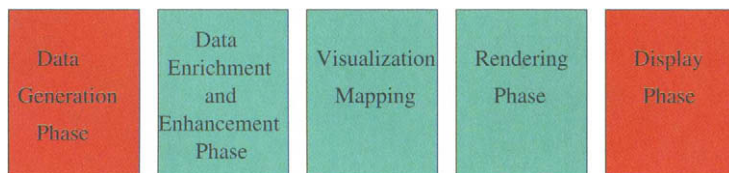


Figure 27.2. The visualization pipeline.

means the creation of numerical data by simulation, measurements during experiments or observation of natural phenomena. The **data enrichment and enhancement phase** modifies the data to reduce its amount or improve the information content. Domain transformations, interpolation, sampling, and noise filtering are typical operations in this phase. The **visualization mapping** section is the heart of the whole transformation. The application data is mapped to visual primitives and attributes. This chapter will give an overview of successful techniques for vector and tensor data. The **rendering phase** does the usual computer graphics operation of creating an image on the screen from the graphics primitives and attributes combined with a camera model, lighting operations, anti-aliasing filtering, and hidden surface removal. Finally, the **display phase** shows the image on the screen or prints it on paper.

27.3. DATA SET TYPES AND INTERPOLATION METHODS

The data sets in visualization consist usually of two parts. The first part contains the grid and the second part data values associated with the grid. In our case the data values will always be vectors or tensors. The types of the data sets are not determined by the visualization process, but by the data generation process. This is an important issue for every visualization that takes place, especially since the data is usually discrete and an interpolation has to take place for all but the simple point-based direct visualization methods.

In numerical simulations, two types of algorithms still produce most of the data, finite difference and finite element methods. Both create a set of points, vertices, and associate the data in most cases with these points. Besides this, a grid consisting of a large number of cells is created. Each cell contains a small number of vertices. The cells do not intersect except at their boundaries and all cells fill the domain of the simulation. Finite difference methods prefer regular grids like cartesian and rectilinear grids, sometimes deformed into so-called structured grids by a mapping from a cartesian computational domain into the physical domain where the simulation is defined. Figure 27.3 gives a typical example. Finite element methods are more flexible, since they allow different kinds of cells in one

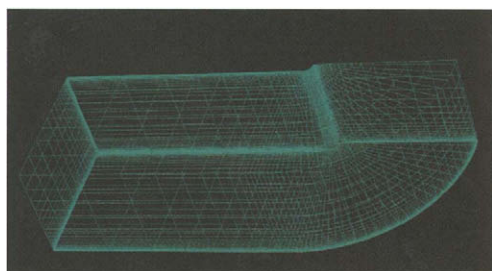


Figure 27.3. A typical structured grid used for a CFD simulation.

simulation and are able to fill complex geometries with cells. Typical cell types are given in Figure 27.3. A grid is called unstructured, if the cell neighbors are not given implicit

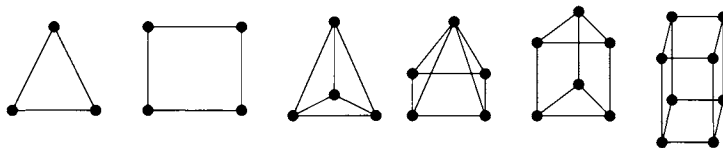


Figure 27.4. Linear two-dimensional and three-dimensional finite elements. There are also non-linear elements with additional points at edges, faces and in the center.

by the indexing of the cells as it is possible for example for cartesian grids. There is a large variety of interpolation functions available, most often coupled with the cell types. Surveys on this topic are found in introductory texts on finite element methods [3,37]. Grid types in numerical simulations can be a lot more complicated by overlapping grids, several grid blocks with different grid types and multilevel techniques. A nice survey about grid types is given by [21]. Data sets from measurements or experiments are often regular since they are the result of image-generating processes like computer tomography (CT). If they come from other sources, scattered data is a typical type. Before a visualization can take place, one has to do scattered data interpolation. A description of this research area can be found in the chapter on Scattered Data Techniques in this handbook.

27.4. DIRECT MAPPINGS TO GEOMETRIC PRIMITIVES

The easiest way to create a visualization with the computer is to map the data values to geometric primitives. These primitives are collected into a scene that is presented by standard computer graphics techniques. If you are not familiar with computer graphics, you may consult a computer graphics textbook [17]. These direct visualization methods can be distinguished by the domain of information that each primitive represents. We will discuss point-based, line-based, surface-based, and volume-based methods in this section.

27.4.1. Point-based methods

Vectors and tensors are mathematical constructions with a strong physical background, so many elementary techniques show this heritage. The easiest techniques are hedgehogs. At one or more positions, usually where the discrete data is given, one shows a single graphical primitive representing the vector or tensor. For vectors, arrows are the common choice as in the left image of Figure 27.4.1. For symmetric tensors, tripods of the orthogonal eigenvectors give one typical way as can be depicted from the right image in Figure 27.4.1. Since the length of the three line segments can only indicate the absolute value of the eigenvectors, color is used to indicate the sign of the eigenvalues. Here, red indicates positive eigenvalues and green stands for negative eigenvalues. In case of only positive eigenvalues, ellipsoids are another typical choice. Glyphs give only information about

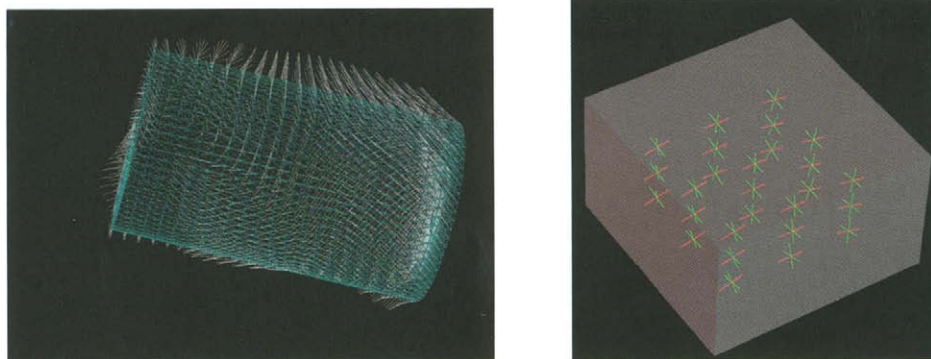


Figure 27.5. Left Image: Arrows or short line segments are a vary old technique to show vector data at a single point. Right Image: Tripods consisting of the three eigenvectors can be used to show tensor data at one position.

the data at a point. It is possible to include also gradient information. The local flow probe [13] gives a nice example by including the local streamline with curvature, torsion, divergence, and curl in the presentation of the vector data.

27.4.2. Line-based methods

This second kind of methods has an one-dimensional spatial domain. For vector fields, they are all based on integral curves. Let

$$\begin{aligned} v : R^d \supset D &\rightarrow R^d \quad (d = 2 \text{ or } 3) \\ x &\mapsto v(x) \end{aligned} \quad (27.1)$$

be a d -dimensional, steady vector field over a domain D as in section 3. An integral curve of v through $a \in D$,

$$c : R \supset (\alpha, \beta) \rightarrow D, \quad t \mapsto c(t), \quad \alpha < 0 < \beta \quad (27.2)$$

fulfils the two conditions

$$c(0) = a \quad (27.3)$$

$$\frac{\partial c}{\partial t}(t) = v(c(t)) \quad (27.4)$$

The calculation is typically based on a numerical method for ordinary initial value problems, like Euler method, Midpoint-Rule, Runge-Kutta-Fehlberg methods or Predictor-Corrector methods [46], sometimes with adaptive stepsize control to maintain given error

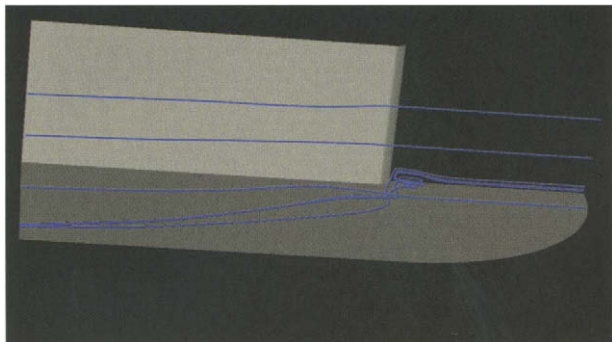


Figure 27.6. Integral curves are a simple, but effective line-type technique to show vector data.

bounds. Therefore, one calculates points on the curve which can be connected by line segments or small tube elements which leads to pictures like Figure 27.4.2. If one wants to include neighboring information in three-dimensional vector fields, one can compute a second close streamline to form a ribbon [6]. This can be improved by calculating the curvature of the streamline [61], since the two streamlines of the ribbon may diverge. The stream polygon [50] provides another concept for including information about a three-dimensional vector field along an integral curve. By rotating and shearing the polygon, one can visualize rotation and rate of deformation, for example in fluid flows. Very nice visualizations of integral curves can be generated by using an illumination model for curves. This has been demonstrated by Zöckler, Stalling and Hege [64]. Nielson and Jung [38] give an overview over efficient techniques for the computation of integral curves over tetrahedral meshes that allow an exact solution of the problem if linear interpolation is used. The formulation in barycentric coordinates simplifies the calculations in this case.

In the tensor field case, Dickinson [16] introduced the concept of a tensor line which corresponds to the integral curves of vector fields. Let

$$T : R^d \supset D \rightarrow R^d \times R^d \quad x \mapsto \begin{pmatrix} t_{11} & \dots & t_{1d} \\ \vdots & \ddots & \vdots \\ t_{1d} & \dots & t_{dd} \end{pmatrix} \quad (27.5)$$

be a symmetric tensor field over the domain D . At each position $x \in D$, there are in general d real eigenvalues $\lambda_1 > \dots > \lambda_d$ and d orthogonal eigenvectors e_1, \dots, e_d with $T(x)e_i = \lambda_i e_i$. This defines d eigenvector fields

$$E_i : D \rightarrow R^d, \quad (27.6) \\ x \mapsto e_i.$$

A tensor line of the i -th eigenvector field of T through $b \in D$ is a curve $d : (\gamma, \delta) \rightarrow D$,

$t \mapsto d(t)$, $\gamma < 0 < \delta$ with the following two conditions:

$$d(0) = b \tag{27.7}$$

$$\frac{\partial d}{\partial t}(t) \wedge e_i(d(t)) = 0, \tag{27.8}$$

i.e. the tangent of d is at every position parallel to the i -th eigenvector. Since this definition is far less known than than the integral curves above, Figure 27.4.2 illustrates this idea in two dimensions. Besides steady data, there is also unsteady data. Usually,

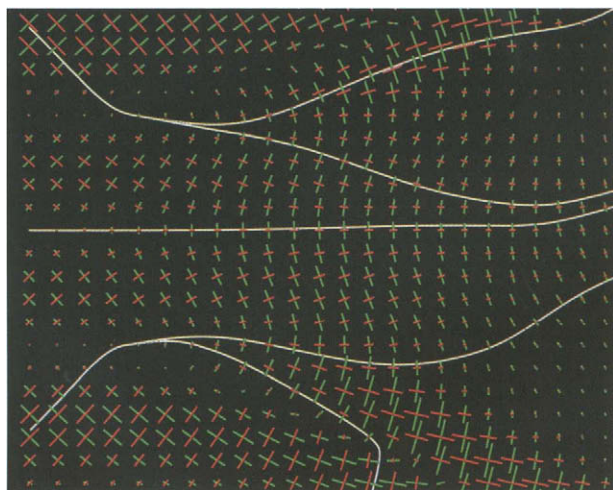


Figure 27.7. Tensor lines indicate the direction of maximum, medium or minimum forces, stress or strain. In this example, the tensor lines follow the maximum rate of strain.

one will be given data at discrete timesteps with fixed or moving positions as mentioned in section 27.3. From experimental flow visualization techniques, one has copied four different line-type methods [28,29]. Let

$$v : D \times [\tau_0, \tau_n] \rightarrow R^d, \quad (x, \tau) \mapsto v(x, \tau) \tag{27.9}$$

be an unsteady vector field. A pathline of v through $a \in D$ at $\bar{\tau} \in [\tau_0, \tau_n]$ is a curve $p : [\tau_0, \tau_n] \rightarrow R^d$, $\tau \mapsto p(\tau)$ with the conditions

$$p(\bar{\tau}) = a \tag{27.10}$$

$$\frac{\partial p}{\partial \tau}(\tau) = v(p(\tau), \tau). \tag{27.11}$$

A pathline models the movement of a particle in the field (for example a positively charged particle in an electrical field). For velocity fields in fluid flows, one uses also timelines and streaklines.

A timeline is created by starting a line of colored dye particles in a windtunnel at one moment and tracing it by photographs. Mathematically, a timeline at time $\hat{\tau}$ of v defined by a curve $c : R \supset I \rightarrow R^d$ at timestep τ_0 is a curve

$$t : I \rightarrow R^d, \quad \sigma \mapsto p_{c(\sigma), t_0}(\hat{\tau}) \quad (27.12)$$

where

$$p_{c(\sigma), t_0} : [\tau_0, \tau_n] \rightarrow R^d \quad (27.13)$$

is a pathline of v through $c(\sigma)$ at times τ_0 .

A streakline arises by inducing continuously dye into a fluid flow from a fixed position. More formal, a streakline of v through a point $a \in D$ is a curve

$$s : [\tau_0, \tau_n] \rightarrow R^d, \quad \tau \mapsto p_{a, \tau}(\tau_n) \quad (27.14)$$

where

$$p_{a, \tau} : [\tau_0, \tau_n] \rightarrow R^d \quad (27.15)$$

is a pathline of v through a at time τ . Besides these two special cases for fluid dynamics, one uses also instantaneous integral curves in unsteady vector field visualization, i.e. integral curves of the vector fields

$$v_\tau : D \rightarrow R^d, \quad x \mapsto v(x, \tau), \quad (27.16)$$

especially $\tau = \tau_0, \dots, \tau_n$. Interestingly, there has not been much research on the line-type time-dependent methods for tensor fields.

27.4.3. Surface and volume-based methods

Besides point-based and line-based methods, several people have also tried surface and volume-based approaches. Computational Fluid Dynamics (CFD) is again a dominant application area. From there, the idea of a stream surface has been taken by Hultquist [26]. Let $v : R^3 \rightarrow R^3$, $x \mapsto v(x)$ be a steady vector field. A stream surface S of v defined by a curve $c : [\alpha, \beta] \rightarrow R^3$, $\sigma \mapsto c(\sigma)$ is a map

$$S : [\alpha, \beta] \times (\gamma, \delta) \rightarrow R^3, \quad \gamma < 0 < \delta \quad (27.17)$$

$$(\sigma, \tau) \mapsto S(\sigma, \tau) \quad (27.18)$$

with

$$S(\sigma, 0) = c(\sigma) \quad (27.19)$$

$$\frac{\partial S}{\partial \tau}(\sigma, \tau) = v(S(\sigma, \tau)). \quad (27.20)$$

In other words, a stream surface consists of all integral curves of v defined by the points on a continuous curve. Hultquist's algorithm traces several integral curves from points on the defining curve and interpolates between them by triangles. If two neighboring integral curves become closer than a threshold, he stops one of the traces. If two neighboring integral curves diverge, he introduces an additional trace to keep the triangles (and the

error) small. Van Wijk [60] has published a different algorithm based on local implicit surface pieces. There is less experience with surface techniques for tensor fields so far.

There are only a few volume-based visualization techniques for vector fields and tensor fields. We mention tridimensional line integral convolution (3D LIC) in the section about texture-based methods. Some people have applied successful scalar data methods like volume rendering on vector fields. An early example based on filters and textures has been presented by Crawfis and Max [9]. It uses textures to create the impression of a large number of moving short line segments (arrows, splats) in the volume. It can also be seen as a texture-based technique. Several people have looked at using volume graphics approaches to vector fields, usually using derived scalars like components, magnitude or rotation magnitude, see Crawfis et al. for a nice example [10]. There are also successful attempts to use a large number of particles moving in a vector field creating virtual smoke [33,24].

27.5. ATTRIBUTE MAPPINGS

Besides the mapping to geometric primitives, one can also use attributes of computer graphics primitives to visualize the data. Color is probably the oldest attribute, but it is not easy to use for vector or tensor data, if one wants to show the whole data and not only a derived scalar like magnitude or component. Other attributes offer better possibilities. Texture is a well-known standard computer graphics method to show fine details without geometric modeling of the detail information. Essentially, it is the mapping of a picture on geometry, typically polygons. It was first developed by Catmull [8]. The maximal information content of textures with respect to the screen resolution make them attractive for visualization purposes. Spot noise by van Wijk [59] is an early example. The basic underlying idea is that a large number of particles leads to textures because the individual particles can not be distinguished. For the visualization of vector fields, small, randomly placed ellipsoids are used with eccentricity proportional to the magnitude and the longer axis aligned with the vector direction. Spot noise influenced the most popular texture based visualization algorithm, line integral convolution (LIC), invented by Cabral and Leedom [7]. LIC starts with a white noise texture of grey values and smears the grey values along the integral curves. This enhances the color correspondence in the direction of the vector field and leads to nice vector field visualizations, see Figure 27.5. The smearing effect is created by convoluting the pixel values along the integral curves. Stalling and Hege [53] have given a very fast and efficient implementation which is often used in applications. A comparison of LIC and sport noise has been published by de Leeuw and van Liere [11]. Due to its success for steady fields on rectilinear grids, many people have tried to extend LIC to other data set types. Forssell and Cohen discuss curvilinear surfaces [18], Battke et al. arbitrary surfaces [5]. Several people have worked on animation of textures and dye (color) advection to apply LIC to unsteady vector fields. Shen, Johnson, Ma [51] use color to animate the texture. The convolution filter can also be used, as is demonstrated by the unsteady flow line integral convolution (UFLIC) algorithm by Shen and Kao [52]. The application of LIC to three-dimensional vector fields has been tried by several authors, but due to the complexity of three-dimensional textures, this is a difficult endeavour. One needs a lot of hints for the eyes and research into perception

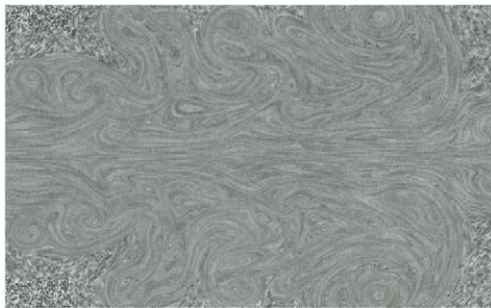


Figure 27.8. Line integral convolution allows the description of fine details. In some of the corners where no movement is present in the fluid, one can still see the original white noise.

is necessary to find good tricks to do this. Interrante et al. have done research in this direction [27].

27.6. STRUCTURE AND FEATURE BASED MAPPINGS

Visualization creates meaningful images to the user to understand his data. Since the whole data is in digital form, it makes sense to use the computer to look for structure elements and specific features, once a precise definition is given. For vector and tensor fields, topology provides a clearly defined mathematical framework with high application relevance. Topology extraction and visualization is therefore a good way to concentrate information about the data or enhance the presentation to the scientist or engineer working with the data. We review the history and state of the art in the first two subsections. For many interesting aspects of data sets, there are mathematical formulations, often many with partly contradicting content. These formulations can be taken as basis for feature extraction algorithms. The results of these algorithms are visualized. Examples are given in the last subsection.

27.6.1. Vector field topology

Integral curves of steady, Lipschitz-continuous vector fields as introduced in section 27.4.2 exist through every point and are unique. This follows from the well-known Picard-Lindelöf theorem of ordinary differential equations. Therefore, the whole domain is filled with curves, so one can analyze the topological behavior of all these curves. The general idea of topological methods is to group all curves together that can be continuously deformed into each other. Such curves all called homotop in topology. The first step is to identify all potential start and end points (sets) of integral curves. More formally, let $c_a : R \rightarrow D$ be an integral curve through $a \in D$ of a vector field $v : R^d \supset D \rightarrow R^d$. The α -limit set of c_a is the set

$$A = \{p \in \bar{D} \mid \exists (\tau_n)_{n=0}^{\infty} \subset R, \tau_n \rightarrow -\infty, \lim_{n \rightarrow \infty} c_a(\tau_n) \rightarrow p\}, \quad (27.21)$$

and the ω -limit set of c_a is

$$\Omega = \{p \in \bar{D} \mid \exists (\tau_n)_{n=0}^\infty \subset R, \tau_n \rightarrow \infty, \lim_{n \rightarrow \infty} c_a(\tau_n) \rightarrow p\}. \tag{27.22}$$

(\bar{D} denotes the closure of the domain D .) The most popular case for A or Ω is a critical point $\{p\}$ with $v(p) = 0$. Points of the domain boundary ∂D can also act as α - or ω -limit set. Further possibilities are closed integral curves, tori and so-called strange attractors/repellers that do not fit into the other categories. These results belong to the theory of dynamical systems. An excellent comic-style introduction is given by Abraham and Shaw [1], more formal texts are also available [25,20].

If all potential α - and ω -limit sets are found, the boundaries of the regions consisting of homotop curves are calculated. These curves in two dimensions resp. surfaces in three dimensions are called separatrices. Details on their computation can be found in the articles cited below. For two-dimensional vector fields, Helman and Hesselink did pioneer work in this area [22,23]. They treated exclusively critical points. Nielson et al. give a nice description including the use of barycentric coordinates [40]. A clear treatment of the boundary can be found in an article by Scheuermann et al. [48]. Figure 27.6.1 illustrates the difference between the two approaches. The detection and analysis of closed integral curves has been finished by Wischgoll and Scheuermann just recently [62]. An example is presented in Figure 27.6.1. Since strange attractors and repellers do not

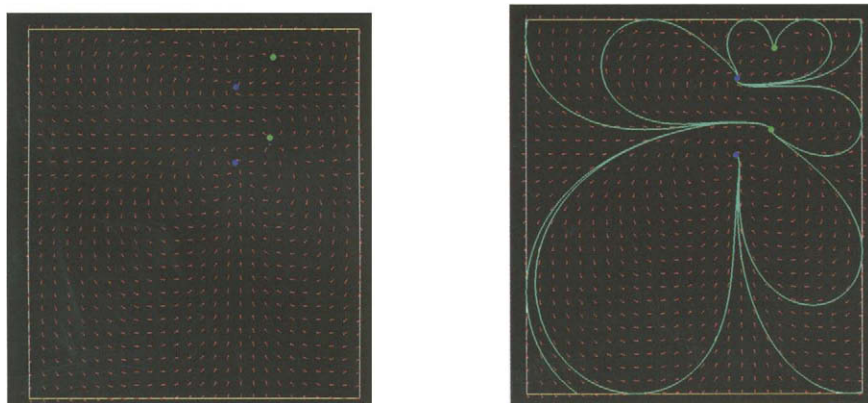


Figure 27.9. The inclusion of the boundary is necessary in this academic example to find the classes of homotop integral curves.

appear in generic planar vector fields as a result of a famous theorem of Peixoto [44], the planar topology visualization is now complete for vector fields. In three dimensions, one has dealt with critical points alone without defining the separating surfaces that replace the separating curves from the two dimensional case [19]. Another aspect of topology visualization is the simplification of the topological structure. Especially in turbulent

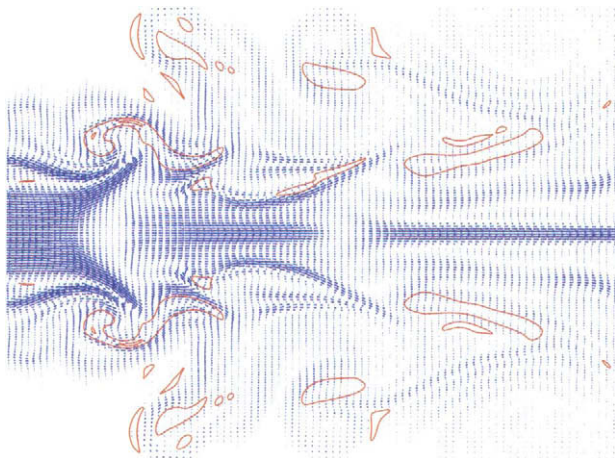


Figure 27.10. There may be closed integral curves in a data set. They play a role similar to critical points in vector field topology. As can be seen here, closed integral curves may be inside each other.

velocity vector fields, the number of critical points and separatrices is so high that a depiction of the field is difficult. Nielson et al. [39] used wavelet transforms of the vector field to reduce its structural complexity. This work has been extended to a nice treatment of wavelets over curvilinear grids [41] which opened the way for the use of multiresolution visualization techniques for this important class of data sets. De Leeuw and van Liere [12] simplified the topological graph consisting of critical points and separating points at the boundary as vertices and separatrices as edges using topology based rules without changing the underlying vector field. Tricoche, Scheuermann and Hagen [55] simplified the vector field topology by merging critical points into higher order critical points, see Figure 27.6.1. Lodha et al. [32] maintain the topology as long as possible while reducing the information content of the remaining vector field. Vector Field Topology can also be used to compare different flows. One needs a measure for the similarity of different topologies. Lavin, Batra and Hesselink have used the earth mover's distance to compare the critical points in the field [30,4].

27.6.2. Tensor field topology

The success of the vector field topology visualization led Delmarcelle and Hesselink [14] to analyze the structure of symmetric second-order tensor fields. The basic idea is exactly the same as for vector fields, one looks for homotop integral curves — in this case tensor lines. The role of the critical points is now filled by the so-called degenerate points. At these points, the tensor field has two equal eigenvalues, so there is no unique tangent given by the eigenvectors and one can not define a tensor line through these points. Separatrices can be defined again, but due to the difference between tangents and vectors (vectors

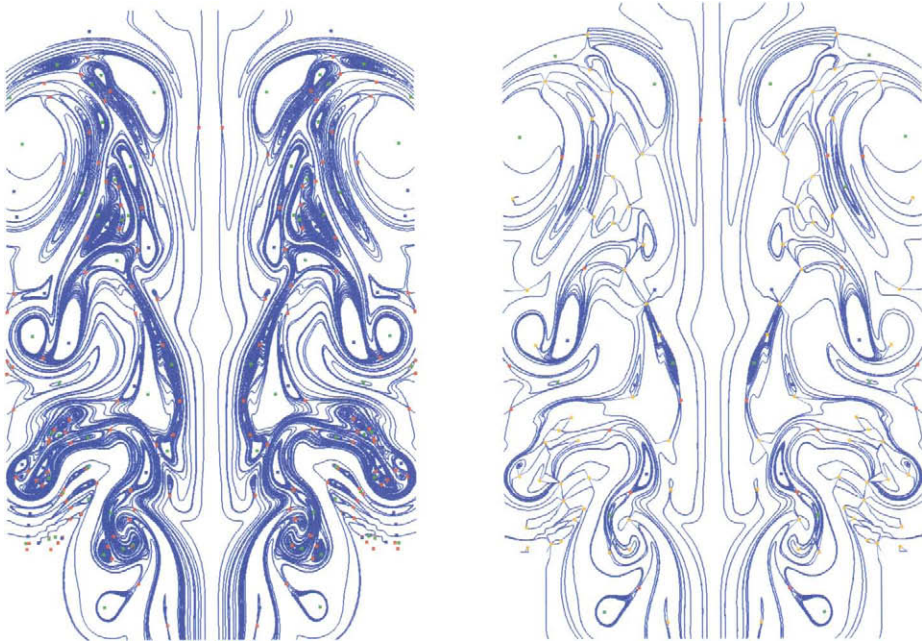


Figure 27.11. The topology of turbulent vector fields can be rather complicated. It is possible to scale the topology to allow a simpler depiction of large scale structures.

define an orientation), one gets different patterns, see Figure 27.6.2.

Lavin, Batra and Hesselink [31] extended some of these ideas to three dimensions, but as in the vector case, there is still no complete algorithm for tridimensional topology visualization of symmetric tensor fields. Tricoche, Scheuermann and Hagen [56] show the simplification of tensor field structure based on the fusion of degenerate points of higher order.

27.6.3. Feature detection algorithms

“A feature is defined as anything contained in a data set that might be of interest for interpretation” [47, p. 15]. This vague definition comes from the broad variation of application areas and the different measurement and simulation methods. The main point is that a feature must be local with respect to the data set domain, so that algorithms always try to find the domain feature. An excellent overview in the area of fluid flows is given by Roth in his recent Ph.D. thesis [47]. Delmarcelle and Hesselink [15] distinguish features by the dimension of the feature domain. This leads to point, line, surface and volume features. The critical and degenerate points from subsections 27.6.1 and 27.6.2 are typical examples for point features. Again, fluid flow applications dominate this research area up to now. Vortex cores [42,2,54,36] are typical line features describing the

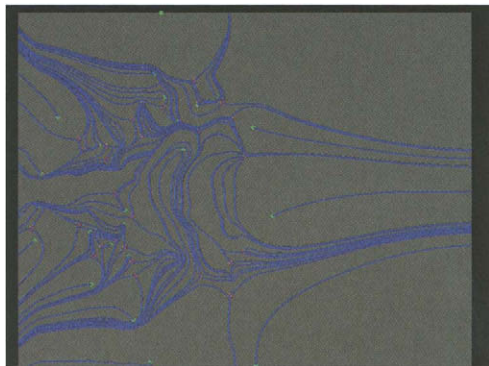


Figure 27.12. The topology of tensor fields shows different patterns than vector field topology because the eigenvectors do not define an orientation. The image shows a tensor field topology of the rate of deformation tensor in a swirling jet simulation.

“center” of a vortex in three dimensions. Shock fronts [43] are surface features where a physical property like pressure is not continuous. It is often found by looking for high absolute values of the gradient. Volume features are often defined as all points with all points above or below a threshold of a scalar function. This scalar function may be derived from a vector field like the absolute value of the vorticity vector field [63]. A general, application-independent approach based on these ideas to feature extraction and visualization has been shown by Post and Silver, together with several coworkers. It is mainly directed towards scalar fields, but it marks a way for further research for feature extraction in vector and tensor fields [57,58].

REFERENCES

1. R.H. Abraham and C.D. Shaw. *Dynamics, the Geometry of Behavior I-IV*. Aerial Press, Santa Cruz, CA, 1982, 1983, 1985, 1988.
2. D. Banks and B. Singer. Vortex tubes in turbulent flows: Identification, representation, reconstruction. In *IEEE Visualization '94*, pages 132–139, Washington, DC, 1994.
3. K.-J. Bathe. *Finite Element Procedures*. Prentice Hall, Upper Saddle River, NJ, 1996.
4. R. Batra and L. Hesselink. Feature comparisons of 3D vector fields using earth mover's distance. In *IEEE Visualization '99*, pages 105–114, San Francisco, CA, 1999.
5. H. Battke, D. Stalling, and H.-C. Hege. Fast line integral convolution for arbitrary surfaces in 3D. In *Visualization and Mathematics*, pages 181–195, Springer, Heidelberg, 1997.
6. P.G. Bunning. Numerical algorithms in cfd post-processing. Technical Report Lec-

- ture Series 1989-07, van Karman Institute for Fluid Dynamics, 1989.
7. B. Cabral and L.C. Leedom. Imaging vector fields using line integral convolution. *Computer Graphics*, SIGGRAPH '93, 27(4):263–270, 1993.
 8. E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Computer Science Department, University of Utah, Salt Lake City, UT, December 1974.
 9. R. Crawfis and N. Max. Direct volume visualization of three dimensional vector fields. In *Volume Visualization Workshop 1992*, pages 55–60, New York, ACM Siggraph, 1992.
 10. R. Crawfis, N. Max, and B. Becker. Vector field visualization. *IEEE Computer Graphics and Applications*, 14(5):50–56, 1994.
 11. W.C. de Leeuw and R. van Liere. Comparing LIC and spot noise. In *IEEE Visualization '98*, pages 359–365, IEEE Computer Society Press, Los Alamitos, CA, 1998.
 12. W.C. de Leeuw and R. van Liere. Visualization of global flow structures using multiple levels of topology. In *EUROGRAPHICS — IEEE TCVG Symposium on Visualization*, pages 45–52, 1999.
 13. W.C. de Leeuw and J.J. van Wijk. A probe for local flow field visualization. In *IEEE Visualization '93*, pages 39–45, IEEE Computer Society Press, Los Alamitos, CA, 1993.
 14. T. Delmarcelle and L. Hesselink. Visualizing second-order tensor fields with hyperstreamlines. *IEEE Computer Graphics & Applications*, 13(4):25–33, 1993.
 15. T. Delmarcelle and L. Hesselink. A unified framework for flow visualization. In R.S. Gallagher, editor, *Computer Visualization: Graphics Techniques for Scientific and Engineering Analysis*, CRC Press, 1994.
 16. R.R. Dickenson. A unified approach to the design of visualization software for the analysis of field problems. In *SPIE Proceedings*, Vol. 1083, pages 173–180, 1989.
 17. J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics — Principles and Practice*. Addison-Wesley, Reading, MA, 2nd edition, 1996.
 18. L.K. Forssell and S.D. Cohen. Using line integral convolution for flow visualization: curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transaction on Visualization and Graphics*, 1(2):133–141, 1995.
 19. A. Globus, C. Levit, and T. Lasinski. A tool for visualizing the topology of three-dimensional vector fields. In *IEEE Visualization '91*, pages 33–40, San Diego, 1991.
 20. J Guckenheimer and P. Holmes. *Dynamical Systems and Bifurcation of Vector Fields*. Springer-Verlag, New York, 1983.
 21. B. Hamann and R.J. Moorhead II. A survey of grid generation methodologies and scientific visualization efforts. In *Scientific Visualization*, pages 59–101. IEEE Computer Society, Los Alamitos, CA, 1997.
 22. J.L. Helman and L. Hesselink. Surface representations of two- and three-dimensional fluid flow topology. In G.M. Nielson and B. Shriver, editors, *Visualization in Scientific Computing*, pages 6–13, Los Alamitos, CA, 1990.
 23. J.L. Helman and L. Hesselink. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications*, 11(3):36–46, May 1991.
 24. A. Hin and F. Post. Visualization of turbulent flow with particles. In *IEEE Vi-*

- sualization '93*, pages 46 – 52, IEEE Computer Society Press, Los Alamitos, CA, 1993.
25. M.W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems and Linear Algebra*. Academic Press, New York, 1974.
 26. J.P.M. Hultquist. Constructing stream surfaces in steady 3D vector fields. In A.E. Kaufman and G.M. Nielson, editors, *IEEE Visualization '92*, pages 171–178, Boston, MA, 1992.
 27. V. Interrante and C. Grosch. Visualizing 3D flow. *IEEE Computer Graphics and Applications*, 18(4):49 – 53, 1998.
 28. D.A. Lane. Visualization of time-dependent flow fields. In *IEEE Visualization '93*, pages 32–38, IEEE Computer Society Press, Los Alamitos, CA, 1993.
 29. D.A. Lane. Visualizing time-varying phenomena in numerical simulations of unsteady flows. Technical Report NAS-96-001, NASA Ames Research Center, 1996.
 30. Y. Lavin, R. Batra, and L. Hesselink. Feature comparisons of vector fields using earth mover's distance. In *IEEE Visualization '98*, pages 103–109, Research Triangle Park, NC, 1998.
 31. Y. Lavin, Y. Levy, and L. Hesselink. Singularities in nonuniform tensor fields. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97*, pages 59–66, IEEE Computer Society, Los Alamitos, CA, 1997.
 32. S.K. Lodha, J.C. Renteria, and K.M. Roskin. Topology preserving compression of 2D vector fields. *IEEE Visualization 2000*, pages 343–350, Salt Lake City, UT, 2000.
 33. K.-L. Ma and P.J. Smith. Virtual smoke: an interactive 3D flow visualization technique. In A.E. Kaufman and G.M. Nielson, editors, *IEEE Visualization '92*, pages 46–53, IEEE Computer Society Press, Los Alamitos, CA, 1992.
 34. B.H. McCormick and T.A. DeFanti. Visualization in scientific computing. *Computer Graphics*, 21(6), 1987.
 35. W. Merzkirch. *Flow Visualisation*, 2nd Edition. Academic Press, New York, 1987.
 36. H. Miura and S. Kida. Identification of central lines of swirling motion in turbulence. In *Proceedings of Intl. Conference on Plasma Physics*, pages 866–869, Nagoya, Japan, 1996.
 37. G.A. Mohr. *Finite Elements for Solids, Fluids, and Optimization*. Oxford University Press, Oxford, 1992.
 38. G.M. Nielson and I.-H. Jung. Tools for computing tangent curves for linearly varying vector fields over tetrahedral domains. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):360–372, 1999.
 39. G.M. Nielson, I.-H. Jung, and J. Sung. Haar wavelets over triangular domains with applications to multiresolution models for flow over a sphere. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97*, pages 143–149, IEEE Computer Society, Los Alamitos, CA, 1997.
 40. G.M. Nielson, I.-H. Jung, J. Sung, N. Srinivasan, and J.-B. Yoon. *Tools for Computing Tangent Curves and Topological Graphs*, pages 527–562. IEEE Computer Society Press, Los Alamitos, CA, 1997.
 41. G.M. Nielson and J. Sung. Wavelets over curvilinear grids. *IEEE Visualization '98*, pages 313–317, IEEE Computer Society Press, Los Alamitos, 1998.
 42. H.-G. Pagendarm, B. Henne, and M. Rütten. Detecting vortical phenomena in vector

- data by medium-scale correlation. *IEEE Visualization '99*, pages 409–412, IEEE Computer Society Press, Los Alamitos, 1999.
43. H.-G. Pagendarm and B. Seitz. An algorithm for detection and visualization of discontinuities. In P. Palamidese, editor, *Scientific Visualization — Advanced Software Techniques*, pages 161–177. Ellis Horwood Ltd., 1993.
 44. M. Peixoto. Structural stability on two-dimensional manifolds. *Topology*, 2:101–121, 1961.
 45. F. Post and T. van Walsum. Fluid flow visualization. In H. Hagen, H. Müller, and G.M. Nielson, editors, *Focus on Scientific Visualization*, pages 1–40. Springer, Berlin, 1993.
 46. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C*, 2nd Edition. Cambridge University Press, Cambridge, UK, 1992.
 47. M. Roth. *Automatic Extraction of Vortex Core Lines and Other Line-Type Features for Scientific Visualization*. PhD thesis, ETH Zürich, 2000.
 48. G. Scheuermann, B. Hamann, K.I. Joy, and W. Kollmann. Visualizing local vector field topology. *Journal of Electronic Imaging*, 9(4):356 – 367, 2000.
 49. W. Schroeder, K.W. Martin, and B. Lorensen. *The Visualization Toolkit*, 2nd Edition. Prentice-Hall, Upper Saddle River, NJ, 1998.
 50. W. Schroeder, C. Volpe, and W. Lorensen. The stream polygon: a technique for 3D vector field visualization. *IEEE Visualization 1991*, pages 126–132, Los Alamitos, CA, 1991.
 51. H.-W. Shen, C.R. Johnson, and K.-L. Ma. Visualizing vector fields using line integral convolution and dye advection. *IEEE Symposium on Volume Visualization*, pages 63–70. ACM, New York, NY, 1996.
 52. H.-W. Shen and D.L. Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):98–108, 1998.
 53. D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. *Computer Graphics*, SIGGRAPH '95, 29(4):249–256, 1995.
 54. D. Sujudi and R. Haimes. Identification of swirling flow in 3D vector fields. In *12th AIAA CFD Conference*, AIAA Paper 95–1715, San Diego, CA, 1995.
 55. X. Tricoche, G. Scheuermann, and H. Hagen. A topology simplification method for 2D vector fields. *IEEE Visualization 2000*, pages 359–366, Los Alamitos, CA, 2000.
 56. X. Tricoche, G. Scheuermann, and H. Hagen. Vector and tensor field topology simplification on irregular grids. To appear, *Joint Eurographics and IEEE TCVG Symposium on Visualization 2001*, 2001.
 57. T. van Walsum and F. Post. Selective visualization of vector fields. *Computer Graphics Forum*, EUROGRAPHICS '94, 13(3):339–347, 1994.
 58. T. van Walsum, F.H. Post, D. Silver, and F.J. Post. Feature extraction and iconic visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):111–119, 1996.
 59. J.J. van Wijk. Spot noise. *Computer Graphics*, SIGGRAPH '91, 25(4):309–318, 1991.
 60. J.J. van Wijk. Implicit stream surfaces. *IEEE Visualization '93*, pages 254–252, 1993.

61. G. Volpe. Streamlines and streamribbons in aerodynamics. Technical Report AIAA-89-0140, AIAA, 1989.
62. T. Wischgoll and G. Scheuermann. Detection and visualization of closed streamlines in planar flows. *IEEE Transactions on Visualization and Computer Graphics*, 2001.
63. N. Zabusky, O. Boratav, R. Pelz, M. Gao, D. Silver, and S. Cooper. Emergence of current patterns of vortex stretching during reconnection: a scattering paradigm. *Physical Review Letters*, 67(18):2469–2472, 1991.
64. M. Zöckler, D. Stalling, and H. Hege. Interactive visualization of 3D-vector fields using illuminated streamlines. *IEEE Visualization '96*, pages 107–113. IEEE Computer Society, Los Alamitos, CA, 1996.

Chapter 28

Splines over Triangulations

Frank Zeilfelder and Hans-Peter Seidel

28.1. INTRODUCTION

In the past 35 years, many research papers have been written on bivariate, respectively multivariate splines. This work has been motivated in many cases by the aim to develop powerful tools for fields of application, such as scattered data fitting, the construction and reconstruction of surfaces and the numerical solution of boundary-value problems.

A natural generalization of the classical *univariate spline spaces* (cf. [16,87,112]) which has been widely considered in the literature is defined w.r.t. triangulations (i.e. a finite set of closed triangles in \mathbb{R}^2 such that the intersection of any two triangles is empty, a common edge or a common vertex). For given integers $r, q, 0 \leq r < q$, the space of *bivariate splines* of degree q and the smoothness r with respect to Δ is defined by

$$S_q^r(\Delta) = \{s \in C^r(\Omega) : s|_T \in \Pi_q, T \in \Delta\},$$

where

$$\Pi_q = \text{span}\{x^i y^j : i, j \geq 0, i + j \leq q\}$$

is the space of *bivariate polynomials* of total degree q . Many research papers on bivariate splines deal with certain subspaces of $S_q^r(\Delta)$, the so-called super splines. Suppose that $\rho_i, i = 1, \dots, V$, are integers satisfying $r \leq \rho_i < q, i = 1, \dots, V$, and let $\theta = (\rho_1, \dots, \rho_V)$, where V is the number of vertices of Δ . The space of *bivariate super splines* with respect to Δ is defined by

$$S_q^{r,\theta}(\Delta) = \{s \in S_q^r(\Delta) : s \in C^{\rho_i}(v_i), i = 1, \dots, V\}.$$

In contrast to the case of univariate splines, even standard problems such as determining the dimension and the approximation order of bivariate splines and constructing explicit interpolation schemes for these spaces are difficult to solve. In particular, these spaces

are very complex when the degree q approaches the smoothness r , which is one of the fundamental phenomena in bivariate spline theory .

The aim of this survey is to summarize results on splines over triangulations. We organize the paper as follows.

In Section 28.2, we survey results on *Bernstein-Bézier techniques*, which are important for CAGD applications. In the context of bivariate splines these techniques provide a useful tool for analysing the structure of these spaces. Section 28.3 deals with the dimension of bivariate spline spaces. In contrast to the univariate case, the dimension of these spaces is not known, in general: the most general results are lower and upper bounds. In Section 28.4 and 28.5, we discuss interpolation by bivariate spline spaces \mathcal{S} , where \mathcal{S} can be the space $S_q^r(\Delta)$ as well as a super spline space $S_q^{r,\theta}(\Delta)$. A set $\{z_1, \dots, z_d\}$ in Ω , where $d = \dim \mathcal{S}$, is called a *Lagrange interpolation set* for \mathcal{S} if for each function $f \in C(\Omega)$, a unique spline $s \in \mathcal{S}$ exists such that $s(z_i) = f(z_i)$, $i = 1, \dots, d$. If also partial derivatives of a sufficiently differentiable function f are involved and the total number of Hermite conditions is d , then we speak of a *Hermite interpolation set* for \mathcal{S} . In Section 28.4, we discuss classical *finite elements* and their modern generalizations, the so-called *macro element methods* , which lead to Hermite interpolation by super splines. Hermite- and Lagrange interpolation methods for bivariate spline spaces $S_q^r(\Delta)$ are summarized in Section 28.5. Most of these methods have been developed recently, in particular *local Lagrange interpolation* methods for C^1 spline spaces . A different method for analysing splines over triangulations is based on so-called *multivariate B-splines* . This approach is described in Section 28.6.

28.2. BERNSTEIN-BÉZIER TECHNIQUES

In this section, we describe Bernstein-Bézier techniques. These methods are important for applications in CAGD (cf. [13,55,66]). In the context of bivariate splines, these techniques appear to play a fundamental role for mainly two reasons. First, since the Bernstein-Bézier representation of the polynomial pieces is used in many research papers to analyse the structure of the spline space. And second, these techniques allow an efficient and stable computation of bivariate splines.

Let $T = [t_0, t_1, t_2]$ be a triangle with vertices t_0, t_1, t_2 in \mathbb{R}^2 . Given a point $u \in \mathbb{R}^2$ there exist unique scalars $\lambda_0(u), \lambda_1(u), \lambda_2(u)$ such that

$$u = \sum_{i=0}^2 \lambda_i(u)t_i \text{ and } \sum_{i=0}^2 \lambda_i(u) = 1.$$

The coefficients $\lambda = (\lambda_0, \lambda_1, \lambda_2)$ are called *barycentric coordinates* . Given T , the *Bernstein polynomials* $B_\alpha^{T,q} \in \Pi_q$ of degree q w.r.t. T are defined as

$$B_\alpha^{T,q}(u) = \frac{q!}{\alpha_0! \alpha_1! \alpha_2!} \lambda_0^{\alpha_0}(u) \lambda_1^{\alpha_1}(u) \lambda_2^{\alpha_2}(u), \quad |\alpha| = \alpha_0 + \alpha_1 + \alpha_2 = q.$$

The unique representation

$$p(u) = \sum_{|\alpha|=q} B_\alpha^{T,q}(u) b_\alpha, \quad u \in \mathbb{R}^2, \tag{28.1}$$

of a polynomial $p \in \Pi_q$ is then called the *Bernstein-Bézier representation* of p and $b_\alpha \in \mathbb{R}$ are called the *Bernstein-Bézier coefficients* of p . The value $p(\mathbf{u})$ can be computed recursively by applying the so-called *de Casteljau algorithm*, which reads as follows:

$$p(\mathbf{u}) = \sum_{|\alpha|=q-l} B_\alpha^{T,q-l}(\mathbf{u}) b_\alpha^l(\mathbf{u}), \quad l = 1, \dots, q,$$

where $b_\alpha^0(\mathbf{u}) = b_\alpha$, $|\alpha| = q$, and

$$b_\alpha^l(\mathbf{u}) = \sum_{i=0}^2 \lambda_i(\mathbf{u}) b_{\alpha+e^i}^{l-1}(\mathbf{u}), \quad l = 1, \dots, q, \quad |\alpha| = q - l. \tag{28.2}$$

The Bernstein-Bézier representation of the polynomial pieces of a spline from $S_q^r(\Delta)$ can be used to translate C^r smoothness (across common edges) into useful conditions (cf. [17,24,53]). Let $T_0 = [\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2]$ and $T_1 = [\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$ be triangles with common edge $[\mathbf{t}_1, \mathbf{t}_2]$ and s be a function that is given in the piecewise polynomial Bernstein-Bézier representation:

$$s|_{T_i}(\mathbf{u}) = \sum_{|\alpha|=q} B_\alpha^{T_i,q}(\mathbf{u}) b_{i,\alpha}, \quad \mathbf{u} \in T_i, \quad i = 0, 1,$$

where $b_{i,\alpha} \in \mathbb{R}$. Then, $s \in S_q^r(\{T_0, T_1\})$ holds if and only if:

$$b_{1,\beta+\rho e^3} = \sum_{|\alpha|=\rho} B_\alpha^{T_0,\rho}(\mathbf{t}_3) b_{0,\alpha+\beta}, \quad \beta = (\beta_1, \beta_2, 0), \quad |\beta| = q - \rho, \quad \rho = 0, \dots, r. \tag{28.3}$$

These conditions can be checked by running the Casteljau algorithm for the polynomials

$$p_\rho(\mathbf{u}) = \sum_{|\alpha|=\rho} B_\alpha^{T_0,\rho}(\mathbf{u}) b_{0,\alpha+\beta}$$

at $\mathbf{u} = \mathbf{t}_3$. If d is a unit vector in direction of the edge $D = [\mathbf{t}_0, \mathbf{t}_1]$, then the partial derivative $\frac{\partial^i}{\partial d^i}$ of the Bernstein polynomials $B_\alpha^{T,q}$, $|\alpha| = q$, w.r.t. $T = [\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2]$ is given by

$$\frac{\partial^i B_\alpha^{T,q}(\mathbf{u})}{\partial d^i} = \frac{q!}{(q-i)! \|D\|^i} \sum_{|\beta|=i} B_\beta^{T,i}(\mathbf{d}) B_{\alpha-\beta}^{T,q-i}(\mathbf{u}), \quad \mathbf{u} \in \mathbb{R}^2.$$

Analogously, if d_l are unit vectors in direction of the edge $D_l = [\mathbf{t}_0, \mathbf{t}_{l+1}]$, $l = 0, 1$, then the following formula holds for $\mathbf{u} \in \mathbb{R}^2$:

$$\frac{\partial^{i+j} B_\alpha^{T,q}(\mathbf{u})}{\partial d_0^i \partial d_1^j} = \frac{q!}{(q-i-j)! \|D_0\|^i \|D_1\|^j} \sum_{|\beta|=i} \sum_{|\gamma|=j} B_\beta^{T,i}(\mathbf{d}_0) B_\gamma^{T,j}(\mathbf{d}_1) B_{\alpha-\beta-\gamma}^{T,q-i-j}(\mathbf{u}).$$

Hence, the partial derivative $\frac{\partial^{i+j}}{\partial d_0^i \partial d_1^j}$ of a polynomial p in the representation (28.1) fulfills for $\mathbf{u} \in \mathbb{R}^2$:

$$\frac{\partial^{i+j} p(\mathbf{u})}{\partial d_0^i \partial d_1^j} = \frac{q!}{(q-i-j)! \|D_0\|^i \|D_1\|^j} \sum_{|\alpha|=q} \sum_{|\beta|=i} \sum_{|\gamma|=j} B_\beta^{T,i}(\mathbf{d}_0) B_\gamma^{T,j}(\mathbf{d}_1) B_{\alpha-\beta-\gamma}^{T,q-i-j}(\mathbf{u}) b_\alpha, \tag{28.4}$$

which can also be computed by applying the de Casteljaou algorithm. Moreover, evaluating (28.4) at $\mathbf{u} = \mathbf{t}_0$ yields

$$\frac{\partial^{i+j} p(\mathbf{t}_0)}{\partial d_0^i \partial d_1^j} = \frac{q!(-1)^{i+j}}{(q-i-j)! \|D_0\|^i \|D_1\|^j} \sum_{\nu=0}^i \sum_{\mu=0}^j \binom{i}{\nu} \binom{j}{\mu} (-1)^{\nu+\mu} b_{q-\nu-\mu, \nu, \mu} .$$

This formula expresses the connection of partial derivatives at the vertices and the Bernstein-Bézier coefficients and therefore plays an important role for constructing interpolation sets for bivariate splines (cf. [17,27,93,130]). For further results on Bernstein-Bézier techniques and multivariate polynomials, we refer the reader to [12,17,18,24,37,53–55].

We finally note that polynomial surfaces were also studied with the help of a classical tool, the *polar form* [22,106,119]. Given a polynomial $F \in \Pi_q$, the corresponding multiaffin polar form is defined as the unique symmetric multiaffine map $f : (\mathbb{R}^2)^q \mapsto \mathbb{R}$ satisfying $f(\mathbf{u} \dots \mathbf{u}) = F(\mathbf{u})$. Given $F \in \Pi_q$ in its monomial form

$$F(\mathbf{u}) = \sum_{i+j \leq q} a_{i,j} \mathbf{x}^i \mathbf{y}^j,$$

where $\mathbf{u} = (\mathbf{x}, \mathbf{y})$, the corresponding polar form is given as

$$f(\mathbf{u}_1 \dots \mathbf{u}_q) = \sum_{|\alpha|=q} \frac{\alpha_0! \alpha_1! \alpha_2!}{q!} a_{\alpha_1, \alpha_2} \sum_{\substack{S_1, S_2 \subseteq \{1, \dots, q\}, \\ |S_1| = \alpha_1, |S_2| = \alpha_2, \\ S_1 \cap S_2 = \emptyset}} \prod_{j \in S_1} \mathbf{x}_j \prod_{k \in S_2} \mathbf{y}_k,$$

where $\mathbf{u}_i = (\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, \dots, q$. We remark that among other things, polar forms provide an expression for the Bézier points

$$b_\alpha = f(\mathbf{t}_0^{\alpha_0} \mathbf{t}_1^{\alpha_1} \mathbf{t}_2^{\alpha_2}) ,$$

they provide a closed form solution for the de Casteljaou recurrence (28.2)

$$b_\alpha^l(\mathbf{u}) = f(\mathbf{t}_0^{\alpha_0} \mathbf{t}_1^{\alpha_1} \mathbf{t}_2^{\alpha_2} \mathbf{u}^l) ,$$

and they make it easy to phrase the above smoothness conditions (28.3).

28.3. DIMENSION OF SPLINE SPACES

In 1973, the following question was posed [128]: what is the dimension of $S_q^r(\Delta)$? Clearly, results on the dimension of spline spaces play an important role for the whole theory. But in contrast to the case of univariate splines, investigations on the dimension of bivariate splines yield to complex mathematical problems. In principle, the only exception is the case $r = 0$, i.e. continuous spline spaces, where the dimension can be easily found (cf. [113]). For $r \geq 1$, this problem becomes highly non-trivial, and has not been yet completely solved. In this section, we summarize the results concerning this subject which were given for arbitrary triangulations and for classes of triangulations. For doing this, following [7], we set: V_I = number of interior vertices of Δ , V_B = number of boundary vertices of Δ , E_I = number of interior edges of Δ , N = number of triangles of Δ .

Given an arbitrary triangulation Δ , the most general results concerning the dimension of bivariate splines are lower and upper bounds. In 1979, the following lower bound [113] was given:

$$\dim S_q^r(\Delta) \geq \binom{q+2}{2} + \binom{q-r+1}{2} E_I - \left(\binom{q+2}{2} - \binom{r+2}{2} \right) V_I + \sum_{i=1}^{V_I} \sigma_i. \tag{28.5}$$

Here, the σ_i are integers depending on q , r , and the number of edges with different slopes attached to the i -th interior vertex of Δ .

Clearly, the dimension of a spline space is determined if an upper bound can be given that coincides with the lower bound (28.5). In order to establish such an upper bound n , it follows from a standard linear algebra argument that it suffices to construct suitable linear functionals λ_i , $i = 1, \dots, n$, with the property: if $\lambda_i(s) = 0$, $s \in S_q^r(\Delta)$, $i = 1, \dots, n$, then $s \equiv 0$. Upper bounds [114] of the following type were developed:

$$\dim S_q^r(\Delta) \leq \binom{q+2}{2} + \binom{q-r+1}{2} E_I - \left(\binom{q+2}{2} - \binom{r+2}{2} \right) V_I + \sum_{i=1}^{V_I} \bar{\sigma}_i.$$

where $\bar{\sigma}_i$ are integers depending on q , r , and the ordering of the interior vertices (see also [107]). Bounds of the above type also hold for spline spaces with respect to more general partitions than triangulations, the so-called *rectilinear partitions* [79,114], and such bounds were also given for super spline spaces and for spline spaces in more than two variables [2].

However, it is known that the upper bounds do not coincide with the lower bound (28.5), in general. In fact, there are cases where the dimension of a bivariate spline space is larger than the lower bound (28.5). This was first observed [86] by considering the space $S_2^1(\Delta_{MS})$, where Δ_{MS} is the triangulation as in Figure 28.1. The dimension of this spline space is equal to 7 if Δ_{MS} fulfills certain symmetry properties [127], and otherwise, $S_2^1(\Delta_{MS})$ coincides with Π_2 . Thus, this example shows that the dimension can depend on geometrical properties of the whole triangulation. In general, such dependencies can appear if the degree q is nearby the smoothness r (see [23,51]).

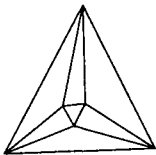


Figure 28.1: Morgan-Scott example: the dimension of $S_2^1(\Delta_{MS})$ depends on the geometry of Δ_{MS} .

We proceed by describing cases where the dimension of $S_q^r(\Delta)$ is known for arbitrary triangulations Δ .

In 1975, the dimension of $S_q^1(\Delta)$, $q \geq 5$, was determined [85] by constructing a *nodal basis* which is based on Hermite interpolation by these spaces. The following formula holds for an arbitrary triangulation Δ :

$$\dim S_q^1(\Delta) = \binom{q+2}{2} N - (2q + 1) E_I + 3V_I + \sigma, \quad q \geq 5, \tag{28.6}$$

where σ denotes the number of *singular vertices*, i.e. interior vertices of Δ that have only two edges with different slopes attached to it. For C^1 -spline spaces, $\sigma_i = 1$ in (28.5) holds if the corresponding vertex is singular, and in all other cases $\sigma_i = 0$. Therefore, it follows from Euler's formulas

$$E_I = 3V_I + V_B - 3, \quad N = 2V_I + V_B - 2,$$

that (28.6) coincides with the lower bound (28.5). This result was extended [3] to spline spaces $S_q^r(\Delta)$, $q \geq 4r + 1$, by coupling the investigations [115] on splines defined on a *star* (i.e. a set of triangles having one common vertex) with the methods developed in [6,7]. In these research papers the concept of *minimal determining sets* was introduced: given a set \mathcal{D} of linear functionals defined on $\mathcal{S} \subseteq S_q^0(\Delta)$, a set $\mathcal{M} \subseteq \mathcal{D}$ is called a minimal determining set for \mathcal{S} provided that setting λ_s for all $\lambda \in \mathcal{M}$ uniquely determines $s \in \mathcal{S}$. In order to construct such minimal determining sets for spline spaces, the Bernstein-Bézier techniques described in Section 28.2 turned out to be useful, since the Bernstein-Bézier coefficients of the polynomial pieces of a bivariate spline can be understood as linear functionals. In this approach one takes advantage of the fact that this type of linear functionals is directly connected to the smoothness conditions of the space via (28.3).

In 1991, the dimension of $S_q^r(\Delta)$, $q \geq 3r + 2$, was determined [65] for arbitrary triangulations Δ . This result was generalized [67] to super spline spaces of degree at least $3r + 2$ (see also [27]). Again, in these cases the dimension of the (super) spline space coincides with the corresponding lower bound. These results are achieved by using local arguments, i.e. vertices, edges and triangles were considered separately. In particular, it was shown that in these cases a basis of *star-supported* splines exists. On the other hand, it is known that such a basis does not generally exist if $q < 3r + 2$ and $r \geq 1$ (cf. [5]). A result [20] which is connected with this fact was given earlier, where it is shown that in this case the spline spaces are defective in the sense that they do not give *optimal approximation order* (see Section 28.4), in general. Otherwise, i.e. in the case $q \geq 3r + 2$, optimal approximation order was shown in several papers by using different methods and by analysing different aspects (cf. [19,31,48,74]).

But the problem of finding an explicit formula for the dimension of $S_q^r(\Delta)$, $q < 3r + 2$, $r \geq 1$, w.r.t. arbitrary triangulations Δ remains open. The only exception known from the literature is the space $S_4^1(\Delta)$. In 1987, it was shown [7] by using arguments from graph theory which are not purely local that the following formula holds for any Δ :

$$\dim S_4^1(\Delta) = 6V + \sigma - 3.$$

Again, this number coincides with the lower bound (28.5).

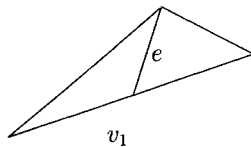


Figure 28.2: A degenerate edge e attached to the vertex v_1 .

One case of particular interest in spline theory is the case of cubic C^1 splines, since it can be seen from the lower bound (28.5) that its dimension must always be larger than the number of vertices of Δ . This is in contrast to the case $S_2^1(\Delta)$, where the lower bound (28.5) equals $V_B + \sigma + 3$. The literature shows that the structure of the space $S_3^1(\Delta)$ is very complex. In fact, at the time of writing of this survey it is still unknown if the dimension of $S_3^1(\Delta)$ is always equal to the lower bound $3V_B + 2V_I + \sigma + 1$, which has been widely conjectured. By using a homological approach [11] it was shown that this conjecture holds *generically*. In addition, the above conjecture holds for general classes of triangulations. In connection with an interpolation method it was proved [47] that the dimension of $S_3^1(\Delta)$ equals the lower bound (28.5), when Δ is contained in the natural class of *nested polygon triangulations* (see Section 28.5). Moreover, a numerical algorithm for determining the dimension of $S_3^1(\Delta)$ was discussed in [60].

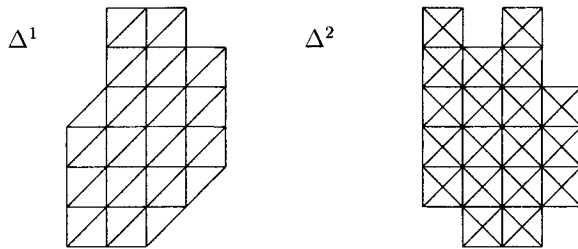


Figure 28.3: The uniform type triangulations Δ^i , $i = 1, 2$.

We proceed by describing results on the dimension of spline spaces w.r.t. general classes of triangulations.

The dimension of $S_{3r+1}^r(\Delta)$, $r \geq 2$, was determined [4] for the general class of *non-degenerate triangulations* Δ , i.e. triangulations that do not contain *degenerate edges* (see Figure 28.2).

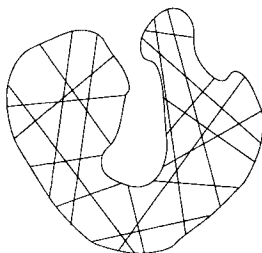


Figure 28.4: A crosscut partition.

Moreover, results on the dimension for *uniform type triangulations* exist in the literature. These are Δ^1 and Δ^2 triangulations, i.e. triangulations as in Figure 28.3. The dimension of such type of spline spaces, and more generally for so-called *crosscut partitions* (see Figure 28.4), was determined for arbitrary q and r [28,29] (see also [114],

and for so-called *quasi crosscut partitions* [80]). For these spline spaces a basis consisting of the polynomials, *truncated power functions* and so-called *cone splines* exists. If such triangulations become non-uniform, i.e. the length of the edges of the underlying quadrangulation are allowed to be different, then the dimension is known in the cases $r = 1$, $q \geq 2$, (for Δ^1) and $r \in \{1, 2\}$, $q \geq r + 1$, (for Δ^2).

In the next two sections, we describe interpolation methods for bivariate (super) splines. We finally remark that the dimension of the spline spaces considered there is always determined.

28.4. FINITE ELEMENT AND MACRO ELEMENT METHODS

In this section, we describe classical finite element methods and its modern extensions, the so-called macro element methods. These are Hermite interpolation methods for bivariate splines, which for low degree splines are based on a suitable splitting procedure applied to every triangle or quadrilateral.

We begin with the classical finite elements (cf. [99]). In 1968, a method [8] was developed which is based on choosing a suitable spline space such that interpolation by bivariate polynomials on every triangle of an arbitrary triangulation Δ automatically leads to Hermite interpolation by C^1 splines. In this method, the super spline space $S_5^{1,\theta_1}(\Delta)$, where $\theta_1 = (2, \dots, 2)$ is used and every polynomial piece in Π_5 is determined separately by interpolating function value, first and second-order derivatives at the vertices, and the cross boundary derivative at the midpoint of each edge (see Figure 28.5). This approach was generalized [132,133] to Hermite interpolation by the super spline spaces $S_q^{r,\theta_r}(\Delta)$, $q \geq 4r + 1$, $r \geq 1$, where $\theta_r = (2r, \dots, 2r)$ (see also [81]). A link between this classical method and modern Bernstein-Bézier techniques was given in [116] (for the special case $S_9^{2,\theta_2}(\Delta)$, see also [130]).

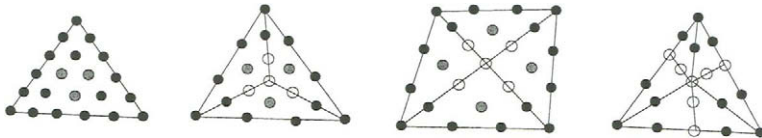


Figure 28.5: The classical finite elements of Argyis, Fried-Scharpf, Clough-Tocher, Fraijs de Veubecke and Sander, and Powell-Sabin (from the left to the right): the Bernstein-Bézier coefficients determined by the interpolation conditions at the vertices are symbolized by black circles, the coefficients determined by the cross boundary derivatives are symbolized by grey circles, and the remaining coefficients determined by the differentiability properties are symbolized by white circles.

In order to keep computational costs small, it is desirable in general, however, to use low degree splines in relation to the smoothness. The following classical methods have been developed for this purpose. The idea of these approaches is to modify the given partition, which can be a triangulation or a convex quadrangulation. In contrast to the finite element method described above, more than one polynomial piece is needed for each triangle or quadrilateral such that the method is local. These classical approaches lead

to Hermite interpolation by cubic and quadratic C^1 splines.

In 1966, a Hermite interpolation set for $S_3^1(\Delta_{CT})$ was constructed [34] (see also [32,33, 52]) where Δ_{CT} is a triangulation obtained from an arbitrary triangulation Δ by splitting each triangle $T \in \Delta$ into three subtriangles, the so-called Clough-Tocher split. This Hermite interpolation set consists of function and gradient value at the vertices of Δ and the cross boundary derivative at the midpoints of all edges of Δ (see Figure 28.5).

Another classical scheme [57,111] for cubic C^1 splines works for triangulated convex quadrangulations (see also [72]). Such triangulations are obtained from a set of convex quadrilaterals by adding both diagonals to every quadrilateral. The corresponding Hermite interpolation set consists of function and gradient value at the vertices and the cross boundary derivative at the midpoints of all edges of the underlying convex quadrangulation (see Figure 28.5).

In 1977, quadratic C^1 splines were considered [105] that interpolate function and gradient value at all vertices of an arbitrary triangulation Δ (see Figure 28.5). The splines were defined w.r.t. the so-called Powell-Sabin triangulation Δ_{PS} , which is obtained by splitting every triangle $T \in \Delta$ into six subtriangles. Here, the splitting points are chosen such that each interior edge of Δ leads to a singular vertex of Δ_{PS} (see Section 28.3). For further results on the Powell-Sabin element and a modification of it, we refer to [40,63,110]. A multiresolution analysis based on quadratic Hermite interpolation using Powell-Sabin splits has recently been constructed in [36].

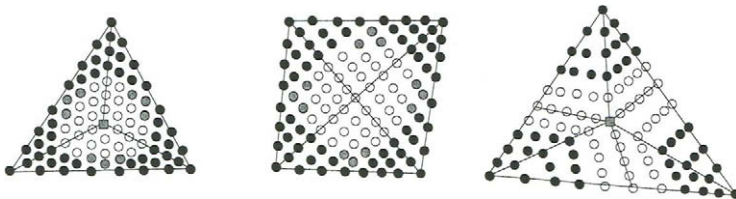


Figure 28.6: The macro elements of Lai-Schumaker (from the left to the right $S_7^{2,\theta}(\Delta_{CT})$, $S_7^{2,\theta}(\Delta)$, $S_5^{2,\theta}(\Delta_{PS})$): the Bernstein-Bézier coefficients determined by the interpolation conditions at the vertices are symbolized by black circles, respectively by a grey box, the coefficients that can be determined by cross boundary derivatives across the edges are symbolized by grey circles, and the remaining coefficients determined by the differentiability properties are symbolized by white circles.

We proceed by describing modern extensions of the above classical methods to spline spaces of higher smoothness, the so-called macro element methods. These methods were developed by using Bernstein-Bézier techniques. In contrast to the above classical elements and the methods described in the next section, these extensions lead to interpolation by super spline subspaces.

We start with the generalizations of the Clough-Tocher element. In 1994, Hermite interpolation sets were constructed [69] for certain super spline spaces $S_{3r}^{r,\theta}(\Delta_{CT})$, where r is odd and $S_{3r+1}^{r,\theta}(\Delta_{CT})$, where r is even. These sets consist of function value and derivatives up to a certain order at the vertices and suitable derivatives at interior points

of each edge of the given triangulation Δ . Later, this method was improved [76] by reducing the number of degrees of freedom. The corresponding Hermite interpolation sets for such splines contain, in addition, the function values and derivatives up to a certain order at the splitting points. An example of this construction is given in Figure 28.6.

A generalization of Fraeijs de Veubecke's and Sander's method for splines on triangulated convex quadrangulations was also developed [70]. The following cases were considered: $S_{3r}^{r,\theta}(\Delta)$ if r is odd, $S_{3r+1}^{r,\theta}(\Delta)$ if r is even. Here, the components of θ concerning the vertices of the underlying quadrangulation are $\frac{3r-1}{2}$ if r is odd and $\frac{3r}{2}$ if r is even. The corresponding Hermite interpolation method is to interpolate function value and derivatives up to order $r + \lfloor \frac{r}{2} \rfloor$ at the vertices and suitable derivatives at interior points of each edge of the underlying quadrangulation. For such type of triangulations Δ a *quasi interpolation method* for the space $S_{3r}^{r,\theta}(\Delta)$, $r \geq 1$, was developed [75]. Earlier, the particular case $S_6^{2,\theta}(\Delta)$, where $\rho_i \in \{2, 3\}$ was investigated [73]. In this case, the quadrilaterals of the underlying quadrangulation do not need to be convex and the super spline property appears only at certain interior vertices. Recently, macro elements for the above type of spline spaces were constructed [78] with the aim of removing certain degrees of freedom at the intersection points of the diagonals. This is done by assuming an additional *supersmoothness* at these points. An example of this construction is given in Figure 28.6.

Now, we discuss generalizations of the Powell-Sabin element. In 1996, the triangulation Δ_{PS}^1 was considered [71]. This triangulation is obtained by applying the Powell-Sabin split to each triangle of a Δ^1 triangulation (see Section 28.3). There it is shown that the function value and the derivatives up to order $r + \lfloor \frac{r}{2} \rfloor$ at all vertices of Δ^1 build a Hermite interpolation set for the super spline space $S_{2r}^{r,\theta}(\Delta_{PS}^1)$ if r is odd, $S_{2r+1}^{r,\theta}(\Delta_{PS}^1)$ if r is even, where $\theta = (r + \lfloor \frac{r}{2} \rfloor, \dots, r + \lfloor \frac{r}{2} \rfloor)$. Later, Hermite interpolation sets were constructed [77] for lower degree super spline spaces with respect to Δ_{PS} . These sets contain, in addition, the function values and derivatives of a certain order at the splitting points. An example of this construction is given in Figure 28.6.

We finally remark that it was shown that the constructions [75–78] yield to so-called *stable, local bases*, which implies that the associated spline space \mathcal{S} has optimal approximation order, i.e., for each sufficiently differentiable f ,

$$\text{dist}(f, \mathcal{S}) \leq Kh^{r+1},$$

where h is the maximal diameter of the triangles and K is a constant depending on no other geometrical properties than the smallest angle in the corresponding triangulation. Such bases have also been constructed for (super) spline spaces of degree $g \geq 3r + 2$ on arbitrary triangulations (cf. [31,45,74]). In this case, a Hermite interpolation operator of a super spline subspace that yields optimal approximation order, where the corresponding fundamental splines have minimal support was constructed in [48].

28.5. INTERPOLATION BY SPLINE SPACES

Considering the results discussed in the previous sections, the natural problem of constructing interpolation sets for the spline space $S_q^r(\Delta)$ appears. In particular, since the Hermite interpolation methods described above cannot be transformed into Lagrange interpolation on the whole triangulation straightforwardly, the question arises: how can

Lagrange interpolation sets for spline spaces be constructed? We note that concerning the construction and reconstruction of surfaces, it is sometimes desirable that only function values are involved and no (cross boundary) derivatives have to be estimated. For example, in many practical applications a surface is described by a linear spline on a fine triangulation (i.e. with many triangles), and an interpolating spline on a coarse subtriangulation can then be constructed by taking the Lagrange data directly from the linear spline.

Since interpolation by splines is strongly connected with the problem of determining the dimension, the literature shows that it is a complex problem to construct explicit interpolation schemes for $S_q^r(\Delta)$, and in particular for Lagrange interpolation. Indeed, there are cases where not even one single Lagrange interpolation set is known. However, as described below, many efficient interpolation methods were developed for splines of certain degree q and smoothness r w.r.t. certain classes of triangulations. Moreover, we mention that in contrast to the case of univariate splines, Schoenberg-Whitney type conditions do not characterize interpolation by bivariate splines : the natural multivariate analogue of such conditions [42,49] characterizes *almost interpolation*, but not interpolation (see also [124]).

In this section, we summarize results on Hermite- and Lagrange interpolation by spline spaces.

First, it is obvious that a Lagrange interpolation set for $S_q^0(\Delta)$, $q \geq 1$, is obtained by the union of all points $\frac{\alpha_0}{q} \mathbf{t}_0 + \frac{\alpha_1}{q} \mathbf{t}_1 + \frac{\alpha_2}{q} \mathbf{t}_2$, $\alpha_0 + \alpha_1 + \alpha_2 = q$, where $T = [\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2]$ is a triangle in Δ with vertices $\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2$. In particular, the set of vertices of Δ is a Lagrange interpolation set for $S_1^0(\Delta)$. An algorithm [30] for constructing more general Lagrange interpolation sets for $S_1^0(\Delta)$ was given in 1986, and recently, a characterization [50] of Lagrange interpolation sets for $S_1^0(\Delta)$ was found. For $r \geq 1$ the interpolation problem becomes more complex. In this case, explicit interpolation schemes were given in the literature for certain classes of triangulations respectively for splines of certain degree q and smoothness r . In the following we describe these methods.

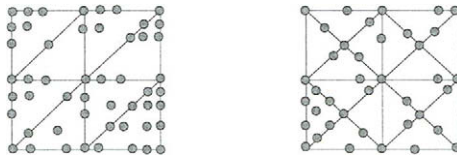


Figure 28.7: Example for the interpolation method of Nürnberger-Rießinger: the Lagrange interpolation points for $S_1^1(\Delta^1)$ and $S_3^1(\Delta^2)$ are symbolized by grey circles.

Many research papers (see for instance [10,25,68,109,123,125,126]) that appeared between 1981 and 1994 investigated the problem of constructing interpolation sets for cubic and quadratic C^1 splines w.r.t. uniform type triangulations (see Figure 28.3). Then, in the beginning 90ies, a general method [89,90] for constructing Lagrange and Hermite interpolation sets for $S_q^r(\Delta^i)$, $i = 1, 2$, was developed. The basic idea of this method is to construct line segments in Ω and to place points on these lines which satisfy the *interlacing condition* of Schoenberg-Whitney for certain univariate spline spaces such

that the *principle of degree reduction* can be applied. Figure 28.7 shows examples of Lagrange interpolation sets constructed by this method. Hermite interpolation sets for $S_q^r(\Delta^i)$, $i = 1, 2$, are obtained by using these Lagrange interpolation sets and by "taking limits", which means, roughly speaking, that the points are shifted to the vertices. An extension of this method to crosscut partitions was given in [1] (see also [98]). Results on the approximation order of this method can be found in [46,88,91].

We proceed by describing interpolation methods for classes more general than Δ^i , $i = 1, 2$. These classes are triangulations constructed from given points, arbitrary triangulations and general classes of triangulations.

We start with methods [93], where triangulations are constructed which are suitable for Lagrange and Hermite interpolation by $S_q^r(\Delta)$, $q \geq 2r + 1$, $r = 1, 2$. These methods are based on an inductive principle: by starting with one triangle, in each step a set of triangles building a suitable polyhedron is added to the subtriangulation constructed so far. The vertices of these triangles are locally chosen scattered points. The construction of the triangulation is such that the corresponding splines can be extended in each step. Lagrange and Hermite interpolation sets were constructed simultaneously, and again, Hermite interpolation sets are obtained by "taking limits". The corresponding interpolating splines can be computed step by step by using Bernstein-Bézier techniques, where in each step only small linear systems of equations have to be solved. Examples of Lagrange interpolation points inside a polyhedron are given in Figure 28.8. Numerical tests with large numbers of interpolation conditions showed that this interpolation method yields good approximations for $S_q^1(\Delta)$, $q \geq 4$, and $S_q^2(\Delta)$, $q \geq 7$. In order to obtain good approximations in the remaining cases (for non-uniform triangulations Δ) variants based on applying the Clough-Tocher split (see Section 28.4) to some of the triangles were proposed. This general method can be applied to certain classes of given triangulations Δ , in particular the class of triangulated quadrangulations [92]. Moreover, for quadratic C^1 splines a general class of triangulations Δ_Q was given, where Lagrange interpolation at the (non-singular) vertices together with three additional points in the starting triangle is always possible.

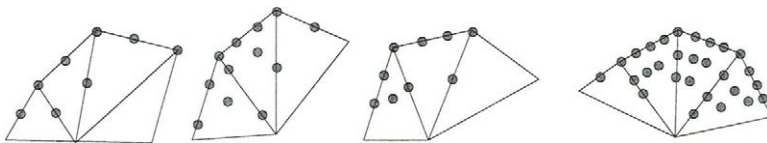


Figure 28.8: Example for the Lagrange interpolation points inside a polyhedron: from the left to the right: $S_3^1(\Delta)$, $S_4^1(\Delta)$, $S_5^2(\Delta)$, $S_6^2(\Delta)$.

We now describe interpolation methods for splines on arbitrary and general classes of triangulations.

Hermite interpolation sets for $S_q^1(\Delta)$, $q \geq 5$, where Δ is an arbitrary triangulation, were constructed in [43,85] (see Section 28.3). Later, a different method [44] was developed for constructing explicit Hermite and Lagrange interpolation sets in these cases. This approach can also be applied to $S_4^1(\Delta)$, where Δ has to be slightly modified if exceptional constellations of triangles occur. Earlier, a Hermite interpolation scheme for $S_4^1(\Delta)$ was

defined [59] in the special case when Δ is an *odd degree triangulation*, i.e. every interior vertex has odd degree. Moreover, quasi-interpolation by $S_4^1(\Delta)$ was considered [26]. There it is shown that optimal approximation order can be achieved by quasi-interpolation, if certain edges are swapped. The case $S_3^1(\Delta)$ is more complex since not even the dimension of these spaces is known for arbitrary triangulations Δ (see Section 28.3). In 1987, a global approach [60] for constructing Lagrange interpolation sets involving function values at all vertices of a given triangulation Δ was proposed. This method requires to solve a large linear system of equations, where it is not guaranteed that this system is solvable. A method to construct Lagrange and Hermite interpolation for $S_3^1(\Delta)$ was given in [47]. In these investigations, Δ is contained in the general class of so-called nested polygon triangulations, i.e. triangulations consisting of nested closed simple polygons whose vertices are connected by line segments. This construction of interpolation sets for $S_3^1(\Delta)$ is inductive by passing through the points of the nested polygons in clockwise order: in each step, a point of a nested polygon and all triangles with this vertex having a common edge with the subtriangulation considered so far are added. Then the interpolation points are chosen locally on these triangles, where the number of interpolation points is different if so-called *semi-singular vertices* exist or not. Numerical examples with a large number of interpolation conditions showed that in order to obtain good approximations, it is desirable to subdivide some of the triangles of Δ . The method of constructing interpolation points also works for these modified triangulations.

Recently, the problem of *local Lagrange interpolation* for C^1 splines was investigated [94,97]. In this context local means that the fundamental Lagrange splines s_i determined by $s_i(z_j) = \delta_{i,j}$, $j = 1, \dots, d$, have local support. (Here, $\delta_{i,j}$ denotes Kronecker's symbol.) We note that the classical Hermite interpolation methods described in Section 28.4 cannot be transformed straightforwardly into a local Lagrange interpolation scheme on a given triangulation.

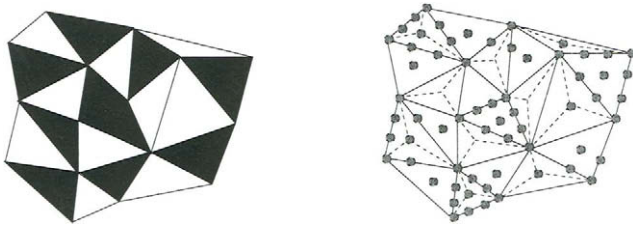


Figure 28.9: Local Lagrange interpolation by cubic C^1 splines: the triangles to be splitted result from the coloring of a given triangulation Δ . The Lagrange interpolation points are symbolized by grey circles.

In [94] (see also [95]) an algorithm was developed for constructing local Lagrange interpolation sets for C^1 splines of degree ≥ 3 . This algorithm mainly consists of two algorithmic steps and is based on an appropriate coloring of the triangles with two colors. Given an arbitrary triangulation Δ , in the first step, Lagrange interpolation points are chosen on the edges of Δ such that the interpolating spline is uniquely determined (only) on the edges. In the second step of this algorithm, the triangles are colored black and

white (by a fast algorithm) such that at most two neighboring triangles have the same color (see Figure 28.9). Then, the white triangles are subdivided by a Clough-Tocher split, and in the interior of the black triangles, additional Lagrange interpolation points are chosen. Finally, the Lagrange interpolating spline is uniquely determined on the whole triangulation. Figure 28.9 shows an example of such an interpolation set for the case of cubic C^1 splines.

Since recently, the construction of local Lagrange interpolation schemes by cubic C^1 splines on quadrangulations of convex quadrilaterals [97] is under investigation. For a special class of triangulated quadrangulations such schemes have been given in [96] (see Figure 28.10). We remark that these interpolation methods yield to optimal approximation order and the corresponding Lagrange interpolating splines can be computed efficiently by using Bernstein-Bézier techniques since the algorithmical complexity of these methods is linear in the number of triangles. Numerical tests given in [94–96] with upto 10^6 Lagrange interpolation points demonstrate the efficiency of these methods.

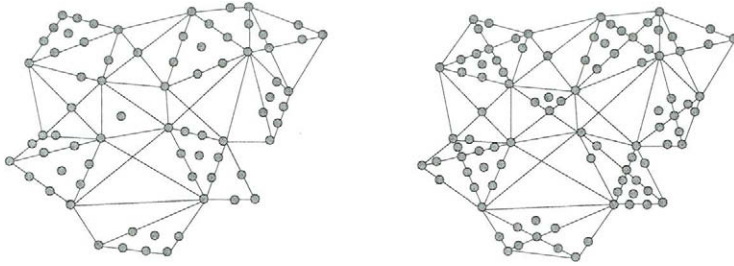


Figure 28.10: Local Lagrange interpolation by cubic C^1 splines on classes of triangulated quadrangulations: the Lagrange interpolation points are symbolized by grey circles.

28.6. TRIANGULAR B-SPLINES

The results of Section 28.4 and 28.5 show that there are many cases where the spline space $S_q^r(\Delta)$ provides powerful tools for applications. On the other hand, the structure of these spaces is very complex, in general. In particular, the discussion of Section 28.3 shows that determining the dimension is difficult for arbitrary triangulations. Therefore, it is a complex task to construct suitable basis functions for these spaces, in general.

In this section, we describe a different approach, which was developed with the aim to construct smooth piecewise polynomial basis functions for splines over arbitrary triangulations. This approach is based on a geometrical way to construct smooth piecewise polynomial functions $M : \mathbb{R}^2 \mapsto \mathbb{R}$ by projecting a polyhedron $P \subset \mathbb{R}^n$ onto \mathbb{R}^2 and by defining $M(\mathbf{u})$ as the $(n - 2)$ -dimensional volume of the fibre $\pi^{-1}(\mathbf{u})$. This definition generalizes the geometric definition of univariate *B-splines* and hence the resulting functions are called multivariate B-splines. Depending on whether P is just any polyhedron, a box, or a simplex, the resulting multivariate B-splines are also called polyhedral, box, or simplex splines.

If the given triangulation Δ happens to be of uniform type (see Figure 28.3, for instance), then box splines are the natural choice. Box splines are a natural generalization

of uniform B-splines and have a very rich structure. In particular, they have a stable recurrence and can be generated by subdivision [21]. In the CAGD context, box splines have been first considered in [108]. Computational aspects and algorithms for converting to piecewise Bernstein-Bézier representation (28.1) through the use of masks have been given in [14]. Surface fitting with box splines was discussed in [35]. The first book which was completely devoted to box splines is [21].

If an arbitrary triangulation Δ is given, then in this approach simplex splines have to be used. Simplex splines can be defined recursively as follows: Given the knots $\mathbf{t}_0, \dots, \mathbf{t}_{q+2} \in \mathbb{R}^2$ one can show [82,83] that the recursion

$$M(\cdot | \mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2) = \frac{\chi(\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2)}{\mathbf{d}(\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2)}$$

$$M(\mathbf{u} | \mathbf{t}_0, \dots, \mathbf{t}_{l+2}) = \sum_{i=0}^{l+2} \lambda_i M(\mathbf{u} | \mathbf{t}_0, \dots, \mathbf{t}_{i-1}, \mathbf{t}_{i+1}, \dots, \mathbf{t}_{l+2}), \quad l = 1, \dots, q,$$

with $\mathbf{u} = \sum_i \lambda_i(\mathbf{u}) \mathbf{t}_i$, and $\sum_i \lambda_i(\mathbf{u}) = 1$ is well-defined and yields a simplex spline M of degree q that is C^{q-1} continuous if the knots are in general position. (Here, $\mathbf{d}(\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2)$ stands for the area of the triangle $T = [\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2]$ multiplied by two.) Further details on simplex splines can be found, e.g., in [15,38,61,84,122].

The next problem is to construct a spline space from these functions. While for box splines one can consider the space spanned by translates, this problem is difficult for simplex splines: given an arbitrary triangulation Δ , exactly what simplex splines should be considered?

Solutions to this problem were given first in [39,64], and later in [41,121]. These constructions start with a triangulation Δ , where for every vertex \mathbf{t}_i of Δ a cloud of points $\mathbf{t}_{i,0}, \dots, \mathbf{t}_{i,q}$ is assigned. Then a rule of selecting $\binom{q+2}{2}$ subsets of $q + 3$ knots from the three clouds associated with a triangle is given. Each such subset yields a simplex spline of degree q which is generally C^{q-1} . The linear span Ξ_q of all these degree q simplex splines is then the spline space of interest. Note that both schemes produce splines over a refined partition of Δ .

The two schemes differ in the knot selection rule, in the ease of use, and in the class of surfaces that they are able to represent. The first scheme [39,64] contains the polynomial space, i.e. $\Pi_q \subset \Xi_q$, but the representation of arbitrary piecewise polynomials remained unsolved. This defect was overcome by the scheme [41,121]. The knot selection rule of this scheme is based on the use of polar forms [117,120] (see Section 28.2), and for a given triangle $T = [\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2]$ selects the simplex splines

$$M_\alpha^{T,q}(\mathbf{u}) = M(\mathbf{u} | V_\alpha), \quad |\alpha| = q,$$

with

$$V_\alpha = \{\mathbf{t}_{0,0}, \dots, \mathbf{t}_{0,\alpha_0}, \mathbf{t}_{1,0}, \dots, \mathbf{t}_{1,\alpha_1}, \mathbf{t}_{2,0}, \dots, \mathbf{t}_{2,\alpha_2}\}.$$

Furthermore, the new scheme not only allows the representation of the polynomials, but also allows the representation of piecewise polynomials, i.e. $S_q^{q-1}(\Delta) \subset \Xi_q$. Moreover, up to normalization, the coefficients in the resulting representation

$$F(\mathbf{u}) = \sum_{T \in \Delta, \alpha} M_\alpha^{T,q}(\mathbf{u}) \mathbf{d}_\alpha^T$$

of a piecewise polynomial F as a linear combination of simplex splines are given as

$$\mathbf{d}_\alpha^T = f_T(\mathbf{t}_{0,0} \cdots \mathbf{t}_{0,\alpha_0-1} \mathbf{t}_{1,0} \cdots \mathbf{t}_{1,\alpha_1-1} \mathbf{t}_{2,0} \cdots \mathbf{t}_{2,\alpha_2-1})$$

by evaluating the polar form f_T (see Section 28.2) of the restriction F_T of F to the triangle $T = [\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2]$ on a suitable sequence of knots [118]. Note that this formula captures completely the analog formula for the de Boor points in the B-spline expansion of a univariate spline, and also the formula for the Bernstein-Bézier points in the expansion of a polynomial surface in the representation (28.1).

Practical aspects of implementing the simplex spline scheme [41,121] have been discussed [9,129]. A first implementation on triangular B-splines has been described in [56,62]. Efficient evaluation routines for triangular B-splines have been given in [100] for the quadratic case, and in [58] for arbitrary degree q . Here, efficiency is obtained by reusing partial results. Algorithms for surface fitting and modeling with triangular B-splines have been discussed in [102–104]. Finally extensions of the approach to a spherical setting have been presented in [101].

Acknowledgment. The authors thank Günther Nürnberger for discussing the various topics described in this paper.

REFERENCES

1. M.H. Adam. *Bivariate Spline-Interpolation auf Crosscut-Partitionen*. PhD thesis, University of Mannheim, 1995.
2. P. Alfeld. Upper and lower bounds on the dimension of multivariate spline spaces. *SIAM J. Numer. Anal.*, 33(2):571–588, 1996.
3. P. Alfeld and L.L. Schumaker. The dimension of bivariate spline spaces of smoothness r for degree $d \geq 4r + 1$. *Constr. Approx.*, 3:189–197, 1987.
4. P. Alfeld and L.L. Schumaker. On the dimension of bivariate spline spaces of smoothness r and degree $d = 3r + 1$. *Numer. Math.*, 57:651–661, 1990.
5. P. Alfeld and L.L. Schumaker. Non-existence of star-supported spline bases. *SIAM J. Math. Anal.*, 31:1482–1501, 2000.
6. P. Alfeld, B. Piper, and L.L. Schumaker. Minimally supported bases for spaces of bivariate piecewise polynomials of smoothness r and degree $d \geq 4r + 1$. *Computer Aided Geometric Design*, 4:105–123, 1987.
7. P. Alfeld, B. Piper, and L.L. Schumaker. An explicit basis for C^1 quartic bivariate splines. *SIAM J. Numer. Anal.*, 24:891–911, 1987.
8. J.H. Argyis, I. Fried, and D.W. Scharpf. The TUBA family of plate elements for the matrix displacement method. *Aeronaut. J. Roy. Aeronaut. Soc.*, 72:701–709, 1968.
9. S. Auerbach, R.H.J. Gmelig Meyling, M. Neamtu, and H. Schaeben. Approximation and geometric modeling with simplex B-splines associated with irregular triangles. *Computer Aided Geometric Design*, 8:67–87, 1991.
10. R.K. Beatson and Z. Ziegler. Monotonicity preserving surface interpolation. *SIAM J. Numer. Anal.*, 22(2):401–411, 1985.
11. L.J. Billera. Homology of smooth splines: generic triangulations and a conjecture of Strang. *Trans. Am. Math. Soc.*, 310(2):325–340, 1988.

12. W. Boehm, G. Farin, and J. Kahmann. A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design*, 1:1-60, 1984.
13. W. Boehm, J. Hoschek, and H.-P. Seidel. Mathematical aspects of computer aided geometric design, In M. Artin, H. Kraft, and R. Remmert, editors. *Duration and Change: Fifty Years at Oberwolfach*, pages 106–138. Springer, 1994.
14. W. Boehm, H. Prautzsch, and P. Arner. On triangular splines. *Constr. Approx.*, 3:157–167, 1987.
15. B.D. Bojanov, H.A. Hakopian, and A.A. Sahakian. *Spline Functions and Multivariate Approximation*. Kluwer, Dordrecht, 1993.
16. C. de Boor. *A Practical Guide to Splines*. Springer, New York, 1978.
17. C. de Boor. B-form basics. In G. Farin, editor, *Geometric Modeling*, pages 131-148. SIAM, Philadelphia, 1987.
18. C. de Boor. Multivariate piecewise polynomials. *Acta Numerica*, 65–109, 1993.
19. C. de Boor and K. Höllig. Approximation power of smooth bivariate pp functions. *Math. Z.*, 197:343-363, 1988.
20. C. de Boor and Q. Jia. A sharp upper bound on the approximation order of smooth bivariate pp functions. *J. Approx. Theory*, 72:24-33, 1993.
21. C. de Boor, K. Höllig, and S. Riemenschneider. *Box Splines*. Springer, Berlin, 1993.
22. P. de Casteljau. *Formes à pôles*, Hermes, Paris, 1985.
23. Z.B. Chen, Y.Y. Feng, and J. Kozak. The blossoming approach to the dimension of the bivariate spline space. *J. Comp. Math.*, 18:183–198, 2000.
24. C.K. Chui. *Multivariate Splines*, CBMS 54. SIAM, Philadelphia, 1988.
25. C.K. Chui and T.X. He. On location of sample points in C^1 quadratic bivariate spline interpolation. In L. Collatz, G. Meinardus, and G. Nürnberger, editors. *Numerical Methods of Approximation Theory*, pages 30–43, ISNM 81. Birkhäuser, Basel, 1987.
26. C.K. Chui and D. Hong. Swapping edges of arbitrary triangulations to achieve the optimal order of approximation. *SIAM J. Numer. Anal.*, 34:1472–1482, 1997.
27. C.K. Chui and M.-J. Lai. On bivariate super vertex splines. *Constr. Approx.*, 6:399–419, 1990.
28. C.K. Chui and R.H. Wang. Multivariate spline spaces. *J. Math. Anal. Appl.*, 94:197–221, 1983.
29. C.K. Chui and R.H. Wang. On smooth multivariate spline functions. *Math. Comput.*, 41:131–142, 1983.
30. C.K. Chui, T.X. He, and R.H. Wang. Interpolation by bivariate linear splines, In J. Szabados and J. Tandori, editors. *Alfred Haar Memorial Conference*, pages 247–255. North Holland, Amsterdam, 1986.
31. C.K. Chui, D. Hong, and Q. Jia. Stability of optimal-order approximation by bivariate splines over arbitrary triangulations. *Trans. Amer. Math. Soc.*, 347:3301-3318, 1995.
32. P.G. Ciarlet. Sur l'élément de Clough et Tocher. *RAIRO Anal. Numér.*, 2:19–27, 1974.
33. P.G. Ciarlet. Interpolation error estimates for the reduced Hsieg-Clough-Tocher triangles. *Math. Comp.*, 32:335–344, 1978.
34. R.W. Clough and J.L. Tocher. Finite element stiffness matrices for analysis of plates in bending. In *Proc. Conf. on Matrix Methods in Structural Mechanics*. Wright Patterson A.F.B., Ohio, 1965.

35. M. Dæhlen and T. Lyche. Boxsplines and applications. In *Geometric Modelling*, pages 35–93. Springer, 1991.
36. M. Dæhlen, T. Lyche, K. Mørken, R. Schneider, and H.-P. Seidel. Multiresolution analysis over triangles based on quadratic Hermite interpolation. *J. Comp. Appl. Math.*, 119:97–114, 2000.
37. W. Dahmen. Bernstein-Bézier representation of polynomial surfaces. In *Proc. ACM SIGGRAPH*, Dallas, 1986.
38. W. Dahmen and C.A. Micchelli. Recent progress in multivariate splines. In C.K. Chui, L.L. Schumaker, and J.D. Ward, editors. *Approximation Theory IV*, pages 27–121. Academic Press, New York, 1983.
39. W. Dahmen and C.A. Micchelli. On the linear independence of multivariate B-splines, I. Triangulations of simploids, *SIAM J. Numer. Anal.*, 19:993–1012, 1982.
40. W. Dahmen, R.H.J. Gmelig Meyling, and J.H.M. Ursem. Scattered data interpolation by bivariate C^1 -piecewise quadratic functions. *Approx. Theory Appl.*, 6:6–29, 1990.
41. W. Dahmen, C.A. Micchelli, and H.-P. Seidel. Blossoming begets B-splines built better by B-patches. *Math. Comp.*, 59:97–115, 1992.
42. O. Davydov. On almost interpolation. *J. Approx. Theory*, 91:398–418, 1997.
43. O. Davydov. Locally linearly independent basis for C^1 bivariate splines of degree $q \geq 5$, In M. Daehlen, T. Lyche and L.L. Schumaker, editors. *Mathematical Methods for Curves and Surfaces II*, pages 71–77. Vanderbilt University Press, Nashville, 1998.
44. O. Davydov and G. Nürnberger. Interpolation by C^1 splines of degree $q \geq 4$ on triangulations. *J. Comput. Appl. Math.*, 126:159–183, 2000.
45. O. Davydov and L.L. Schumaker. On stable local bases for bivariate polynomial splines. *Constr. Approx.*, to appear.
46. O. Davydov, G. Nürnberger, and F. Zeilfelder. Approximation order of bivariate spline interpolation for arbitrary smoothness. *J. Comp. Appl. Math.*, 90:117–134, 1998.
47. O. Davydov, G. Nürnberger, and F. Zeilfelder. Cubic spline interpolation on nested polygon triangulations, In A. Cohen, C. Rabut, and L.L. Schumaker, editors, *Curve and Surface Fitting: Saint Malo 1999*, pages 161–170. Vanderbilt University Press, Nashville, 2000.
48. O. Davydov, G. Nürnberger, and F. Zeilfelder. Bivariate spline interpolation with optimal approximation order. *Constr. Approx.*, 17:181–208, 2001.
49. O. Davydov, M. Sommer, and H. Strauss. On almost interpolation and locally linearly independent basis. *East J. Approx.*, 5:67–88, 1999.
50. O. Davydov, M. Sommer, and H. Strauss. Interpolation by bivariate linear splines. *J. Comp. Appl. Math.*, 119:115–131, 2000.
51. D. Diener. Instability in the dimension of spaces of bivariate piecewise polynomials of degree $2r$ and smoothness order r . *SIAM J. Numer. Anal.*, 27(2):543–551, 1990.
52. G. Farin. A modified Clough-Tocher interpolant. *Computer Aided Geometric Design*, 2:19–27, 1985.
53. G. Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3:83–127, 1986.
54. G. Farin. *Geometric Modeling*. SIAM, Philadelphia, 1987.
55. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic

Press, 1993.

56. P. Fong and H.-P. Seidel. An implementation of triangular B-spline surfaces over arbitrary triangulations. *Computer Aided Geometric Design*, 10:267–275, 1993.
57. G. Fraeijs de Veubeke. Bending and stretching of plates, in: *Proc. Conf. on Matrix Methods in Structural Mechanics*. Wright Patterson A.F.B., Ohio, 1965.
58. M. Franssen, R. Veltkamp, and W. Wesselink. Efficient evaluation of triangular B-spline surfaces. *Computer Aided Geometric Design*, 17:863–877, 2000.
59. J. Gao. Interpolation by C^1 quartic bivariate splines. *J. Math. Res. Expo.*, 11:433–442, 1991.
60. R.H.J. Gmelig Meyling. Approximation by piecewise cubic C^1 -splines on arbitrary triangulations. *Numer. Math.*, 51:65–85, 1987.
61. T. Grandine. The stable evaluation of multivariate simplex splines. *Math. Comp.*, 50:197–205, 1988.
62. G. Greiner and H.-P. Seidel. Modeling with triangular B-splines. *IEEE Computer Graphics Applications*, 14(2):56–60, 1994.
63. G. Heindl. Interpolation and approximation by piecewise quadratic C^1 functions of two variables, In W. Schempp and K. Zeller, editors, *Multivariate Approximation Theory*, pages 146–161, ISNM 51. Birkhäuser, Basel, 1979.
64. K. Höllig. Multivariate splines. *SIAM J. Numer. Anal.*, 19:1013–1031, 1982.
65. D. Hong. Spaces of bivariate spline functions over triangulation. *Approx. Theory Appl.*, 7:56–75, 1991.
66. J. Hoschek and D. Lasser. *Grundlagen der geometrischen Datenverarbeitung*. Teubner, Stuttgart, 1992.
67. A. Ibrahim and L.L. Schumaker. Super spline spaces of smoothness r and degree $d \geq 3r + 2$. *Constr. Approx.*, 7:401–423, 1991.
68. F. Jeeawock-Zedek. Operator norm and error bounds for interpolating quadratic splines on a non-uniform type-2 triangulation of a rectangular domain. *Approx. Theory and Appl.*, 10(2):1–16, 1994.
69. M. Laghchim-Lahlou and P. Sablonnière. Triangular finite elements of HCT type and class C^p . *Adv. in Comp. Math.*, 2:101–122, 1994.
70. M. Laghchim-Lahlou and P. Sablonnière. Quadrilateral finite elements of FVS type and class C^r . *Numer. Math.*, 30:229–243, 1995.
71. M. Laghchim-Lahlou and P. Sablonnière. C^r -finite elements of Powell-Sabin type on the three directional mesh. *Adv. Comp. Math*, 6:191–206, 1996.
72. M.-J. Lai. Scattered data interpolation and approximation using bivariate C^1 piecewise cubic polynomials. *Computer Aided Geometric Design*, 13:81–88, 1996.
73. M.-J. Lai and L.L. Schumaker. Scattered data interpolation using C^2 supersplines of degree six. *SIAM J. Numer. Anal.*, 34:905–921, 1997.
74. M.-J. Lai and L.L. Schumaker. On the Approximation Power of Bivariate Splines. *Adv. in Comp. Math.*, 9:251–279, 1998.
75. M.-J. Lai and L.L. Schumaker. On the approximation power of splines on triangulated quadrangulations. *SIAM J. Numer. Anal.*, 36:143–159, 1999.
76. M.-J. Lai and L.L. Schumaker. Macro-elements and stable local bases for splines on Clough-Tocher triangulations. *Numer. Math.*, to appear.
77. M.-J. Lai and L.L. Schumaker. Macro-elements and stable local bases for splines on

- Powell-Sabin triangulations. *Math. Comp.*, to appear.
78. M.-J. Lai and L.L. Schumaker. Quadrilateral macro-elements. *Numer. Math.*, to appear.
 79. C. Manni. On the dimension of bivariate spline spaces over rectilinear partitions. *Approx. Theory and Appl.*, 7(1):23–34, 1991.
 80. C. Manni. On the dimension of bivariate spline spaces over generalized quasi-cross-cut partitions. *J. Approx. Theory*, 69:141–155, 1992.
 81. A. Le Méhauté. Unisolvent interpolation in \mathbb{R}^n and the simplicial finite element method, In C. Chui, L.L. Schumaker, and F. Utreras, editors, *Topics in Multivariate Approximation*, pages 141–151. Academic Press, New York, 1987.
 82. C.A. Michelli. On a numerically efficient method for computing with multivariate B-splines, In W. Schempp, and K. Zeller, editors, *Multivariate Approximation Theory*, pages 211–248. Birkhäuser, Basel, 1979.
 83. C.A. Michelli. A constructive approach to Kergin interpolation in \mathbb{R}^k , multivariate B-splines and Lagrange interpolation. *Rocky Mt. J. Math.*, 10:485–497, 1980.
 84. C.A. Michelli. *Mathematical Aspects of Geometric Modelling*, CBMS 65. SIAM, Philadelphia, 1995.
 85. J. Morgan and R. Scott. A nodal basis for C^1 piecewise polynomials of degree $n \geq 5$. *Math. Comp.*, 29:736–740, 1975.
 86. J. Morgan and R. Scott. The dimension of piecewise polynomials. Unpublished manuscript, 1977.
 87. G. Nürnberger. *Approximation by Spline Functions*. Springer, Berlin, 1989.
 88. G. Nürnberger. Approximation order of bivariate spline interpolation. *J. Approx. Theory*, 87:117–136, 1996.
 89. G. Nürnberger and T. Rießinger. Lagrange and Hermite interpolation by bivariate splines. *Numer. Funct. Anal. Optim.*, 13:75–96, 1992.
 90. G. Nürnberger and T. Rießinger. Bivariate spline interpolation at grid points. *Numer. Math.*, 71:91–119, 1995.
 91. G. Nürnberger and G. Walz. Error analysis in interpolation by bivariate C^1 -splines. *IMA J. Numer. Anal.*, 18:485–508, 1998.
 92. G. Nürnberger and F. Zeilfelder. Spline interpolation on convex quadrangulations, In C.K. Chui and L.L. Schumaker, editors, *Approximation Theory IX*, pages 259–266. Vanderbilt University Press, Nashville, 1998.
 93. G. Nürnberger and F. Zeilfelder. Interpolation by spline spaces on classes of triangulations. *J. Comput. Appl. Math.*, 119:347–376, 2000.
 94. G. Nürnberger and F. Zeilfelder. Lagrange interpolation by bivariate C^1 -splines with optimal approximation order. Submitted.
 95. G. Nürnberger and F. Zeilfelder. Local Lagrange interpolation by cubic splines on a class of triangulations. In K. Kopotun, T. Lyche, and M. Neamtu, editors, *Trends in Approximation Theory*, pages 341–350. Vanderbilt University Press, Nashville, 2001.
 96. G. Nürnberger, L.L. Schumaker, and F. Zeilfelder. Local Lagrange interpolation by bivariate C^1 cubic splines. In T. Lyche and L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces III*, pages 393–404. Vanderbilt University Press, Nashville, 2000.
 97. G. Nürnberger, L.L. Schumaker, and F. Zeilfelder. Lagrange interpolation by C^1 cubic

- splines on triangulations of separable quadrangulations. Submitted.
98. G. Nürnberger, O. Davydov, G. Walz, and F. Zeilfelder. Interpolation by bivariate splines on crosscut partitions, In G. Nürnberger, J.W. Schmidt, and G. Walz, editors, *Multivariate Approximation and Splines*, ISNM 125, pages 189–204. Birkhäuser, Basel, 1997.
 99. P. Oswald. *Multilevel Finite Element Approximation: Theory and Applications*. Teubner, Stuttgart, 1994.
 100. R. Pfeifle and H.-P. Seidel. Faster evaluation of quadratic bivariate DMS spline surfaces. *Proc. Graphics Interface '94, Banff*, page 182–189. Morgan Kaufman Publishers, 1994.
 101. R. Pfeifle and H.-P. Seidel. Spherical triangular B-splines with applications to data fitting, *Eurographics '95*,14(3):89–96, 1995.
 102. R. Pfeifle and H.-P. Seidel. Fitting triangular B-splines to functional scattered data. *Computer Graphics Forum*, 15(1):15–23, 1996.
 103. R. Pfeifle and H.-P. Seidel. Scattered data approximation with triangular B-splines, In J. Hoschek and P. Kaklis, editors, *Advance Course on Fairshape*, pages 253–263. Teubner, Stuttgart, 1996.
 104. R. Pfeifle and H.-P. Seidel. Triangular B-splines for blending and filling polygonal holes. *Proc. Graphics Interface '96, Toronto*, pages 186–193. Morgan Kaufman Publishers, 1996.
 105. M.J.D. Powell and M.A. Sabin. Piecewise quadratic approximation on triangles. *ACM Trans. Math. Software*, 4:316–325, 1977.
 106. L. Ramshaw. Blossoms are polar forms. *Computer Aided Geometric Design*, 6:323–358, 1989.
 107. D.J. Ripmeester. Upper bounds on the dimension of bivariate spline spaces and duality in the plane, In M. Daehlen, T. Lyche and L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, pages 455–466. Vanderbilt University Press, Nashville, 1995.
 108. M. Sabin. *The Use of Piecewise Forms for Numerical Representation of Shape*. PhD thesis, Hungarian Academy of Science, Budapest, Hungary, 1976.
 109. P. Sablonnière. Bernstein-Bézier methods for the construction of bivariate spline approximants. *Computer Aided Geometric Design*, 2:29–36, 1985.
 110. P. Sablonnière. Error bounds for Hermite interpolation by quadratic splines on an α -triangulation. *IMA J. Numer. Anal.*, 7:495–508, 1987.
 111. G. Sander. Bornes supérieures et inférieures dans l'analyse matricielle des plaques en flexion-torsion. *Bull. Soc. Royale Science Liège*, 33:456–494, 1964.
 112. L.L. Schumaker. *Spline Functions: Basic Theory*. Wiley-Interscience, New York, 1980.
 113. L.L. Schumaker. On the dimension of piecewise polynomials in two variables. In W. Schempp and K. Zeller, editors. *Multivariate Approximation Theory*, pages 396–412. Birkhäuser, Basel, 1979.
 114. L.L. Schumaker. Bounds on the dimension of spaces of multivariate piecewise polynomials. *Rocky Mountain J. Math.*, 14:251–264, 1984.
 115. L.L. Schumaker. Dual bases for spline spaces on a cell, *Computer Aided Geometric Design*, 5:277–284, 1987.

116. L.L. Schumaker. On super splines and finite elements. *SIAM J. Numer. Anal.*, 4:997–1005, 1989.
117. H.-P. Seidel. Symmetric recursive algorithms for surfaces: B-patches and the de Boor algorithm for polynomials over triangles. *Constr. Approx.*, 7:257–279, 1991.
118. H.-P. Seidel. Representing piecewise polynomials as a linear combination of multivariate B-splines, In T. Lyche and L.L. Schumaker, editors, *Curves and Surfaces*, pages 559–566. Academic Press, 1992.
119. H.-P. Seidel. An introduction to polar forms. *IEEE Computer Graphics & Applications*, 13(1):38–46, 1993.
120. H.-P. Seidel. Polar forms and triangular B-spline surfaces, In F. Du and F. Hwang, editors, *Computing in Euclidian Geometry*, pages 299–350. World Science Publishers, 1995.
121. H.-P. Seidel. Simplex splines, polar simplex splines and triangular B-splines, In G. Mullineux, editor, *Mathematics of Surfaces VI*, pages 535–547. Oxford University Press, 1995.
122. H.-P. Seidel and A. Vermeulen. Simplex splines support surprisingly strong symmetric structures and subdivision, In P.J. Laurent, A. Le Méhauté, and L.L. Schumaker, editors, *Curves and Surfaces III*, pages 443–455. AK Peters, Boston, 1994.
123. R. Sibson and G.D. Thomson. A seamed quadratic element for contouring. *Computer Journal*, 24(4):378–382, 1981.
124. M. Sommer and H. Strauss. A condition of Schoenberg-Whitney type for multivariate spline interpolation. *Adv. in Comp. Math.*, 5:381–397, 1996.
125. Z. Sha. On interpolation by $S_3^1(\Delta_{m,n}^1)$. *Approx. Theory Appl.*, 1:1–18, 1985.
126. Z. Sha. On interpolation by $S_2^1(\Delta_{m,n}^2)$. *Approx. Theory Appl.*, 1:71–82, 1985.
127. X.Q. Shi. The singularity of Morgan-Scott triangulation. *Computer Aided Geometric Design*, 8:201–206, 1991.
128. G. Strang. Piecewise polynomials and the finite element method. *Bull. Amer. Math. Soc.*, 79:1128–1137, 1973.
129. C. Traas. Practice of bivariate quadratic simplicial splines, in W. Dahmen, M. Gasca, and C.A. Michelli, editors, *Computation of Curves and Surfaces*, pages 383–422, NATO ASI Series. Kluwer Academic Publishers, 1990.
130. T. Whelan. A representation of a C^2 interpolant over triangles. *Computer Aided Geometric Design*, 3:53–66, 1986.
131. F. Zedek. Interpolation de Lagrange par des splines quadratique sur un quadrilatere de \mathbb{R}^2 . *RAIRO Anal. Numér.*, 26:575–593, 1992.
132. A. Ženišek. Interpolation polynomials on the triangle. *Numer. Math.*, 15:283–296, 1970.
133. A. Ženišek. A general theorem on triangular finite C^m -elements. *RAIRO Anal. Numér.*, 2:119–127, 1974.

Chapter 29

Kinematics and Animation

Bert Jüttler and Michael G. Wagner

This chapter demonstrates that the techniques of Computer Aided Geometric Design can be generalized to Kinematics, Computer Animation, and Robotics. Our approach relies on spatial rational spline motions which can be seen as the kinematical analogue of rational spline curves. The potential applications include keyframe interpolation in Computer Graphics, motion planning in Robotics, and sweep surface modelling in Geometric Design.

29.1. INTRODUCTION

The idea to use the powerful tools of Computer Aided Geometric Design in spatial kinematics originated in Computer Graphics, where rigid body motions are needed for visualizing moving objects in Computer Animation (keyframe interpolation), and for generating smooth camera motions, e.g., in Virtual Reality. Initially, the Bézier technique was generalized to the unit quaternion sphere via ‘slerping’ (see Section 29.4.1), following ideas by Shoemake and others [34,37,46]. This technique generates unit quaternion curves which can be identified with spherical motions, thus representing the rotational part of a rigid body motion. Although this and similar generalizations seem to work well, and are apparently still in use in Computer Graphics [14], it was soon realized that these spherical generalizations of the standard algorithms lead to major difficulties, such as the absence of a subdivision property, non-linear interpolation conditions, the difficult parametric representation of the resulting point trajectories, and problems with the construction of C^2 (acceleration continuous) motions, see Section 29.4.2 for more.

Independently, a similar approach was developed by Ge and Ravani in Robotics, for designing robot motions via Bézier type curves in dual quaternion space [10]. Unlike the slerp techniques, this approach produces motions with rational point trajectories, the so-called *rational motions*. In kinematical geometry, these motions had been studied since the end of the 19th century [5,43,52].

Another source of the theory of rational motions can be identified in the discussion of sweeping (kinematical) surfaces. Sweeping is a very intuitive technique for generating

free-form surfaces, by moving a (rigid or possibly evolving) profile curve through space, see Figure 12 of Bézier's preface to [8]. One of the first publications on rational sweeping surfaces is due to Röschel [32,44].

Computational techniques for rational spline motions have been further explored in the simultaneous Ph.D. theses of the two authors [20,47]. These motions have now been developed into a useful tool for geometric motion design and for applications in Robotics, Computer Graphics, and Geometric Modelling. Some of the results have been gathered in this chapter, which is organized as follows. The next two sections summarize fundamentals from spatial kinematics and on quaternions. Section 4 is devoted to the various non-rational techniques for motion design, using curves on the unit quaternion sphere. After introducing spherical rational motions (Section 5), we give an outline of available algorithms for spatial rational motions, along with a brief discussion of several applications. Finally we conclude this chapter and suggest some directions for further research.

29.2. THE KINEMATICAL MAPPING

This section collects some facts about the description of rigid body motions by homogeneous 4×4 matrices. We use Euler parameters to represent rotation matrices, leading directly to the kinematical mapping of spherical kinematics.

29.2.1. Coordinates

In the sequel we describe the points \mathbf{p} in 3-space with the help of homogeneous coordinates $\mathbf{p} = (p_0, p_1, p_2, p_3)^T \in \mathbb{R}^4 \setminus \{(0, 0, 0, 0)^T\}$. If the 0-th component satisfies $p_0 \neq 0$, we may obtain the corresponding Cartesian coordinates $\underline{\mathbf{p}} = (\underline{p}_1, \underline{p}_2, \underline{p}_3)^T \in \mathbb{R}^3$ of the very point \mathbf{p} from $\underline{p}_i = p_i/p_0$, where $i = 1, 2, 3$. The homogeneous coordinate vectors \mathbf{p} and $\lambda\mathbf{p}$ describe the same point for any constant factor $\lambda \neq 0$. Consequently, the set of points in 3-space, which is projectively closed by adding points at infinity, is identified with the set of all one-dimensional subspaces in \mathbb{R}^4 .

The coordinate p_0 of \mathbf{p} is commonly referred to as the homogenizing coordinate or the *weight* of \mathbf{p} . Points with $p_0 = 0$ correspond to points at infinity; they can be identified with the ∞^2 equivalence classes of parallel lines. For further information on homogeneous coordinates see Chapter 2.

29.2.2. Motions of a rigid body

Let us consider two coordinate systems in three dimensional Euclidean space, the *fixed* coordinate system E^3 ("world coordinates") and the *moving* coordinate system \hat{E}^3 . Points can be described in either coordinate system. We denote the fixed coordinates of a point by \mathbf{p} or $\underline{\mathbf{p}}$, and the moving coordinates by $\hat{\mathbf{p}}$ or $\hat{\underline{\mathbf{p}}}$, respectively. In order to convert moving coordinates into fixed coordinates we have to apply the coordinate transformation that maps \hat{E}^3 onto E^3 . Using homogeneous coordinates, this coordinate transformation can be represented by a 4×4 matrix of the form

$$M = \left[\begin{array}{c|ccc} m_{0,0} & 0 & 0 & 0 \\ m_{1,0} & m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,0} & m_{3,1} & m_{3,2} & m_{3,3} \end{array} \right], \text{ with } m_{0,0} \neq 0, \quad (29.1)$$

such that $\hat{\mathbf{p}} \mapsto \mathbf{p} = M\hat{\mathbf{p}}$. The homogeneous resp. Cartesian coordinate vectors

$$\mathbf{v} = M(1, 0, 0, 0)^\top = (m_{0,0}, m_{1,0}, m_{2,0}, m_{3,0})^\top \quad \text{resp.} \quad \underline{\mathbf{v}} = \begin{pmatrix} m_{1,0} & m_{2,0} & m_{3,0} \\ m_{0,0} & m_{0,0} & m_{0,0} \end{pmatrix}^\top \quad (29.2)$$

describe the *position of the origin* in \hat{E}^3 with respect to the fixed coordinate system E^3 . The 3×3 matrix \underline{R} ,

$$\underline{R} = \frac{1}{m_{0,0}} \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{bmatrix}, \quad (29.3)$$

describes the *orientation* of the moving coordinate system \hat{E}^3 . It is a *special orthogonal matrix*. That is, it satisfies the orthogonality condition $\underline{R}\underline{R}^\top = I$ where I denotes the 3×3 identity matrix, and $\det(\underline{R}) = 1$.

If the matrix $M = M(t)$ depends on the time t , where t varies in some interval $[t_0, t_1]$, then we speak of a *rigid body motion*, cf. Figure 29.1, left. For any point $\hat{\mathbf{p}} \in \hat{E}^3$ of the moving system \hat{E}^3 we obtain its trajectory from

$$\hat{E}^3 \times [t_0, t_1] \rightarrow E^3 : \quad (\hat{\mathbf{p}}, t) \mapsto \mathbf{p}(t) = M(t)\hat{\mathbf{p}} \quad (29.4)$$

where $M(t)$ is of the form (29.1) with time-dependent components $m_{i,j}(t)$.

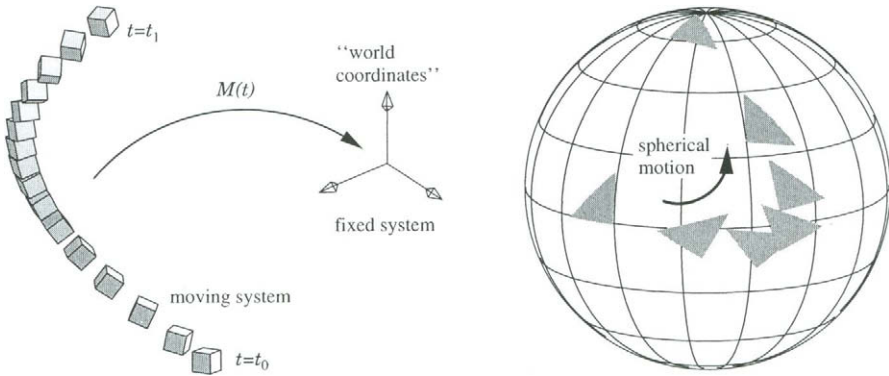


Figure 29.1. Spatial motion of a rigid body (left) and associated spherical motion (right).

Obviously, the trajectory of the origin in homogeneous and Cartesian coordinates is given by $\mathbf{v}(t)$ and $\underline{\mathbf{v}}(t)$, see (29.2).

One may also describe the motion directly in Cartesian coordinates, which leads to

$$\hat{E}^3 \times [t_0, t_1] \rightarrow E^3 : \quad (\hat{\mathbf{p}}, t) \mapsto \underline{\mathbf{v}}(t) + \underline{R}(t)\hat{\mathbf{p}} \quad (29.5)$$

where $\underline{\mathbf{v}}(t)$ are the Cartesian coordinates of the trajectory of the origin (29.2).

The associated rotational (or spherical) part of the motion is described by the special orthogonal matrix $\underline{R} = \underline{R}(t)$, see (29.3). If the trajectory of the origin is replaced with the null vector, $\underline{\mathbf{v}}(t) \equiv (0, 0, 0)^\top$, then the trajectory $\mathbf{p}(t)$ of any point $\hat{\mathbf{p}}$ lies on a sphere of radius $\|\hat{\mathbf{p}}\|$, centered at the origin. Consequently, the rotational part $\underline{R}(t)$ of $M(t)$ describes an intrinsic motion of the unit sphere, see Figure 29.1, right. It will be called the *associated spherical motion* of the rigid body motion $M(t)$. In the figure, it is visualized by several positions of a moving triangle on the unit sphere.

29.2.3. Euler parameters

When designing motions we encounter the problem that the rotational part $\underline{R}(t)$ of $M(t)$ has to satisfy the orthogonality conditions. Consequently it is not possible to simply prescribe the functions $m_{i,j}(t)$ since the resulting motion would in general not preserve the rigidity of the object; it would not be Euclidean. In order to resolve this problem, we will describe \underline{R} by a set of independent parameters. There exist a number of different approaches. A well-known set of parameters is based on a classical result of Euler: any special orthogonal 3×3 matrix \underline{R} can be written as

$$\underline{R} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \quad (29.6)$$

where the q_i satisfy

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1. \quad (29.7)$$

The 4 parameters $\mathcal{Q}^0 = (q_0, q_1, q_2, q_3)$ are called the (normalized) “Euler parameters”. They should not be confused with Eulerian angles, which are also often used in spatial kinematics! Note that “antipodal” Euler parameters $\pm \mathcal{Q}^0$ are correspond to the same rotation matrix \underline{R} . If we further denote

$$q_0 = \cos \frac{\phi}{2} \quad \text{and} \quad \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \sin \frac{\phi}{2} \vec{\mathbf{r}} \quad (29.8)$$

with a unit vector $\vec{\mathbf{r}}$, we may give a simple geometric interpretation of these parameters; the spherical displacement described by \underline{R} is a rotation with angle ϕ about the axis spanned by $\vec{\mathbf{r}}$, cf. Figure 29.2.

Given a rigid body motion $M = M(t)$, there are various ways to compute its normalized Euler parameters. By comparing (29.1) and (29.6) we obtain the relations

$$\begin{aligned} q_0 : q_1 : q_2 : q_3 &= m_{0,0} + m_{1,1} + m_{2,2} + m_{3,3} : m_{3,2} - m_{2,3} : m_{1,3} - m_{3,1} : m_{2,1} - m_{1,2} \\ &= m_{3,2} - m_{2,3} : m_{0,0} + m_{1,1} - m_{2,2} - m_{3,3} : m_{2,1} + m_{1,2} : m_{1,3} + m_{3,1} \\ &= m_{1,3} - m_{3,1} : m_{2,1} + m_{1,2} : m_{0,0} - m_{1,1} + m_{2,2} - m_{3,3} : m_{3,2} + m_{2,3} \\ &= m_{2,1} - m_{1,2} : m_{1,3} + m_{3,1} : m_{3,2} + m_{2,3} : m_{0,0} - m_{1,1} - m_{2,2} + m_{3,3}, \end{aligned} \quad (29.9)$$

see [51]. At least one of these equations gives a result different from $0 : 0 : 0 : 0$. which may then be used, along with (29.7), to determine the normalized Euler parameters.

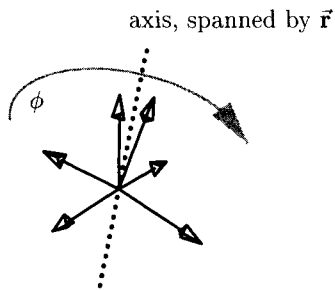


Figure 29.2. The geometric meaning of the Euler parameters. ϕ : angle of rotation, \vec{r} : unit direction vector of the axis.

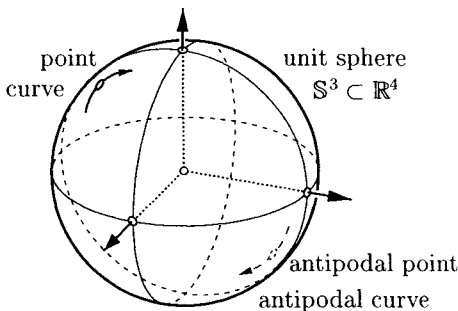


Figure 29.3. The kinematical mapping identifies a spherical motion with a pair of antipodal curves on the 4D unit sphere.

29.2.4. The kinematical mapping

Examining equations (29.6) and (29.9) we notice that there is a birational transformation which maps each rotation matrix onto two antipodal vectors $\pm Q^0 = \pm(q_0, q_1, q_2, q_3)$ of normalized Euler parameters, and vice versa. This transformation is called the *kinematical mapping* of spherical kinematics.

If we identify Q^0 with a point in a four dimensional image space, the kinematical mapping defines a correspondence between the 3D rotations and the pairs of antipodal points on the 4D unit sphere $S^3 \subset R^4$. Furthermore, we may identify a spherical *motion* with a set of two antipodal *curves* on the unit sphere S^3 . This property is schematically illustrated in Figure 29.3. If no ambiguity is to be expected, then both the mapping from the set of 3D rotations to the unit sphere with identified antipodal points, and its inverse, will be referred to as the the kinematical mapping.

29.3. QUATERNIONS

Quaternions are a powerful tool to describe 3D rotations in spherical kinematics. In the following we will introduce the basic concepts of quaternion calculus and explain their relationship with the kinematical mapping.

29.3.1. Fundamentals

For any quadruple of real numbers q_0, \dots, q_3 , we call the pair

$$Q = [q_0, \vec{q}] = [\underbrace{q_0}_{\text{real part}}, \underbrace{(q_1, q_2, q_3)^T}_{\text{vector part}}] \tag{29.10}$$

of the scalar q_0 and the vector $\vec{q} = (q_1, q_2, q_3)^T$ a *quaternion*. The quaternion $\bar{Q} = [q_0, -\vec{q}]$ is called the conjugate quaternion of Q . Let us further define two operations that act on

the set of all quaternions \mathbb{H} ,

$$\begin{aligned} \mathcal{Q} + \mathcal{R} &= [q_0, \vec{q}] + [r_0, \vec{r}] = [q_0 + r_0, \vec{q} + \vec{r}], \\ \mathcal{Q} * \mathcal{R} &= [q_0, \vec{q}] * [r_0, \vec{r}] = \left[\underbrace{q_0 r_0 - \vec{q} \cdot \vec{r}}_{\text{real part}}, \underbrace{q_0 \vec{r} + r_0 \vec{q} + \vec{q} \times \vec{r}}_{\text{vector part}} \right]. \end{aligned} \quad (29.11)$$

With these operations, the set of quaternions \mathbb{H} forms a skew field (Hamilton, 1840). Quaternions with

$$\mathcal{Q} * \bar{\mathcal{Q}} = [q_0^2 + q_1^2 + q_2^2 + q_3^2, (0, 0, 0)] = [1, (0, 0, 0)] \quad (29.12)$$

are called *unit quaternions*, they are marked with \mathcal{Q}^0 . (Compare with the definition of normalized Euler parameters, (29.7).) Quaternions with vanishing scalar part $[0, \vec{p}]$ will be identified with vectors in \mathbb{R}^3 . We may express the usual scalar and cross product of two vectors \vec{q}, \vec{r} in terms of the quaternion multiplication

$$\begin{aligned} [\vec{q} \cdot \vec{r}, (0, 0, 0)] &= -\frac{1}{2} ([0, \vec{q}] * [0, \vec{r}] + [0, \vec{r}] * [0, \vec{q}]), \\ [0, \vec{q} \times \vec{r}] &= \frac{1}{2} ([0, \vec{q}] * [0, \vec{r}] - [0, \vec{r}] * [0, \vec{q}]). \end{aligned} \quad (29.13)$$

Now we consider the quaternion product

$$\vec{p} \rightarrow \vec{p}' = \underbrace{\bar{\mathcal{Q}}^0 * [0, \vec{p}] * \mathcal{Q}^0}_{=\mathcal{X}}, \quad (29.14)$$

where \mathcal{Q}^0 is a unit quaternion, resulting in a vector-type quaternion $[0, \vec{p}'] \sim \vec{p}'$. In fact, this product can be shown to satisfy the condition

$$\overline{\mathcal{Q} * [0, \vec{p}] * \mathcal{Q}} = -\bar{\mathcal{Q}} * [0, \vec{p}] * \mathcal{Q}, \quad (29.15)$$

hence $\bar{\mathcal{X}} = -\mathcal{X}$, which characterizes the vector-type quaternions. With the help of the relationships (29.13) it can easily be shown that the mapping $\vec{p} \mapsto \vec{p}'$ preserves both products between any two vectors \vec{p} and \vec{r} . Hence, the mapping (29.14) can equivalently be described as

$$\vec{p} \rightarrow \vec{p}' = \underline{U} \vec{p}, \quad (29.16)$$

where U is a special orthogonal matrix, depending on the four components of the quaternion \mathcal{Q}^0 . A short calculation indeed confirms that *the matrix U is the special orthogonal matrix with the normalized Euler parameters \mathcal{Q}^0* , see (29.6). Moreover, the *composition of rotations $\underline{U} = \underline{U}_1 \cdot \underline{U}_2$ corresponds to the multiplication of quaternions $\mathcal{Q}^0 = \mathcal{Q}_2^0 * \mathcal{Q}_1^0$* .

29.3.2. Homogeneous quaternions and the kinematical mapping

We now rewrite equation (29.14) in order to allow the use of non-normalized quaternions, by switching to a homogeneous representation. Firstly we note that (29.14) is equivalent to

$$[1, \vec{p}] \rightarrow [1, \vec{p}'] = \bar{\mathcal{Q}}^0 * [1, \vec{p}] * \mathcal{Q}^0$$

In addition, we identify the homogeneous coordinates of a point \mathbf{p} with the quaternion $[p_0, (p_1, p_2, p_3)]$. This leads us to the homogeneous quaternion representation of a rotation (or spherical displacement),

$$\hat{\mathbf{p}} \mapsto \mathbf{p} = \overline{\mathcal{Q}} * \hat{\mathbf{p}} * \mathcal{Q}, \tag{29.17}$$

where \mathbf{p} and $\hat{\mathbf{p}}$ are the homogeneous coordinates of a point with respect to fixed and moving coordinate system, respectively, and \mathcal{Q} is a quaternion. The associated unit quaternions $\pm \mathcal{Q}^0 = \|\mathcal{Q}\|^{-1} \mathcal{Q}$, where $\|\mathcal{Q}\| = \sqrt{\mathcal{Q} * \overline{\mathcal{Q}}}$, consist of the Euler parameters according to (29.8).

Similar to the homogeneous coordinates of a point \mathbf{p} , the quaternion \mathcal{Q} in (29.17) can be considered a homogeneous representation of the rotation. Again, as in the point case, linearly dependent quaternions describe the same rotation. However, the normalization (29.12) defines a different underlying geometric structure than in the point case, where normalized coordinates were characterized by $p_0 = 1$. In the quaternion space, the geometric structure is that of a 3-dimensional elliptic space, see Chapter 3, [2]. The unit quaternion sphere $\mathbb{S}^3 \subset \mathbb{R}^4$ with identified pairs of antipodal points is the standard model of this geometry. For our applications, it is more appropriate to use homogeneous coordinates for points in 3-dimensional elliptic (i.e. quaternion) space. These coordinates will be called *homogeneous quaternion coordinates* of rotations.

29.3.3. Summary: homogeneous quaternion coordinates for 3D rotations

The kinematical mapping maps a point $\mathcal{Q} \neq [0, (0, 0, 0)]$ in 3-dimensional elliptic space, described by homogeneous quaternion coordinates, to the special orthogonal matrix

$$\underline{U}(\mathcal{Q}) = \frac{1}{u_0(\mathcal{Q})} U(\mathcal{Q}), \tag{29.18}$$

where

$$U(\mathcal{Q}) = (u_{i,j})_{i,j=1,2,3} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{29.19}$$

and $u_0(\mathcal{Q}) = q_0^2 + q_1^2 + q_2^2 + q_3^2$. Any point in elliptic 3-space \mathcal{Q} corresponds uniquely to a rotation (or spherical displacement) \underline{U} . Every curve in homogeneous quaternion coordinates $\mathcal{Q}(t)$ can be identified with an intrinsic spherical motion $\underline{U}(t)$. This kinematical mapping is birational. For more information on quaternions, kinematic mappings and their application we refer the reader to [2,3,9,31,51].

29.4. MOTION DESIGN USING CURVES ON \mathbb{S}^3

As we have seen in the previous sections, spherical motions are equivalent with curves in elliptic 3-space. The standard model of elliptic 3-space is the unit quaternion sphere \mathbb{S}^3 in \mathbb{R}^4 with identified pairs of antipodal points. This section discusses some methods that have been proposed in the literature for designing spherical motions with the help of curves on \mathbb{S}^3 .

29.4.1. Slerping

The algorithm of de Casteljau is based on iterated linear interpolation, see Section 4.2.3 of Chapter 4. Though conceptually simple, it gives a very effective tool for curve design that is numerically stable and easy to implement. As a simple approach to curve design on \mathbb{S}^3 one may translate de Casteljau’s algorithm into the geometry of a sphere, in order to produce spherical Bézier-type curves. This is achieved by replacing the line segment connecting two points in Euclidean space with the great circular arc (the geodesic) between two points on a sphere.

More precisely, consider two points \mathcal{B}_0 and \mathcal{B}_1 on the unit quaternion sphere \mathbb{S}^3 . We define the point $\mathcal{B}_0^1(t) = \text{slerp}(\mathcal{B}_0, \mathcal{B}_1, t)$ such that $\mathcal{B}_0^1(t)$ lies on the great circular arc passing through \mathcal{B}_0 and \mathcal{B}_1 and the angles between the coordinate vectors of \mathcal{B}_0 , \mathcal{B}_1 and $\mathcal{B}_0^1(t)$ satisfy

$$\angle(\mathcal{B}_0, \mathcal{B}_0^1(t)) : \angle(\mathcal{B}_0^1(t), \mathcal{B}_1) = t : (1 - t). \tag{29.20}$$

As t varies from 0 to 1, the point $\mathcal{B}_0^1(t)$ traces a great circular arc from \mathcal{B}_0 to \mathcal{B}_1 . Based on this spherical linear interpolation (‘slerp’) we are now able to define a spherical version of de Casteljau’s algorithm as illustrated in Figure 29.4.

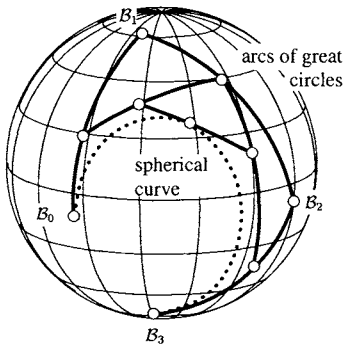


Figure 29.4. Spherical de Casteljau algorithm. Based on a spherical control polygon $\mathcal{B}_0, \dots, \mathcal{B}_n$, repeated spherical linear interpolation results in a spherical curve.

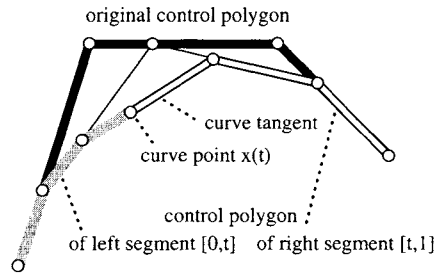


Figure 29.5. Subdivision and tangent property of Bézier curves, cf. Chapter 4. Both properties are not valid for the spherical de Casteljau algorithm.

This technique is commonly referred to as ‘slerping’. It has been introduced by Shoemake in [46] and has since then been subject of much research, mainly in Computer Graphics, e.g. [34,37].

As a first problem, the point $\mathcal{B}_0^1(t)$ is not unique. While this problem can easily be resolved, for example, by restricting \mathcal{B}_0^1 to the shorter arc connecting \mathcal{B}_0 with \mathcal{B}_1 , there are more involved problems that are caused by the subtle differences between elliptic and Euclidean geometry.

29.4.2. Problems of slerp

Two fundamental properties of de Casteljau's algorithm are the subdivision property, and – closely related to it – the fact that the points obtained in last step of the algorithm span the tangent of the resulting Bézier curve. See Sections 4.2.3 and 4.2.5 of Chapter 4 for details. Unfortunately, both properties are lost by the spherical version of the algorithm. The lack of these fundamental properties has serious consequences, as none of the algorithms derived from the subdivision property can be transferred onto the sphere. In particular, C^2 joints are difficult to construct, as the standard Bézier-based construction (leading directly to B-splines, see [16, Section 4.1.2]) is closely related to extrapolation of a curve via de Casteljau's algorithm, cf. [34]. Moreover, it is not possible to use the efficient subdivision-based rendering methods for Bézier curves in the spherical situation.

In addition to these missing fundamental properties, it is very complicated to analyze the curves (and the resulting spherical motions) which result from slerp. Even for relatively low degrees, the parametric representations are rather involved, and it is therefore difficult to apply standard tools from analysis and differential geometry.

Finally, the interpolation problem for slerp Bézier curves leads to a non-linear system of equations; only approximate solutions can be found. This is clearly a serious disadvantage, as interpolation is one of the basic techniques for curve design.

29.4.3. Other approaches

Alternative algorithms for constructing smooth unit quaternion splines have been proposed by various authors, see [1,26–28,42] and elsewhere. For instance, these algorithms are based on the cumulative form of a Bézier or B-spline curve, or on blending techniques for spherical curves. In a more general setting, Park and Ravani [36] have studied Bézier curves on Riemannian manifolds.

As an alternative to generalizing the de Casteljau algorithm it is also possible to generalize the associated subdivision schemes to the spherical case. This idea has stimulated research on non-linear corner cutting algorithms on Riemannian manifolds. For instance, in the case of cubic Bézier curves (which corresponds to the Lane-Riesenfeld algorithm), a thorough analysis has been given by Noakes [35], showing that the limit curve is differentiable, and its derivative Lipschitz. Note that non-linear corner cutting, although conceptually simple, requires rather involved mathematical tools, both for generating and for analyzing motion trajectories.

A detailed study of computational techniques for motion design can be found in the survey article by Röschel [45], providing many further references.

29.4.4. Motion design – desired features

We conclude this section by listing a few features which should be provided by algorithms for motion design.

- The evaluation scheme should be simple and efficient, preferably without involving non-rational functions. Motion design algorithms should be easy to implement, robust and numerically stable. The resulting motion splines should at least exhibit symmetry and subdivision properties.
- Motion trajectories should have a simple representation which follows a generally accepted standard in Computer Aided Design. If possible, trajectories should be repre-

sented as NURBS curves.

- Conditions for interpolation of given data (positions, velocities etc.) and for smooth joints should be easily to formulate and computationally efficient (preferably linear conditions).
- Motion design algorithms (e.g. via interpolation techniques) should be invariant with respect to the choice of the fixed coordinate system (world coordinates), and with respect to the choice of the orientation of the moving coordinate system. That is, coordinate transformations and interpolation algorithms should commute. An additional invariance with respect to the choice of the origin of the moving coordinate system is not really useful, as the origin will mostly have a special meaning in applications, such as the center of gravity or the ‘tool center point’ (TCP).

None of the approaches we have examined so far satisfies all of these properties. In particular we note that point trajectories generated by slerping algorithms are non-rational and therefore do not comply with the industrial NURBS standard (see Chapter 5). In the remainder of this chapter we present an approach based on the kinematical mapping which produces motions generating point trajectories in NURBS form.

29.5. SPHERICAL RATIONAL MOTIONS

The kinematical mapping (29.18) can be used in order to apply the Bézier and B-spline techniques to spherical motions, following the ideas in [10,38]. For instance, consider a rational Bézier curve of degree n in elliptic 3-space,

$$\mathcal{Q}(t) = \sum_{i=0}^n B_i(t) \mathcal{B}_i, \quad t \in [0, 1], \quad (29.21)$$

with the Bernstein polynomials $B_i(t) = \binom{n}{i} t^i (1-t)^{n-i}$. Its control polygon consists of the *control points* $\mathcal{B}_i = [b_{i,0}, (b_{i,1}, b_{i,2}, b_{i,3})^\top] \in \mathbb{H}$ and the *Farin points*

$$\mathcal{F}_{i,i+1} = \mathcal{B}_i + \mathcal{B}_{i+1}, \quad (29.22)$$

see Figure 29.6, left. We use homogeneous coordinates to represent these points. Consequently, the Farin points (also called *weight* or *frame points*) are located on the edges of the control polygon; they represent the weight ratio of neighbouring control points. For further information the reader should consult, e.g., [7,11], and Chapter 5. The combination of control and Farin points provides a projectively invariant description of a rational Bézier or B-spline curve. It is also invariant in the sense of elliptic geometry, as elliptic transformations are special cases of projective mappings.

Now we apply the kinematical mapping, both to the rational Bézier curve (29.21) and to its control and Farin points. Firstly, consider the image of the linear Bézier curve,

$$\mathcal{Q}^{(1)}(t) = (1-t) \mathcal{B}_0 + t \mathcal{B}_1, \quad t \in [0, 1]. \quad (29.23)$$

It turns out to be a rotation of the unit sphere with a constant axis. More precisely, the components of the rotation matrix $\underline{U}^{(1)}(t) = \underline{U}(\mathcal{Q}^{(1)}(t))$ are quadratic rational functions, cf. (29.18). The trajectory of any point $\hat{\mathbf{p}}$ of the moving system is simply a circular arc, which is described as a rational quadratic curve $\underline{U}^{(1)}(t) \hat{\mathbf{p}}$.

Next we consider the image of a rational Bézier curve (29.21) of degree n , see Figure 29.6. It is a *spherical rational motion* of degree $2n$, as the components of $\underline{U}(t) = \underline{U}(\underline{Q}(t))$ are rational functions of degree $2n$. The trajectory of any point $\hat{\underline{p}}$ of the moving system is the rational curve $\underline{U}(t)\hat{\underline{p}}$ of degree $2n$. As an example, Fig. 29.6 shows a cubic rational Bézier curve and its image under the kinematical mapping.

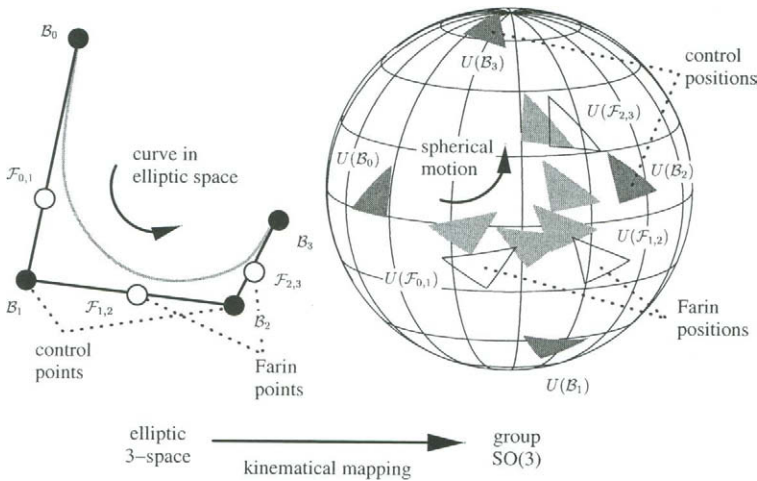


Figure 29.6. The image of a rational Bézier curve in elliptic 3-space under the kinematical mapping is a spherical rational motion of degree $2n$. Applying the mapping to the Bézier control polygon gives a control structure for spherical rational motions.

More generally, the kinematical mapping could be applied to a rational B-spline curve of degree d in elliptic 3-space, resulting from (29.21) by replacing the Bernstein polynomials $B_i(t)$ with B-splines defined over a suitable knot sequence. This produces a spherical rational spline motion $\underline{U}(\underline{Q}(t))$ of degree $2d$. The computation of the B-spline form involves the evaluation of products of B-splines, see [33]. If the preimage curve has single interior knots, then both the preimage curve and the spherical rational spline motion are C^{d-1} . Hence, the inner knots of the spline functions $u_0(\underline{Q}(t))$ and $\underline{U}(\underline{Q}(t))$ have at least multiplicity $d + 1$. Computational techniques for rational spline motions in B-spline form, including a formula for their Bézier segments, have been discussed in [23].

By applying the kinematical mapping to the control and Farin points of the rational Bézier (or B-spline) curve we obtain an *intrinsic control structure* for spherical rational (spline) motions. This control structure has been introduced by Pottmann [38]. It is obtained by applying the kinematical mapping to the control and Farin points of the rational curve, leading to *control positions* and *Farin positions*. The edges of the control polygon are mapped to *rotations* of the unit sphere, joining two neighbouring control positions and the corresponding Farin position. In fact, the edges can be seen as as linear

rational Bézier curves $B_0^1(t)\mathcal{B}_{i-1} + B_1^1(t)\mathcal{B}_i$ ($i = 1, \dots, n$); the Farin point is associated with the parameter value $t = \frac{1}{2}$.

The intrinsic control structure is suitable for interactive motion design. An example is shown in Figure 29.7. The two spherical rational Bézier motions are obtained from the motion of Figure 29.6 by changing the first control position $U(\mathcal{B}_1)$ (left) and by modifying the weight of the second control position $U(\mathcal{B}_2)$ (right), leading to modified Farin points $\mathcal{F}_{1,2}$ and $\mathcal{F}_{2,3}$.

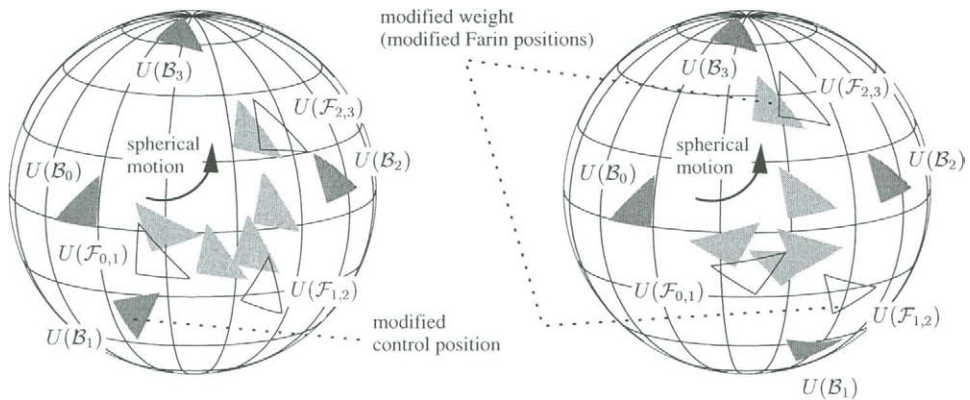


Figure 29.7. Interactive motion design using the intrinsic control structure of a spherical rational motion. Modification of a control position (left) and of a weight (right), compare with Figure 29.6.

A similar technique can be used to generate an intrinsic control structure for *spherical rational curves*, using the so-called generalized stereographic projection (see [6,38], cf. Chapter 31. In fact, the generalized stereographic projection can be derived by restricting the kinematical mapping (29.18) to the trajectory of a single point.

In principle, the properties of the Bézier and B-spline control points (convex hull property etc.) can be transferred to the intrinsic control structure of a spherical rational motion, with the help of the kinematical mapping. However, it is difficult to give a useful geometric interpretation, as the image of a volume in elliptic space is somewhat difficult to visualize on the sphere.

Consider a spherical rational motion $\underline{R}(t) = \frac{1}{r_0(t)}R(t)$ of degree m . That is, the denominator $r_0(t)$ and the 9 components of $R(t)$ are polynomials of maximum degree m , and the matrices $\underline{R}(t)$ are special orthogonal matrices for all t . Then, the trajectory $\underline{R}(t)\underline{\hat{p}}$ of any point $\underline{\hat{p}}$ is a spherical rational curve of degree m , on the sphere (centered at the origin) with radius $\|\underline{\hat{p}}\|$. Clearly, by applying the kinematical mapping to a rational curve of degree n we obtain a spherical rational motion $\underline{R}(t)$ of degree $m = 2n$. Conversely, one may ask whether any motion can be constructed that way.

Proposition [19]. *If the denominator $r_0(t)$ and the 9 components of $R(t)$ do not share polynomial factors, i.e.*

$$\gcd\{r_0(t), r_{1,1}(t), r_{1,2}(t), r_{1,3}(t), r_{2,1}(t), r_{2,2}(t), r_{2,3}(t), r_{3,1}(t), r_{3,2}(t), r_{3,3}(t)\} = 1 \quad (29.24)$$

holds in the polynomial ring $\mathbb{R}[t]$, then m is even, and the spherical rational motion $\underline{R}(t) = \frac{1}{r_0(t)}R(t)$ can be generated by applying the kinematical mapping $\underline{U}(\cdot)$ to a rational curve $\mathcal{Q}(t)$ of degree $m/2$.

The proof consists of two parts. Firstly it is shown that any spherical rational motion corresponds to a rational curve in elliptic 3-space, as it has rational Euler parameters $\mathcal{Q}(t)$. This can be concluded from (29.9), expressing the Euler parameters as rational functions of the matrix components. Secondly, it can be shown that any common factor of the matrix components is a common factor of the corresponding Euler parameters. This observation leads to the degree bound of the proposition. For the details of the proof the reader is referred to [19] or [41, Chapter 8].

Summing up, spherical rational motions can be generated by applying the kinematical mapping to rational curves in elliptic 3-space. According to the proposition, this construction produces motions having the minimum possible degree. In addition, the Bézier resp. B-spline control structure of rational spline curves can be translated to the kinematical setting.

29.6. SPATIAL RATIONAL MOTIONS

The results on spherical rational spline motions can be extended to spatial ones, by combining them with rational trajectories of the origin. This section discusses the construction and classification of rational spline motions, and the use of control polygons and control structures.

29.6.1. Construction

Recall from Section 29.2.2 that the motion of a rigid body is described by a time-dependent transformation $\hat{\mathbf{p}} \mapsto \mathbf{p}(t) = M(t) \hat{\mathbf{p}}$ of the form (29.1), mapping any point $\hat{\mathbf{p}}$ of the moving system to a point on its trajectory $\hat{\mathbf{p}}(t)$. We use homogeneous coordinates to represent both the points and the transformation.

If the components of the matrix $M(t)$ are rational (spline) functions of degree m , then the corresponding rigid body motion is called a *rational (spline) motion of degree n* . All trajectories are rational (spline) curves of degree m , see Figure 29.8 for an example. Consequently, as the trajectories can be described as NURBS curves, rational motions comply with industrial CAD standards.

Consider a transformation matrix of the form

$$M(t) = \left[\begin{array}{c|ccc} v_0^*(t) u_0(t) & 0 & 0 & 0 \\ v_1(t) & & & \\ v_2(t) & v_0^*(t) U(t) & & \\ v_3(t) & & & \end{array} \right]. \quad (29.25)$$

The associated spherical motion $\underline{U}(t) = [1/u_0(t)] U(t)$ has been constructed by applying the kinematical mapping (29.18) to a rational spline curve $\mathcal{Q}(t)$ of degree k in elliptic

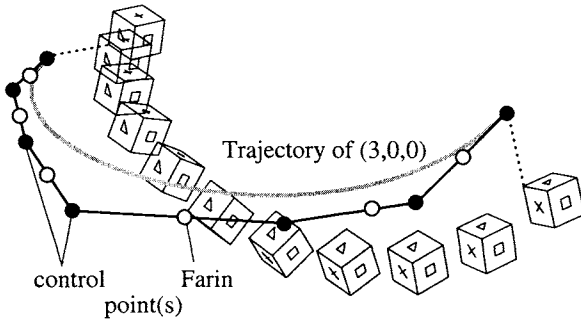


Figure 29.8. Example - a rational motion of degree 6. The motion is visualized by the moving unit cube. In addition, the trajectory of a point, along with its rational Bézier control polygon consisting of control points \mathbf{b}_i and Farin points $\mathbf{f}_{i,i+1}$ is shown.

3-space; the piecewise polynomials $v_0^*(t)$ of degree p and $v_1(t), v_2(t), v_3(t)$ of degree q are arbitrary. Then, the motion (29.25) describes a spatial rational spline motion of degree $m = \max(q, p + 2k)$.

The origin of the moving space generates the trajectory

$$M(t) (1, 0, 0, 0)^T = (v_0^*(t) u_0(t), v_1(t), v_2(t), v_3(t))^T \tag{29.26}$$

Given a spherical rational spline motion, the spatial rational spline motion (29.25) may combine any trajectory $\mathbf{p}(t) = (p_0(t), p_1(t), p_2(t), p_3(t))^T$ of the origin with it, by choosing $v_0^*(t) = p_0(t)$ and $v_i(t) = u_0(t) p_i(t)$, $i = 1, 2, 3$. Generally, by combining a spherical rational spline motion of degree $n = 2k$ with a degree q rational spline curve one obtains a spatial rational spline motion of degree $m = 2k + q$. In applications, however, the degree should often be kept as small as possible. This can be achieved by choosing trajectories $\mathbf{v}(t)$ whose denominator $v_0(t) = v_0^*(t)u_0(t)$ equals the denominator u_0 of the associated spherical rational spline motion, i.e., by choosing $u_0 \equiv 1$.

Similar to the previous section, it is a natural question to ask whether any spatial rational (Bézier) motion can be generated with the help of formula (29.25).

Theorem [19]. *Any spatial rational motion of degree m is obtained from (29.25) by applying the kinematical mapping to a rational (Bézier) curve $\mathcal{Q}(t)$ of degree k in elliptic 3-space, where the degree k satisfies $0 \leq k \leq \lfloor m/2 \rfloor$, leading to the associated spherical rational motion $\underline{U}(\mathcal{Q}(t)) = [1/u_0(\mathcal{Q}(t))] U(\mathcal{Q}(t))$ of degree $2k$, and choosing polynomials $v_0^*(t)$ of degree $m - 2k$, and $v_1(t), v_2(t), v_3(t)$ of degree m .*

This result follows immediately from the previous proposition. Consequently, there are $\lfloor m/2 \rfloor + 1$ different classes of rational motions of degree m , corresponding to the degree $2k$ of the associated spherical rational motion.

29.6.2. Special cases

Rational motions of degree $m \leq 4$ have thoroughly been studied in the theory of space kinematics. The simplest non-trivial example, given by quadratic rational motions with $k = 1$, can be traced back to Darboux [3,5]. In the general situation, these motions are obtained by composing a planar elliptic motion with a harmonic oscillation. The elliptic motion is a special trochoidal motion: a small circle (radius r) rolls within a big circle (radius R), where $r : R = 1 : 2$, see Figure 29.9. All trajectories are ellipses, except for the points on the rim of the small circle, which trace diameters of the big circle. This motion is extended into 3-space, where it becomes the rolling of two circular cylinders. By adding a synchronized harmonic oscillation in the direction of the cylinders' axes we obtain a *Darboux motion*, see again Figure 29.9. As the elliptic motion and the harmonic oscillation have equal frequencies, all trajectories are still ellipses. Darboux motions can be shown to be the most general truly spatial motions which generate planar point trajectories for all points of the moving system.

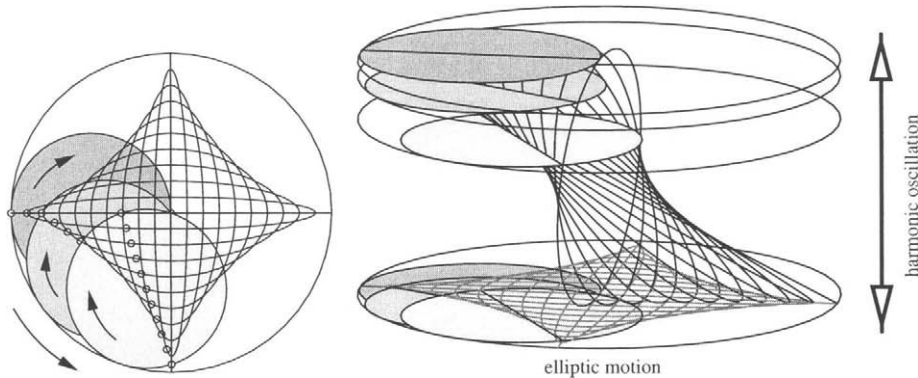


Figure 29.9. Elliptic motion (left) and Darboux motion (right).

More recently, a thorough geometric analysis of rational motions of degree 3 and 4 has been given by Wunderlich and Röschel [43,52].

29.6.3. Affine control structure

Consider again a spatial rational spline motion of degree n . It is described by a transformation matrix $M(t)$, see (29.1) and (29.25), whose components are piecewise polynomial functions (splines) of degree n . Consequently, one may represent the transformation matrix in B-spline form (or even in Bézier form, in the case of a spatial rational motion),

$$M(t) = \sum_i C_i B_i(t), \quad t \in [a, b], \tag{29.27}$$

with 4×4 control matrices C_i and B-splines $B_i(t)$, defined with over a suitable associated knot sequence. Similar to the transformation matrices $M(t)$, the coefficient matrices

$C_i = (c_{j,k}^{(i)})_{j,k=0,\dots,4}$ satisfy

$$c_{0,1}^{(i)} = c_{0,2}^{(i)} = c_{0,3}^{(i)} = 0, \quad i = 0, \dots, N. \quad (29.28)$$

The orthogonality condition (29.3), however, is generally not satisfied (except for control matrices describing positions, e.g., at the boundaries – if the boundary knots have sufficient multiplicity).

Any point $\hat{\mathbf{p}}$ of the moving system traces a rational B-spline curve,

$$M(t) \hat{\mathbf{p}} = \sum_{i=0}^N [C_i \hat{\mathbf{p}}] B_i(t), \quad (29.29)$$

with control points $C_i \mathbf{p}$ and weights $p_0 c_{0,0}^{(i)}$ ($i = 0, \dots, N$). Alternatively, one may again use Farin points to specify the weight ratios, $(C_{i-1} + C_i) \mathbf{p}$ ($i = 1, \dots, N$).

Consider a moving object \hat{O} , which is described as a bounded set of points in the moving system. Collecting the control and Farin points of the trajectories we obtain the *control positions* $C_i \hat{O}$ and *Farin positions* $(C_{i-1} + C_i) \hat{O}$. Generally, the transformations C_i and $(C_{i-1} + C_i)$ do not preserve the rigidity (i.e., distances and angles) of the object, as the orthogonality condition (29.3) is not satisfied. The control and Farin positions are affine images of the moving object, as the matrices describe affine mappings (preserving ratios and parallelism), due to (29.28). The combination of control and Farin positions is called the *affine control structure*, see [23,47].

An example is shown in Figure 29.10a, b. The spatial rational motion (degree 6) of the moving unit cube (a) has been generated by composing the spherical rational motion of Figure 29.6 with a suitable trajectory of the origin (a rational Bézier curve of degree 6). We chose $v_0^* \equiv 1$ in (29.25), hence the resulting spatial motion has still degree 6. The affine control and Farin positions (b) are obtained by collecting the control and Farin points of the trajectories generated by the points of the moving unit cube.

The affine control structure is not suitable for designing the spherical part of the spatial rational motion. In particular, any change of the shape of the affine control positions, and/or of the associated weights (Farin positions) may entail a violation of the orthogonality condition (29.3). The spherical part should be designed with the intrinsic control structure of spherical rational motions. The affine control structure can be used for designing the *translational part* of the motion, as it is possible to apply arbitrary translations to the affine control positions. This is demonstrated in Figure 29.10c,d, where a translation has been applied to the second control position.

The affine control structure can be used for efficiently generating a bounding volume for the moving object. This fact has potential application to collision detection and avoidance; it can also be used for approximate computation of envelopes. If the weights are positive, then any intermediate position of the moving object is contained within the convex hull of the affine control positions. If $v_0^* \equiv 1$ has been chosen (which will mostly be the case in applications), then the denominator of the spatial rational spline motion is a sum of four squares, see (29.19). Consequently, the weights will mostly be positive; they can always be made non-negative by splitting the rational spline motion into suitable smaller segments.

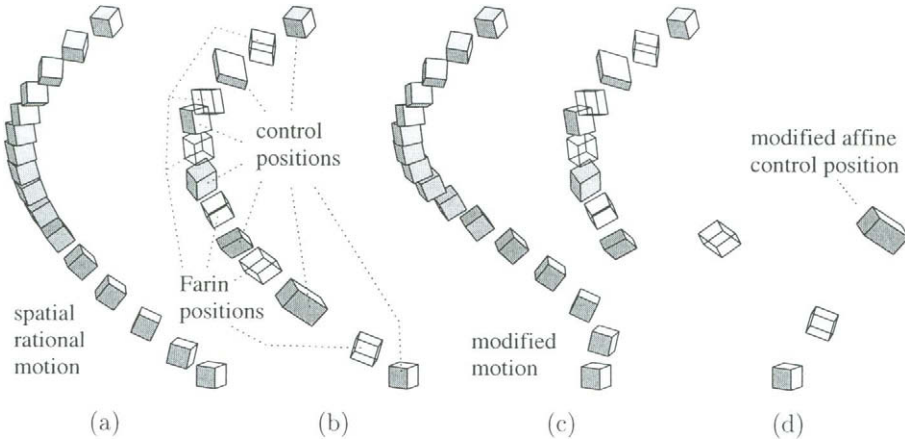


Figure 29.10. Two spatial rational motions (a,c) of degree 6 of a moving unit cube, and their affine control structure (b,d), consisting of control positions (solid) and Farin positions (wireframe). For both motions, the associated spherical motion is that of Figure 29.6.

As an example, we demonstrate the convex hull property of a planar rational motion, see Figure 29.11. Any *planar rational spline motion* of the x_1x_2 -plane can be obtained from (29.25) by applying the kinematical mapping to preimage curves $Q(t)$ in elliptic 3-space with $q_1(t) \equiv q_2(t) \equiv 0$, and choosing $v_3(t) \equiv 0$. For a more detailed geometric discussion of planar rational motions the reader should consult [48]. As observed there, the associated affine control structure consists of *equiform images* of the moving object.

A planar rational motion, along with its (equiform) control and Farin positions is shown in Figure 29.11a. The resulting convex hull gives a bound on the motion of the object. This result can be made more accurate by splitting the motion into smaller segments and generating the convex hull of the resulting control structures, see Figure 29.11b.

The same idea can be applied to spatial rational spline motions. However, the computation of the convex hull in 3D becomes more expensive, and alternative techniques (such as bounding boxes) will be preferred.

29.6.4. Some properties

Spatial rational spline motions provide various desirable features, making them a useful tool in applications.

- The standard rational Bézier and B-spline techniques for CAGD can be applied, providing simple and efficient algorithms for evaluating these motions. Spatial rational spline motions generate point trajectories which can easily be represented in B-spline form, complying with industrial CAD standards. They can be equipped with control structures which are suitable for interactive motion design and for generating convex

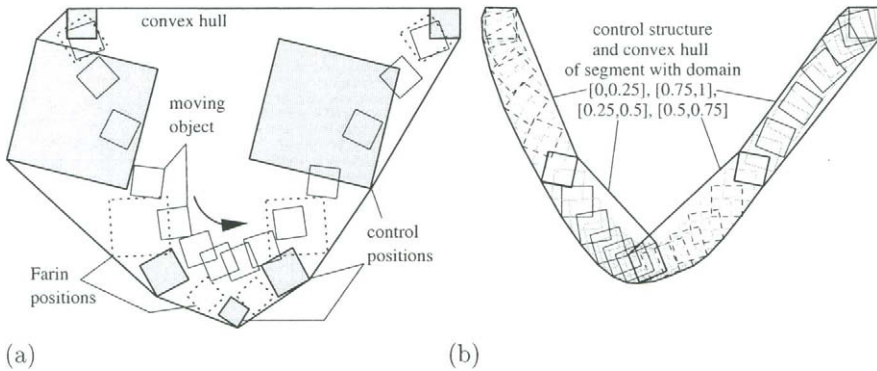


Figure 29.11. Computing the convex hull of a planar rational motion. The motion and its control structure (a), and the convex hulls obtained after splitting the curve into four segments (b).

hulls of moving objects.

- Spatial rational spline motions have a subdivision property, as any subsegment can again be described in rational spline form. Consequently, standard CAGD techniques like hierarchical editing can be used. Moreover, it is possible to generate rational spline motions with arbitrarily high order of differentiability. This was not the case for ‘slerp’ Bézier spline motions [34].
- The class of rational spline motions is invariant with respect to the choice of the coordinate system in both the fixed and the moving system. Moreover, efficient interpolation techniques, generalizing the standard interpolation algorithms for spline curves, can easily be derived, see next section for details. They produce results which are independent of the choice of the fixed coordinate system, and of the choice of the orientation of the moving system [23].

The results of the interpolation algorithm described in the next section depend on the choice of the origin of the moving system. This dependency is often desired in applications, as the origin may have a special geometric meaning (e.g., the center of gravity, tool center point). However, even this dependency can be avoided, using the more sophisticated techniques of [21].

In addition, it is possible to represent trajectories of *moving planes* in rational spline form, and to introduce a *dual control structure* [24,47]. This leads to explicit formulas for envelopes of moving planes [23,47] and, more generally, for envelopes of moving rational developable surfaces (including quadratic cylinders and cones) see [24,47,53]. As an application one may compute the envelope of moving polyhedra, without numerical approximation.

With the help of a kinematical mapping for spatial displacements, a slightly different approach to the design of spatial rational motions has been developed by Ge and Ra-

vani [10]. It is based on the use of dual quaternions (numbers from the ring $\mathbb{H} + \epsilon\mathbb{H}$, where $\epsilon^2 = 0$), see [3]). Following this approach, the motion is described by a sequence of control positions with associated dual weights. One obtains an intrinsic control structure for spatial rational motions, whose “legs” are special Darboux motions (Darboux motions with constant axis; the two cylinders degenerate into a fixed line). The control structure is suitable for interactive motion design. However, this approach is closer to *line trajectories* (i.e., ruled surfaces) than to point trajectories; consequently, it is more difficult to control the trajectory of a specific point, such as the origin of the moving space. Also, the influence of the dual weights is sometimes not very intuitive, and the control structure does not provide a convex hull property. By restricting that approach to spherical control positions one arrives again at the intrinsic control structure for spherical rational motions, see Section 29.5.

29.6.5. Interpolation schemes and applications

Interpolation and approximation of given point data are fundamental techniques for generating curves and surfaces in Computer Aided Geometric Design. This section demonstrates that the standard interpolation schemes can be generalized to the kinematical setting, with the help of spatial rational motions.

Let a sequence of positions $(Pos_i)_{i=1,\dots,N}$ of a moving object be given. Each position is described by a coordinate transformation of the form (29.1) between fixed and moving system. We assume that the origin of the moving coordinate system has a special meaning, such as the center of gravity, tool center point (TCP), etc. The data are to be interpolated with a spatial rational spline motion, see Figure 29.12.

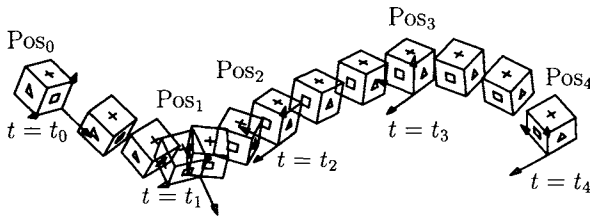


Figure 29.12. Interpolation of 5 given positions of the unit cube with a spatial rational rigid body motion. By browsing through this chapter, you will see an animation of the associated spherical motion in the upper right corners of the odd pages.

In the sequel we give a brief summary of a the interpolation procedure. For further details the reader should consult, e.g., [23].

- 1. Preprocessing.** As the initial step of the interpolation procedure, the given data is converted into quaternion form. More precisely, each of the given positions is described by the Cartesian coordinates \underline{w}_i of the origins, and by the normalized Euler parameters (unit quaternions) \mathcal{R}_i^0 which are associated with the corresponding rotation matrices.

From (29.7) and (29.9) we obtain two solutions, corresponding to a pair of antipodal points on the unit quaternion sphere $\mathbb{S}^3 \subset \mathbb{R}^4$ (III). We pick one of those points (i.e., the sign of the normalized Euler parameters) such that neighbouring points $\mathcal{R}_i^0, \mathcal{R}_{i+1}^0$ on \mathbb{S}^3 belong to one hemisphere, that is the inner product of the corresponding vectors in \mathbb{R}^4

$$\langle \mathcal{R}_i^0, \mathcal{R}_{i+1}^0 \rangle = \frac{1}{2}(\mathcal{R}_i^0 * \overline{\mathcal{R}_{i+1}^0} + \mathcal{R}_{i+1}^0 * \overline{\mathcal{R}_i^0}) \geq 0 \quad (29.30)$$

should be non-negative.

Secondly, we need to associate parameter values t_i with the given positions. Similar to the methods for parameterizing point data (see [16, Section 4.4.1]), these parameters can be estimated from the distances between the given positions. In addition to the distance of the origins, the difference of the orientations should be taken into account. For instance, by generalizing the chordal parameterization, one may choose the differences $t_{i+1} - t_i$ proportional to

$$\|\underline{\mathbf{w}}_{i+1} - \underline{\mathbf{w}}_i\| + \omega \arccos \langle \mathcal{R}_i^0, \mathcal{R}_{i+1}^0 \rangle \quad (29.31)$$

with some weight $\omega > 0$ controlling the influence of the spherical part. Several other possibilities are listed in [21].

- 2. Interpolation of the spherical part.** As the next step, we compute the preimage curve $\mathcal{Q}(t)$ of the kinematical mapping from the interpolation conditions $\mathcal{Q}(t_i) = \mathcal{R}_i^0$. This curve is a rational spline curve of degree d in elliptic 3-space, cf. (29.21). Knots and degree are chosen such that the number of degrees of freedom equals the number of unknowns, where additionally the Schoenberg-Whitney conditions (see Chapter 6) are to be satisfied. For example, one may choose a cubic spline curve with not-a-knot type boundary conditions, producing the knot vector

$$\underbrace{(t_1, \dots, t_1)}_{4\text{-fold knot}}, \underbrace{(t_3, t_4, \dots, t_{N-2})}_{\text{single knots}}, \underbrace{(t_N, \dots, t_N)}_{4\text{-fold knot}} \quad (29.32)$$

The control quaternions \mathcal{B}_i of the preimage curve can be found by solving the resulting banded system of linear equations, see Chapter 6.

Now we generate the spherical part of the interpolating motion by applying the kinematical mapping of spherical kinematics to the preimage spline curve. This results in a spherical rational motion of degree $2d$, described by the spline functions $u_0(t)$ and $U(t)$, see (29.25).

- 3. Translational part.** For the sake of simplicity we may choose $v_0^*(t) \equiv 1$. In order to interpolate the translational parts of the given positions, we have to find spline functions $v_1(t), v_2(t), v_3(t)$ satisfying the interpolation conditions $\underline{\mathbf{v}}(t_i) = \underline{\mathbf{w}}_i$, where

$$\underline{\mathbf{v}}(t) = (v_1(t)/u_0(t), v_2(t)/u_0(t), v_3(t)/u_0(t))^T \quad (29.33)$$

is the trajectory of the origin. It seems to be a natural choice to choose spline functions $v_1(t), v_2(t), v_3(t)$ of degree $2d$ whose knots are those of the spline functions $u_0(t)$ and $U(t)$. This choice, however, leads to an underdetermined system of linear equations, as the translational part of the motion has far more degrees of freedom than

the spherical one. Consequently, additional constraints are needed to pick a unique solution. These may be not-a-knot type conditions at inner knots, enforcing higher order of differentiability for the trajectory of the origin. Alternatively, one may use the additional degrees of freedom for minimizing quadratic ‘energy’ functionals, such as

$$\int_{t_0}^{t_N} \|\dot{\underline{\mathbf{v}}}(t)\|^2 dt \rightarrow \text{Min.} \quad (29.34)$$

See Section 26.5.2 for further information on this technique. In either case, the resulting interpolating spline motion of degree $2d$ is found by solving a system of linear equations.

Various algorithms for interpolation with spline curves can be generalized to the kinematical setting, simply by applying them to the preimage curve of the kinematical mapping, and combining the result with a suitable trajectory of the origin. For instance, a kinematical version of *cubic Hermite splines* has been implemented as part of a commercial robot controller [15]. This leads to rational spline motions whose spherical part has degree 6. Cubic Hermite splines provide some features which are essential in this application, such as real-time capability and certain shape-preserving properties. Compared with traditional techniques, the use of spline curves leads to a substantial reduction of the data volume, and to enhanced programming of robot motions, in particular for the manufacturing of free-form shapes.

Further computational techniques for rational spline motions include the optimization (‘fairing’) of motion segments, to generate spherical motions that minimize (e.g.) the integral of the squared angular acceleration (which can be seen as the analogue of cubic spline curves). Moreover an algorithm for spline motion fitting has been used to reconstruct the motion of the human knee joint from measurement data. See [20,23,25] for additional information.

Although the above interpolation scheme for rational spline motions has many desirable features, it is not fully satisfying from the theoretical point of view, as it lacks what was called ‘invariance with respect to parameterization’ or *parameter invariance* by Röschel [12,45]. If we sample data (positions with associated parameters t_i) from a rational spline motion and apply the interpolation procedure, including the preprocessing step, then it will generally not reproduce the original motion, even if the same spline spaces are used. This is due to the fact that the normalization (29.7) is only valid at the original interpolation nodes, and not everywhere. In order to guarantee the reproduction property one would need to use rational curves on the unit quaternion sphere $\mathbb{S}^3 \in \mathbb{R}^4$. Such curves can be generated with the help of stereographic projection, but then the results depend on the choice of the coordinates, as it is the case for the method described in [18].

In the case of the sphere in 3-space, the *generalized stereographic projection* can be shown to give results which are independent of the chosen system of coordinates, see Chapter 31, [6]. Unfortunately, similar results for spheres in higher dimensions are currently not available. Recently, Gfrerrer [12] has developed a new algorithm for interpolation with rational curves on hyperspheres of arbitrary dimension, producing coordinate-independent results. However, the degree of the resulting spherical motion is about twice as high as the one which would result from the earlier algorithm, and it may happen that no solutions

exist. Furthermore, the generalization of Gferrer's method to rational *spline* curves is still an open problem.

29.6.6. Rational frames and sweeping surfaces

The kinematical version of Hermite interpolation with cubic C^1 splines has also been used to generate highly accurate rational approximation of *rotation minimizing frames*, using spatial rational spline motions of degree 6 [24]. In geometric modeling, the rotation minimizing frame has been introduced by Klok as an alternative to the Frenet frame of a space curve [13,30]. It is useful for sweep surface modeling, as it provides a robust and intuitive way of moving a profile curve along a given 'spine' curve, see [50] for examples.

As an alternative to the approximation of frames, one may also study spatial curves which have an associated rational frame. A spatial rational motion is called a *rational frame* (see Figure 29.13) of a given space curve, if the origin of the moving system travels along that curve, and if additionally the tangent vector of the curve is always parallel to the (say) \hat{x}_1 -axis of the moving system.

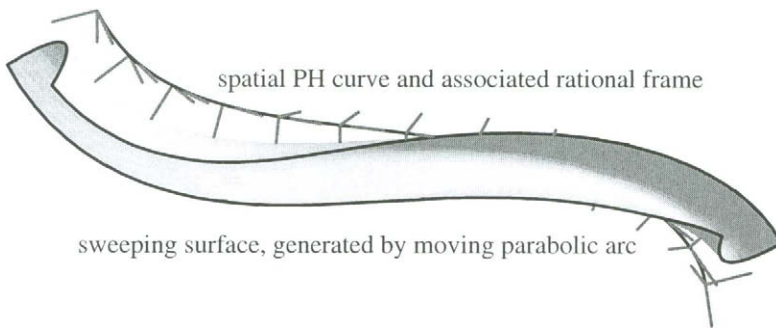


Figure 29.13. Rational frame of a PH curve of degree 7 and a rational sweeping surface.

Note that these frames are closely related to *Pythagorean hodograph (PH) space curves*, see Section 15.2.3 of Chapter 17. In fact, the parametric representation of a polynomial PH curve in 3-space can be generated by integrating the hodograph

$$\dot{\mathbf{x}}(t) = U(\mathcal{Q}(t)) (1, 0, 0)^\top = (q_0^2 + q_1^2 - q_2^2 - q_3^2, 2(q_1 q_2 + q_0 q_3), 2(q_1 q_3 - q_0 q_2))^\top, \quad (29.35)$$

cf. (29.19) and Eq.(10) of Chapter 17, where $\mathcal{Q}(t)$ is an arbitrary preimage curve in elliptic 3-space. For instance, the rational frame in Figure 29.13 has been generated from the spherical motion shown in Figure 29.6. The resulting curves are automatically equipped with rational frames, which can be obtained by combining the trajectory of the origin with the spherical rational motion $\underline{U}(t)$, see (29.18). Recently, even more sophisticated classes of rational space curves have been studied, providing a *rational Frenet frame* or a *rational rotation-minimizing frame*, see [39,40,49].



Spatial rational motions can be used to generate *sweeping surfaces*. These surfaces are generated by moving a fixed profile curve through 3-space, see [8, Preface (written by P. Bézier), Fig 12] for an illustration. Rational sweeping surfaces have been studied in [44,22]. The class of sweeping surfaces can be generalized by allowing simultaneous changes of the moving profile curve. This leads to ‘generalized cylinders’, which have been shown to be a useful tool for the interactive modelling of free-form shapes. Again, the underlying rigid body motion can efficiently be described in rational (B-) spline form, see [4].

29.7. CLOSURE

Based on rational spline techniques and the kinematical mapping, we have shown that computational methods for spatial rigid body motions can be obtained by generalizing the powerful techniques of Computer Aided Geometric Design. We conclude this chapter by listing a few possible topics for further research.

- *Generating optimal motions.* An interesting problem for applications in robotics and NC machining is the efficient generation of energetically or time-optimal motions, taking the robot geometry into account, and their use for robot control. This may help to reduce cycle times in manufacturing, and to increase the lifetime of the machinery. Recently, affine spline motions with piecewise polynomial point trajectories have been used to approximate energetically optimal motions [17]. A related problem is the optimal generation of paths for NC milling, cf. [29].
- *Advanced CAD/CAM interfaces.* Currently, the sophisticated geometry models of CAD are mostly converted to piecewise linear or circular descriptions of tool paths for Numerically Controlled (NC) machining. Here, due to the advancing processor speed, it is now possible to use more advanced geometric models. First attempts in this direction include the use of Pythagorean hodograph curves in NC milling (see Chapter 17), and spline interpolation for robot motion planning [15].
- *Simulation of machining.* This is related to a third challenging problem. Advanced methods for computer-aided simulation of manufacturing processes (e.g. milling) may help to optimize these processes, and to check the quality of the results. For instance, it would be interesting to be able to generate an accurate representation of the surface produced by the cutter of a milling machine, in order to check the quality of the results.

As a promising direction for further research, the advanced techniques for describing in Computer Aided Geometric Design should now be applied to problems from other areas, such as computer-aided manufacturing and numerical simulation in scientific computing. We are convinced that spatial rational spline motions are well suited for this forthcoming task.

REFERENCES

1. A. Barr, B. Currin, S. Gabriel, and J. Hughes. Smooth interpolation of orientations with angular velocity constraints using quaternions. *Computer Graphics*, 26:313–320, 1992. (SIGGRAPH '92).

2. W. Blaschke. *Kinematik und Quaternionen*. Deutscher Verlag der Wissenschaften, Berlin, 1960.
3. O. Bottema and B. Roth, *Theoretical kinematics*. Dover Publications, New York, 1990. (Corr. reprint of the 1979 edition).
4. T.-I. Chang, J.-H. Lee, M.-S. Kim, and S. Hong. Direct manipulation of generalized cylinders based on B-spline motion. *The Visual Computer*, 14:228–239, 1998.
5. G. Darboux. Note III: Sur les mouvements algébriques. In *Leçons de cinématique (G. Kœnigs)*, pages 352–353. Hermann, Paris, 1895.
6. R. Dietz, J. Hoschek, and B. Jüttler. An algebraic approach to curves and surfaces on the sphere and on other quadrics. *Computer Aided Geometric Design*, 10:211–229, 1993.
7. G. Farin. Algorithms for rational Bézier curves. *Computer-Aided Design*, 15:73–77, 1983.
8. G. Farin. *Computer Aided Geometric Design – a practical guide*. Academic Press, Boston, 1993.
9. J. Gallier. *Geometric Methods and Applications*. Springer, New York, 2000.
10. Q. Ge and B. Ravani. Computer aided geometric design of motion interpolants. In *Proc. ASME Design Automation Conf., DE*, 32(2):33–41. ASME, 1991.
11. G. Geise and B. Jüttler. A geometrical approach to curvature continuous joints of rational curves. *Computer Aided Geometric Design*, 10:109–122, 1993.
12. A. Gfrerrer. Rational interpolation on a hypersphere. *Computer Aided Geometric Design*, 16:21–27, 1999.
13. H. Guggenheimer. Computing frames along a trajectory. *Computer Aided Geometric Design*, 6:77–78, 1989.
14. A. Hanson. Visualizing quaternions. Course at SIGGRAPH 2000, New Orleans, 2000.
15. T. Horsch and B. Jüttler. Cartesian spline interpolation for industrial robots. *Computer-Aided Design*, 30:217–224, 1998.
16. J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. AK Peters, Wellesley MA, 1993.
17. D.-E. Hyun, B. Jüttler, and M.-S. Kim. *Minimizing the Distortion of Affine Spline Motions*. Proceedings of Pacific Graphics, 50–59, Tokyo, 2001.
18. J. Johnstone and J. Williams. Rational control of orientation for animation. In W.A. Davis and P. Prusinkiewicz, editors, *Graphics Interface '95*, pages 179–186, Canadian Human-Computer Communication Society, Toronto, 1995.
19. B. Jüttler. Über zwangsläufige rationale Bewegungsvorgänge. *Sb. Österr. Akad. Wiss., Abt. II*, 202:117–132, 1993.
20. B. Jüttler. *Rationale Bézierdarstellung räumlicher Bewegungsvorgänge und ihre Anwendung zur Beschreibung bewegter Objekte*. PhD Thesis, Darmstadt University of Technology, 1994. (Published at Shaker Verlag, Aachen).
21. B. Jüttler. Visualization of moving objects using dual quaternion curves. *Computers & Graphics*, 18(3):315–326, 1994.
22. B. Jüttler. Spatial rational motions and their application in computer aided geometric design. In M. Dæhlen et al., editors, *Mathematical Methods for Curves and Surfaces*, pages 271–280. Vanderbilt University Press, Nashville, TN, 1995.



23. B. Jüttler and M. Wagner. Computer aided design with spatial rational B-spline motions. *ASME J. Mechanical Design*, 118:193–201, 1996.
24. B. Jüttler and M. Wagner. Rational motion-based surface generation. *Computer-Aided Design*, 31:203–213, 1999.
25. C. Keil. Approximation gemessener Kniebewegungen. Diplomarbeit, Dept. of Mechanics, Darmstadt University of Technology, 1995.
26. M.-J. Kim, M.-S. Kim, and S. Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. *Computer Graphics*, 29:369–376, 1995. (SIGGRAPH '95).
27. M.-J. Kim, M.-S. Kim, and S. Shin. A compact differential formula for the first derivative of a unit quaternion curve. *J. of Visualization and Computer Animation*, 7:43–57, 1996.
28. M.-S. Kim and K.-W. Nam. Interpolating solid orientations with circular blending quaternion curves. *Computer-Aided Design*, 27:385–398, 1995.
29. T. Kim and S. Sarma. Time-optimal paths covering a surface. In R. Cipolla and R.R. Martin, editors, *The Mathematics of Surfaces IX*, pages 126–143. Springer, London, 2000.
30. F. Klok. Two moving coordinate frames for sweeping along a 3D trajectory. *Computer Aided Geometric Design*, 3:217–229, 1986.
31. J. Kuipers. *Quaternions and rotation sequences. A primer with applications to orbits, aerospace, and virtual reality*. Princeton University Press, 1998.
32. S. Mick and O. Röschel. Interpolation of helical patches by kinematic rational Bézier patches. *Computers & Graphics*, 14:275–280, 1990.
33. K. Mørken. Some identities for products and degree raising of splines. *Constructive Approximation*, 7:195–208, 1991.
34. G. Nielson and R. Heiland. Animated rotations using quaternions and splines on a 4D sphere. *Program. Comput. Softw.*, 18:145–154, 1992. (Translated from *Programirovanie*, 4:17–27, 1992).
35. L. Noakes. Non-linear corner cutting. *Adv. Comput. Math.*, 8:165–177, 1998.
36. F. Park and B. Ravani. Bézier curves on Riemannian manifolds and Lie groups with kinematic applications. *ASME J. of Mechanical Design*, 115:36–40, 1995.
37. D. Pletinckx. Quaternion calculus as a basic tool in computer graphics. *The Visual Computer*, 5:2–13, 1989.
38. H. Pottmann. Studying NURBS curves and surfaces with classical geometry. In M. Dæhlen et al., editors, *Mathematical Methods for Curves and Surfaces*, pages 413–438. Vanderbilt University Press, Nashville TN, 1994.
39. H. Pottmann and M. Wagner. Principal surfaces. In T.N.T. Goodman and R.R. Martin, editors, *The Mathematics of Surfaces VII*, pages 337–362. Information Geometers, Winchester, 1997.
40. H. Pottmann and M. Wagner. Contributions to motion based surface design. *Int. J. of Shape Modeling*, 4:183–196, 1998.
41. H. Pottmann and J. Wallner. *Computational Line Geometry*. Springer-Verlag, Berlin, 2001.
42. R. Ramamoorthi and A. Barr. Fast construction of accurate quaternion splines. *Computer Graphics*, 31:287–292, 1997. (SIGGRAPH '97).

43. O. Röschel. Rationale räumliche Zwangläufe vierter Ordnung. *Sb. Österr. Akad. Wiss., Abt. II*, 194:185–202, 1985.
44. O. Röschel. Kinematic rational Bézier patches I/II. *Rad Hrvat. Akad. Znan. Umjet. Mat. Znan.*, 456:95–108 (I), 131–138 (II), 1991.
45. O. Röschel. Rational motion design – a survey. *Computer-Aided Design*, 30:169–178, 1998.
46. K. Shoemake. Animating rotations with quaternion curves. *Computer Graphics*, 19:245–254, 1985. (SIGGRAPH '85).
47. M. Wagner. *A Geometric Approach to Motion Design*. PhD thesis, TU Wien, Institute of Geometry, 1994.
48. M. Wagner. Planar rational B-spline motions. *Computer-Aided Design*, 27:129–137, 1995.
49. M. Wagner and B. Ravani. Curves with rational Frenet–Serret motion. *Computer Aided Geometric Design*, 15:79–101, 1997.
50. W. Wang and B. Joe. Robust computation of the rotation minimizing frame for sweep surface modeling. *Computer-Aided Design*, 29:379–391, 1997.
51. E. Weiss. *Einführung in die Liniengeometrie und Kinematik*. Teubner, Leipzig, 1935.
52. W. Wunderlich. Kubische Zwangläufe. *Sb. Österr. Akad. Wiss., Abt. II*, 193:45–68, 1984.
53. J. Xia and Q. Ge. On the exact representation of the boundary surfaces of the swept volume of a cylinder undergoing rational Bézier and B-spline motions. In *ASME Design Engineering Technical Conferences*. ASME, Las Vegas, 1999. (Paper no. DETC99/DAC-8607).

Chapter 30

Direct Rendering of Freeform Surfaces

Gershon Elber

Freeform geometry has been employed in computer graphics for more than three decades. Traditionally, the rendering of freeform surfaces has been preceded by a stage in which a piecewise linear polygonal approximation is derived from the original surfaces, leaving polygons to participate in the actual rendering process.

This chapter considers some recent results on the direct rendering of freeform geometry: in other words, the rendering of freeform curves and surfaces without resorting to piecewise linear approximations.

30.1. INTRODUCTION

Surface rendering is traditionally conducted with the aid of a piecewise linear approximation. Usually, curves are sampled and displayed as polylines and surfaces are approximated by polygons. Rendering of the resulting piecewise linear data is expected to be numerically more stable and is supported by contemporary hardware: finding the intersection between a light ray and a polygon is relatively straightforward. In contrast, computing the intersection between a ray and a polynomial surface is difficult, because it amounts to root finding. With the exception of trivial cases, roots must be found numerically. At the same time, the robustness of the rendering algorithm must be guaranteed. A failure of one computation in a million is unacceptable as even one wrong-colored pixel in the picture may be noticeable. The human visual system is surprisingly sensitive and is able to detect such minor defects and hence demands this extreme level of robustness.

Rendering techniques may be classified into several basic computational approaches. The simplest approach is to scan-convert the polygonal data set into a Z-buffer [15]. Each polygon, in the discrete image space, is converted into the set of pixels that approximately *covers* it. Then, the depth of each pixel is compared against the depth of the corresponding pixel already in the Z-buffer, and only the one at the front is kept. This approach is quite

simple and is nowadays implemented in hardware, even with PC-based systems.

Ray-tracing [16] is another common rendering method. Here, a (primary) ray is traced back from the eye, through a pixel in the image plane, and towards the scene. If the ray hits nothing, a background color is assigned to that pixel. Otherwise, let us assume the ray hits an object \mathcal{O} . If \mathcal{O} is reflective, a reflected ray is emitted from the intersection point on the surface of \mathcal{O} . If \mathcal{O} is translucent, then a refracted ray is emitted into \mathcal{O} , following Snell's law [15]. The final color of the pixel corresponding to the primary ray results from the accumulated contributions of the different surface properties of \mathcal{O} , including reflectivity, translucency, \mathcal{O} 's own color, the illumination, the surface normal of \mathcal{O} at that point, and so on.

Another rendering scheme, known as the Radiosity method [15], attempts to evaluate the exchange of light energy among all the surfaces in the scene. In this technique, the scene is subdivided into small differential area elements and a large set of linear equations is set up to express the exchange of light energy between every pair of elements. An approximate solution to this large set of linear equations is typically found using iterative methods, starting from the objects in the scene that represent the light sources.

All the above rendering methods, as well as many of their derivatives, have already been very successful in the context of polygonal geometry. But the topic of this chapter is their adaptation for the direct rendering of freeform geometry. We seek to eliminate the need for a piecewise linear approximation to feed the rendering pipeline, opening the way for the renderer to process the original freeform geometry. By doing so, we expect to gain in three different ways. Firstly, direct rendering of freeform surfaces is likely to be much more precise, because any piecewise linear representation is merely an approximation. Secondly, a single freeform surface, that can be expressed using a few dozen coefficients, has typically to be approximated using thousands of polygons in order to achieve reasonable accuracy. The direct use of freeform geometry potentially eliminates this explosion in the amount of data. Thirdly, and not least, the graphics community employs intensity (Gouraud) and normal (Phong) interpolation schemes [15] to compensate for the fact that piecewise linear approximations are C^1 -discontinuous. The elimination of the intermediate or auxiliary piecewise linear approximation would also remove the necessity for these interpolation schemes. As a result we would expect better and more accurate shading effects due to the precise normal at each pixel.

It is worth noting at this point that, while the Gouraud and Phong interpolation schemes are effective at hiding the illumination artifacts that result from C^1 discontinuities at the interior of the rendered polygonal object, C^1 discontinuities along the silhouette edges of the object are much more difficult to conceal. Attempts to increase the resolution of the piecewise linear approximation along the silhouette areas are feasible and can improve the visual appearance of the C^1 discontinuities but cannot disguise them completely [17].

There are a couple of good reasons for the use of polygons to render freeform surfaces. First, the polygons that approximate a surface are primitive entities that are simple and can be dealt with efficiently. Secondly, these polygons comprise a *coverage* of the original surface; if we paint all the polygons we are sure that the entire surface area has been visited and properly drawn.

Generalizing this second observation, we now define the concept of a *valid coverage*.

Definition 1 A set of primitives \mathcal{C} is called a valid coverage, with a tolerance δ , for a surface S if, for any point p on S (on \mathcal{C}), there is a point q in one of the primitives in \mathcal{C} (in S), such that $\|p - q\|_2 < \delta$, where $\|\cdot\|_2$ denotes Euclidean distance.

The primitives could be polygons, lines, curves, or even a dense cloud of points. Furthermore Definition 1 can be extended by using a non-Euclidean distance metric, for instance one that takes into account the curvature of the surface S or its Gaussian map.

A further consideration that complements the coverage requirement in Definition 1 is the optimality constraint. We seek a coverage that is optimal under certain criteria. For example, if we were still working with polygons, we would like to use as few as possible.

This chapter aims to present several different solutions to the problem of direct rendering of freeform geometry. In Section 30.2, we consider the direct scan-conversion of curves and, in Section 30.3, we introduce the coverage and rendering of freeform surfaces using primitive elements based on isoparametric curves. In Section 30.4, we consider methods that directly emulate the ray-tracing process for freeform geometry. Some extensions, such as isometric texture mapping and line-art renderings, are presented in Section 30.5. Finally, we draw some conclusions in Section 30.6.

30.2. SCAN-CONVERSION OF CURVES

We will start by considering freeform polynomial or rational curves. In this work, we will restrict ourselves to cubic curves: lower-order curves can clearly be raised to cubics and higher-order curves, which are quite rare in practice, can always be approximated by piecewise cubics [10].

Let $C(t)$, $t \in [0, 1]$, be a cubic curve. We seek to find all the pixels covered by $C(t)$ in the image plane. Traditionally, $C(t)$ is approximated as a polyline, and each of the resulting line segments in the polyline are scan-converted; but will now seek to eliminate the use of the piecewise linear auxiliary representation.

In Section 30.2.1, we consider the forward differencing method, which can generate points on the curve iteratively and very efficiently, and these points can be mapped on to pixels in the image plane. However, the regular forward differencing method has a major limitation that makes it difficult to use for scan-conversion. Prescribing a constant step size in the parametric domain may lead to variable steps in Euclidean space. Polynomial and rational functions are almost never curves of constant speed. In other words, $\|C'(t)\|$ is not usually constant. Thus, a step smaller than the distance between adjacent pixels at one end of the curve might expand into to a step much larger than a pixel at the other end of the curve. The step size must therefore be adjusted. In Section 30.2.2, we consider one possible way of doing this.

30.2.1. Forward differencing

Consider the following basis for cubic polynomials:

$$B_0(t) = 1, \quad B_1(t) = t, \quad B_2(t) = \frac{t(t-1)}{2!}, \quad B_3(t) = \frac{t(t-1)(t-2)}{3!}. \quad (30.1)$$

A cubic curve $C(t)$ may be represented in this basis as $C(t) = aB_0(t) + bB_1(t) + cB_2(t) + dB_3(t)$. The value of the curve at $t = 0$ is a . A forward step, $t \rightarrow t + 1$, in $C(t)$, could

be expressed as $\overline{D}(t) = \overline{a}B_0(t) + \overline{b}B_1(t) + \overline{c}B_2(t) + \overline{d}B_3(t) = C(t+1)$ with the coefficients (see [27]) equal to:

$$\begin{bmatrix} \overline{a} \\ \overline{b} \\ \overline{c} \\ \overline{d} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}. \quad (30.2)$$

Using the basis defined in Equation (30.1), we can advance along the curve with steps of unit size in the parametric domain simply by re-evaluating the four coefficients at every step. Thus, each step necessitates three summations (which could even be evaluated in parallel) resulting in a very efficient traversal of the freeform curve.

To make sure that each every pixel covered by $C(t)$ is indeed visited, the forward steps should be made small enough that they are at most one pixel apart in the image space. However, as already stated, a constant step size in the parameter domain does not guarantee a constant distance in the Euclidean space. We must provide an additional adaptive mechanism to decrease or increase the sizes of the steps. We consider this extension in Section 30.2.2.

30.2.2. Adaptive forward differencing

The basis (30.1) defined in the previous section is commonly known as *Adaptive Forward Differencing* [27] or AFD. Using the AFD basis, we can change the *length* of a forward step by changing the parameter t into $t/2$, or alternatively into $2t$. To halve the step size, $\overline{D}(t) = C(t/2)$, we use the coefficients:

$$\begin{bmatrix} \overline{a} \\ \overline{b} \\ \overline{c} \\ \overline{d} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{8} & \frac{1}{16} \\ 0 & 0 & \frac{1}{4} & -\frac{1}{8} \\ 0 & 0 & 0 & \frac{1}{8} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \quad (30.3)$$

and to double the step size, $\overline{D}(t) = C(2t)$, we use:

$$\begin{bmatrix} \overline{a} \\ \overline{b} \\ \overline{c} \\ \overline{d} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}. \quad (30.4)$$

Scan-converting a curve is now reduced to a series of forward steps (Equation (30.2)) which remain the same length for as long as each step yields a distance between half a pixel and a pixel. If a step comes out longer than a pixel, then the halving operator is applied (Equation (30.3)). Alternatively, if the step size shrinks below half a pixel, then the step size is doubled (Equation (30.4)).

AFD has a fixed initialization cost for each isoparametric curve. Moreover, a forward step is extremely efficient computationally as it amounts to only n summations, where n is the degree of the function (i.e. three for a cubic). Because the halving and doubling steps occur quite infrequently, the cost of the two types of speed changing steps is amortized over many pixels. Note that the summations and binary shifts (multiplications and divisions

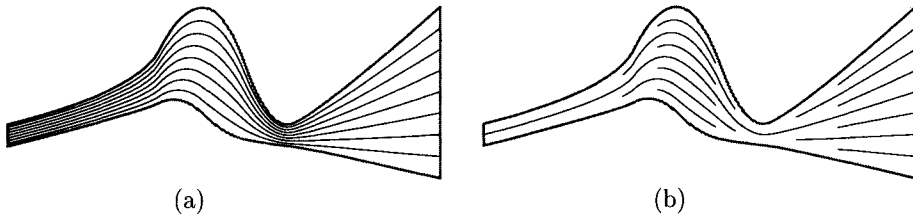


Figure 30.1. Redundancy may occur when surfaces are scan-converted using complete isoparametric curves (a). Adaptive extraction of partial isoparametric curves can produce better result (b).

by powers of two) corresponding to the x , y , and z coordinates in Equations (30.2)–(30.4) can be evaluated in parallel; and they usually will be, as integer arithmetic hardware is necessary for the efficient implementation of AFD.

Higher-order and rational polynomials can be also represented using the AFD basis (Equation (30.1)) if required, although rationals will require an additional division by the denominator to complete the evaluation. In all cases, care must be taken in designing the algorithm to ensure the stability and accuracy of the points generated. These details are among many aspects of AFD that have already been discussed in the literature [6,18,27].

30.3. SURFACE COVERAGE AND RENDERING USING CURVES

We have now explored the direct scan-conversion of curves. To extend the approach to surfaces, we must reduce the surface rendering problem to one of rendering a set of curves. In other words, we derive a *coverage* based on curves as primitives. These curves can subsequently be scan-converted, and shaded corresponding to the surface normal, thus yielding a complete rendering of the freeform surface.

Consider the parametric surface $S(u, v)$. One can try to place isoparametric curves at equally spaced parametric intervals, $u_i = i\sigma_u$, for some small positive real value σ_u , so as to generate adjacent isoparametric curves that are everywhere closer than δ in Euclidean space (see Figure 30.1 (a)), thus constructing a valid coverage \mathcal{C} of the surface S . Such an approach has been employed [18] in the direct rendering of freeform surfaces; the resulting curves are subsequently scan-converted using AFD.

To proceed in this manner, it is necessary to derive the proper parametric step, σ_u , so that the two adjacent isoparametric curves will be sufficiently close in Euclidean space. In one approach [27], the isoparametric curves are adaptively spaced using bounds that are derived from the distance function $d(v) = f(u + \delta_u, v) - f(u, v)$. Squared distance $d^2(t)$ is represented in the Bézier basis, and its bounds are estimated from its convex hull. A similar technique [18,25] uses the mean-value theorem. A bound on the Euclidean distance resulting from a small step h in the parametric space of a Bézier curve is computed as $\|C(u+h) - C(u)\| \leq n h \max\|P_{i+1} - P_i\|$, where n is the degree of $C(u)$, and $\{P_i\}$ are its control points. *Complete isoparametric curves*, that span the entire parametric domain

of the surface, can then be extracted and scan-converted [27,25].

Using a framework based on complete isoparametric curves, there is no upper bound on the number of times that a single pixel is actually revisited and redrawn. Recall Definition 1 and consider the surface region between two adjacent isoparametric curves in \mathcal{C} , with tolerance δ . Because \mathcal{C} is a valid coverage of S , the two adjacent isoparametric curves should be within distance 2δ . This upper bound of 2δ may be reached at one point (or more) along these two curves while they are arbitrarily close to each other elsewhere. Redundancy, which amounts to visiting the same pixels more than once while scan-converting the curves, is bound to occur. Figure 30.1 (a) shows an extreme case of redundancy resulting from the need to introduce *complete isoparametric curves* into the surface coverage.

One way to reduce the amount of pixel redundancy in this direct rendering procedure has been introduced by Rappoport [24]. A surface S will be subdivided if the amount of redundancy in S is greater than some acceptable level. The subdivision criterion is based on the range of partial derivatives in the cross direction $\partial S/\partial v$ over the patch. As division proceeds and the sub-patches generated become smaller, the range of derivatives across any one patch can also be expected to decrease. If there are significant deviations in the magnitudes of the cross derivatives on the original surface, the whole process may require a large number of subdivisions.

We now present a different approach, following Elber and Cohen [12], that visits and paints all the pixels covered by a surface S , avoiding division but providing a bound on the amount of pixel redundancy in the resulting coverage. Figure 30.1 (b) shows an example of this approach, which *adaptively* extracts *partial isoparametric curve segments* and covers the entire surface in an almost optimal way.

Assume that S is in the *viewing space*. Then the x and y surface coordinates are aligned with the image plane coordinates i_x and i_y (that is, $i_x = x$ and $i_y = y$), and S is the surface after it has been mapped on to the image plane, possibly with a perspective transformation. We are now also ready to introduce a complementary optimality constraint:

Definition 2 *A valid coverage of a surface S based on isoparametric curves is considered optimal if the accumulated cost of pixel drawing is minimal over all valid coverages.*

A *cost function of pixel drawing* should take into account the cost of initialization for drawing a curve, amortized over the length of the curve, plus the actual cost of drawing each pixel times the length of the curve in pixels.

Even if one could compute the required spacing in the parametric domain for a valid coverage of a given surface in a given scene, the extraction of all isoparametric curves in that spacing might be suboptimal, as can be seen from the middle part of the surface in Figure 30.2 (a). Because $\partial S/\partial v$ is not constant across the parametric domain of the surface, local dynamic adaptation of the parameter spacing is required, as seen in Figure 30.2 (b), so as to bring the coverage closer to optimal.

Using isoparametric curves as the coverage for a surface, we define *adjacency* between two isoparametric curves:

Definition 3 *Two isoparametric curves of a surface $S(u, v)$, $C_1(u) = S(u, v_1)$, $u \in [u_1^s, u_1^e]$ and $C_2(u) = S(u, v_2)$, $u \in [u_2^s, u_2^e]$, $v_1 \leq v_2$, from a given set \mathcal{C} of isoparamet-*

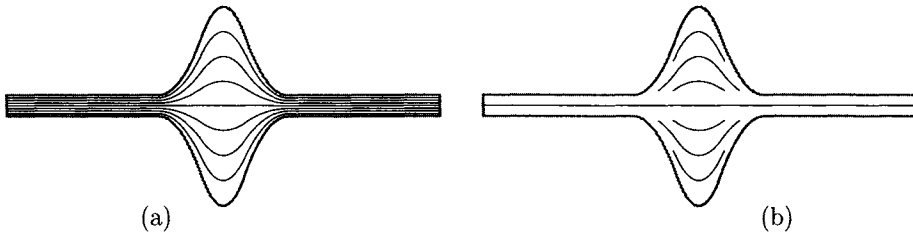


Figure 30.2. Complete isoparametric curves do not provide an optimal valid coverage for this biquadratic B-spline surface (a). Adaptive extraction of isoparametric curves that allows partial curve segments produces a better result and the coverage remains valid (b).

ric curves forming a valid coverage for S , are considered adjacent if, along their common domain $\mathcal{U} = [u_1^s, u_1^e] \cap [u_2^s, u_2^e]$, there is no other isoparametric curve from \mathcal{C} between v_1 and v_2 .

Now consider two adjacent curves, $C_1(u)$ and $C_2(u)$. We seek to discover whether the surface region between the two curves, $S(u, v)$, $u \in \mathcal{U}$, $v_1 < v < v_2$, satisfies the valid coverage constraint within tolerance δ . The notion of ‘walking the dog’ is discussed in [1], and named the *Frechet metric*. The idea is to find the shortest leash that will allow a dog to walk along one curve while its owner walks along the other. If, for $C_1(u)$ and $C_2(u)$, the leash is shorter than 2δ , the region between $C_1(u)$ and $C_2(u)$ is covered, with $C_1(u)$ and $C_2(u)$ serving as the valid coverage. Unfortunately, the Frechet distance between the two curves is extremely difficult to compute, making its use infeasible in real-time rendering, as such evaluations are required many times for each surface in the scene.

Hence, we employ a much more efficient computational alternative, which bounds the Frechet distance from above; we denote this alternative the *iso-distance*:

Definition 4 Given two isoparametric curves, $C_1(u) = S(u, v_1)$ and $C_2(u) = S(u, v_2)$, their iso-distance function $\Delta_{12}(u)$ is defined as $\Delta_{12}(u) = \|C_1(u) - C_2(u)\|_2$.

Clearly, the iso-distance between two curves bounds the Frechet distance from above. However, one can find extreme cases where the iso-distance will be arbitrarily large compared to the Frechet distance between the two curves. In Section 30.5.1, we will discuss how to deal with such cases.

In generating a rendered image, it is often sufficient to compute the iso-distance using only the x and y coordinate functions of the surface, since our concern is a coverage of the surface in the image plane. For example, consider a planar surface orthogonal to the image plane. This planar surface should be rendered as a line in the image plane: a consideration of iso-distance between isoparametric curves in the planar surface in x and y would indeed yield this result.

Nevertheless, ignoring the z -coordinate in the iso-distance computation may yield a wrong result when the surface is self-occluding or contains silhouette edges when seen from the viewing direction. Hence, for proper rendering using xy iso-distance, the surface

S should have no silhouette edges, or it must first be split along silhouette edges. Splitting should be conducted once, as a preprocess, creating a set of trimmed surfaces, each of which is free from interior silhouette edges.

In Section 30.3.1, we present the adaptive isoparametric curves (AIC) algorithm. In Section 30.3.2, we present some examples and discuss the corresponding rendering considerations.

30.3.1. Coverage based on adaptive isoparametric curves

Recall that $C_1(u) = S(u, v_1)$, $u \in [u_1^s, u_1^e]$, and $C_2(u) = S(u, v_2)$, $u \in [u_2^s, u_2^e]$, $v_1 \leq v_2$, are two adjacent isoparametric curves on a surface S . Further, let $\Delta_{12}(u) = \|C_1(u) - C_2(u)\|_2$. While $\Delta_{12}(u)$ is not rational, due to the square root, $\Delta_{12}^2(u)$ is rational. Compare the iso-distance square function, $\Delta_{12}^2(u)$, with δ^2 , where δ^2 is the square of the desired tolerance of the coverage (see Definition 1). Several possibilities can occur (note that, for the sake of simplicity, we examine the distances with respect to δ whereas 2δ might suffice):

1. $\Delta_{12}^2(u) < \delta^2$, $\forall u$. Here, $C_1(u)$ and $C_2(u)$ serve as a valid coverage for the region of the surface between the two curves, for $v_1 \leq v \leq v_2$.
2. $\Delta_{12}^2(u) > \delta^2$, $\forall u$. Here, $C_1(u)$ and $C_2(u)$ cannot be a valid coverage for the region of the surface between the two curves, for $v_1 \leq v \leq v_2$. In other words, we must introduce at least one additional, intermediate, curve between $C_1(u)$ and $C_2(u)$ in order for this set of curves to serve as a valid coverage of the surface region between the two curves, for $v_1 \leq v \leq v_2$.
3. $\Delta_{12}^2(u) = \delta^2$, for $u = u_i$, $i = 1, \dots, n$. In other words, the iso-distance function has n locations along the u domain where it is equal to the prescribed tolerance, δ . All locations along u where $\Delta_{12}^2(u) < \delta^2$ have a valid coverage, using only $C_1(u)$ and $C_2(u)$, as in Case 1 above. In contrast, for the domain along u for which $\Delta_{12}^2(u) > \delta^2$, it is necessary to introduce at least one more intermediate curve between $C_1(u)$ and $C_2(u)$, as in Case 2 above.

These three cases can be reformulated as a recursive procedure that starts with the two extreme boundary curves of the surface and considers them as two adjacent curves. Then, by examining the equation for the square of the iso-distance between the two curves, one of the three different cases can be identified, leading to two further curves which are dealt with recursively. Algorithm 1 encapsulates the entire process. Line (1) handles the case of two adjacent isoparametric curves that can serve as a valid coverage for the surface region between them. Line (2) considers the case where an intermediate curve must be introduced throughout the shared u domain of the two curves. Finally, in line (3), the final possibility is taken care of, where $\Delta_{12}^2(u)$ intersects the δ^2 line; this case further recurses into either Case (1) or Case (2).

There is one other problem. Consider a closed surface, S , where the first and last isoparametric curves, $C_1(u)$ and $C_2(u)$, of S are identical. In that case, Algorithm 1 would terminate immediately on S . One can always enforce a single subdivision at the top level of the recursion. However, even if the surface is not closed, two different regions might be close to each other, which would raise a similar difficulty. While a complete

solution is lacking, in most practical cases δ is far smaller than the surface size and it is highly unlikely that such a failure would occur.

In Figure 30.3, four steps of Algorithm 1 are presented, which demonstrate how these near-optimal coverages are constructed, using segments of isoparametric curves. Clearly, complete optimality cannot be claimed, if only because the iso-distance function is not true distance.

30.3.2. Rendering using adaptive isoparametric curves

Before we can scan-convert the isoparametric curves corresponding to the surface, one must be able to evaluate the normal of the surface at each point of each curve in the coverage. Toward this end, we evaluate the unnormalized surface normal field of S ,

$$\bar{n}(u, v) = \frac{\partial S}{\partial u} \wedge \frac{\partial S}{\partial v}, \quad (30.5)$$

where \wedge denotes the cross product.

For a bicubic polynomial surface S , the surface normal $\bar{n}(u, v)$ is a biquintic vector field. For every curve $C_i(u) \in \mathcal{C}$ that we extract from S , we also extract the normal field curve $\bar{n}_i(u)$ from $\bar{n}(u, v)$. The quintic normal field $\bar{n}_i(u)$ is then scan-converted along with the Euclidean curve $C_i(u)$, generating a precise yet unnormalized normal for every pixel. Normalizing the normal field necessitates the evaluation of a square root, which is quite expensive but could be tightly approximated using a lookup table.

Figure 30.4 shows a simple surface rendered using the AIC coverage. Figure 30.4 (a) shows a rendering in which the iso-distance between two adjacent isoparametric curves is several pixels, while Figure 30.4 (b) shows a similar rendering with a sub-pixel tolerance δ .

Figure 30.5 (a) shows the Utah teapot scan-converted using traditional polygon rendering methods and Figure 30.5 (b) shows the result based on adaptive isoparametric curves. The highlights that result from the use of the AIC method are superior due to the precise normals that are assigned to each and every pixel. Moreover, using the AIC-based coverage the silhouette curves are clearly smoother. Of course, one could alleviate the artifacts in the highlights and along the silhouette areas in the polygon-based picture in Figure 30.5 (a) by increasing the resolution of the approximation; but then the number of polygons generated could quite easily become very large. The image in Figure 30.5 (a) has a moderate resolution, but even so 7000 polygons were needed to generate it.

Parametric texture mapping, where the u and v parameters of the surface serve as coordinates into the texture image space, is commonly used in rendering freeform surfaces. By scan-converting isoparametric curves, indexing into the texture image is simplified because one of the surface parameters is fixed, and we are marching along a vertical or a horizontal line of pixels in the texture image. Figure 30.6 (a) shows an example of the Utah teapot with a variety of textures mapped on to it. The body is covered with a parametric texture mapping of the same image applied recursively. The spout, handle, and lid all employ volumetric texture mapping [21,22] in different patterns: camouflage, wood, and ‘virtual planet’, respectively.

It is considered difficult to approximate and convert a trimmed surface into polygons for display and rendering purposes [26,28]. On the other hand, rendering a trimmed surface as a set of isoparametric curves is appealing since the representation of each curve remains

Algorithm 1**Input:**

$S(u, v)$, $u \in [u_{min}, u_{max}]$, $v \in [v_{min}, v_{max}]$: input surface;
 δ : maximum distance (tolerance) between isoparametric curves;

Output:

C : the set of v -constant isoparametric curve segments of $S(u, v)$,
 adjacent within δ , covering S ;

Algorithm:

```

adapIsoCrvs(  $S$ ,  $\delta$  )
begin
   $C_1(u)$ ,  $C_2(u) \leftarrow$  isoparametric curves of  $S$  in  $u$  direction at  $v_{min}$ ,  $v_{max}$ 
  return
    {  $C_1(u)$  }  $\cup$ 
    adapIsoCrvsAux(  $S$ ,  $\delta$ ,  $v_{min}$ ,  $v_{max}$ ,  $C_1(u)$ ,  $C_2(u)$  )  $\cup$ 
    {  $C_2(u)$  };
end
end

adapIsoCrvsAux(  $S$ ,  $\delta$ ,  $v_{min}$ ,  $v_{max}$ ,  $C_1(u)$ ,  $C_2(u)$  )
begin
   $[u_1^s, u_1^e] \leftarrow$   $u$  domain of  $C_1(u)$ ;
   $[u_2^s, u_2^e] \leftarrow$   $u$  domain of  $C_2(u)$ ;
   $\mathcal{U} = (u_{min}, u_{max}) \leftarrow [u_1^s, u_1^e] \cap [u_2^s, u_2^e]$ , common domain of  $C_1(u)$  and  $C_2(u)$ ;
   $\Delta_{12}^2(u) \leftarrow$  squared iso-distance between  $C_1(u)$  and  $C_2(u)$ ,  $u \in \mathcal{U}$ ;
   $\mathcal{Z} \leftarrow$  roots of  $(\Delta_{12}^2(u) - \delta^2)$ ;
  if  $\mathcal{Z}$  empty then
    if  $\Delta_{12}^2(u) < \delta^2$ ,  $\forall u \in \mathcal{U}$  then
      (1) return {};
    else
       $v_{mid} \leftarrow (v_{min} + v_{max})/2$ ;
       $C_{12}(u) \leftarrow$  isoparametric curve of  $S$  at  $v_{mid}$ ,  $u \in \mathcal{U}$ ;
      return
        (2) adapIsoCrvsAux(  $S$ ,  $\delta$ ,  $v_{min}$ ,  $v_{mid}$ ,  $C_1(u)$ ,  $C_{12}(u)$  )  $\cup$ 
           {  $C_{12}(u)$  }  $\cup$ 
           adapIsoCrvsAux(  $S$ ,  $\delta$ ,  $v_{mid}$ ,  $v_{max}$ ,  $C_{12}(u)$ ,  $C_2(u)$  );
        end
    else
      (3) Subdivide  $C_1(u)$ ,  $C_2(u)$  at all  $u^i \in \mathcal{Z}$  into  $\{C_1^i(u), C_2^i(u)\}$  pairs;
           return  $\bigcup_i$  adapIsoCrvsAux(  $S$ ,  $\delta$ ,  $v_{min}$ ,  $v_{max}$ ,  $C_1^i(u)$ ,  $C_2^i(u)$  );
        end
  end
end

```

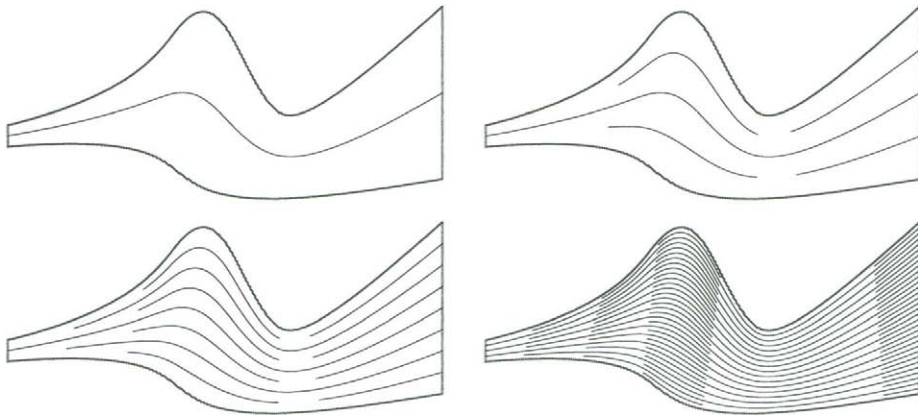


Figure 30.3. A bicubic surface covered using adaptive isoparametric curves (Algorithm 1) with different values of the tolerance δ .

exact after clipping to the domain of the trimmed surface. Because every one of these isoparametric curves is either a vertical or a horizontal line in the parametric domain, clipping itself is relatively straightforward. Figure 30.6 (b) shows another rendering of the Utah teapot, this time with holes formed using trimming curves.

30.4. RAY-TRACING

Methods to ray-trace freeform geometry directly by the numerical evaluation of roots or zero-sets are already known. The demand for extreme robustness and stability (for example, the failure of one pixel in a million would be unacceptable) puts severe restrictions on the possible methods one can use. Some approaches support only special classes of surfaces such as surfaces of revolution, extrusion surfaces and sweep surfaces, or combinations of these. In one such development [4], sweep surfaces are rendered directly, utilizing the generating curves of the sweep surface as the basis for the coverage.

Here, we will examine two approaches that support or emulate direct ray-tracing methods for freeform geometry. In Section 30.4.1, we will consider a method that is known as *Bézier clipping*, which is a robust yet efficient numerical method to derive *Ray-Surface Intersections* (RSI). In Section 30.4.2, we examine an extension to the AIC-based coverage that supports ray-tracing and is called *Ruled Tracing*.

30.4.1. Bézier clipping

In general, finding the roots of polynomial functions of arbitrary degree is a difficult problem. However, where polynomials represent geometry, the Bézier form may yield some benefits. It should be noted that a B-spline surface can easily be converted to a piecewise Bézier form and hence the method presented here will be equally valid in the domain of B-spline surfaces.

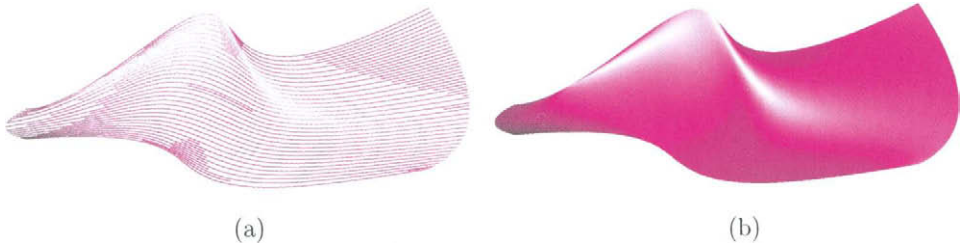


Figure 30.4. In (a), a simple surface is rendered using the AIC-based coverage with a tolerance δ that is several pixels wide. In (b), the same surface is rendered using the AIC-based coverage but with a sub-pixel tolerance δ .

Consider the parametric Bézier curve $C(t) = (x(t), y(t))$ of degree d . We seek the intersection points of $C(t)$ with the line $L : Ax + By + C = 0$ (see Figure 30.7 (a)). Substituting $C(t)$ into L , we are left with an implicit equation in one variable whose zero-set corresponds to the desired intersection points, $f(t) = Ax(t) + By(t) + C = 0$. Clearly, $f(t)$ is a scalar Bézier function of the same degree as $C(t)$,

$$f(t) = \sum_{i=0}^d c_i B_{i,d}(t),$$

where $B_{i,d}(t)$ are Bézier basis functions of degree d . Since the function is scalar, one could replace the coefficients c_i by $(c_i, \frac{i}{d})$, in the xy -plane, because $\sum_{i=0}^d \frac{i}{d} B_{i,d}(t) = t$. Let us denote the Bézier curve with coefficients $P_i = (c_i, \frac{i}{d})$ as $\hat{f}_0(t)$.

The curve $\hat{f}_0(t)$ is contained in the convex hull $\mathcal{CH}(\hat{f}_0)$ formed by its control points, $\{P_i\}$. Moreover, the zeros of $f(u)$ correspond to the intersection points of $\hat{f}_0(t)$ with the x -axis. Consider the first control point, P_0 of $\hat{f}_0(t)$ (see Figure 30.7 (b)). If P_0 is below the x -axis, as seen in Figure 30.7 (b), then the curve $\hat{f}_0(t)$ is guaranteed to be below the x -axis, as long as $\mathcal{CH}(\hat{f}_0)$ is also below the x -axis. Similar arguments hold if P_0 is above the x -axis. Moreover, the same line of reasoning holds for the last control point, P_d . Hence, one can clip $\hat{f}_0(t)$ from $t = 0$ up to t_1 (see Figure 30.7 (c)), where $\mathcal{CH}(\hat{f}_0)$ intersects the x -axis for the first time, and from $t = 1$ back down to t_2 , where $\mathcal{CH}(\hat{f}_0)$ intersects the x -axis for the last time, creating $\hat{f}_1(t)$. The clipped curve $\hat{f}_1(t)$ contains exactly the same zeros as $\hat{f}_0(t)$. This numerical process can be allowed to iterate until the zero-set solution is located with sufficient precision.

The extension of this Bézier clipping process to surfaces requires several intermediate representations, although it follows the general direction of the approach we have taken for curves. Consider the tensor product Bézier surface:

$$S(u, v) = \sum_i \sum_j P_{ij} B_{i,d_u}(u) B_{j,d_v}(v).$$

The ray intersecting $S(u, v)$ is defined as the intersection of two orthogonal planes, $A_k x + B_k y + C_k z + D_k = 0$, $k = 1, 2$. Nishita et al. [20] employ the scan plane and the plane



Figure 30.5. In (a), the Utah teapot is rendered using a traditional polygonal approximation, with over 7000 polygons. In (b), the same Utah teapot is rendered using AIC. Note the superior highlights that result, as well as the smoother silhouette edges.

containing the ray and the y -axis are for primary rays. Substitute the control points $P_{ij} = (x_{ij}, y_{ij}, z_{ij})$ of $S(u, v)$ into the planes, and let $d_{ij}^k = A_k x_{ij} + B_k y_{ij} + C_k z_{ij} + D_k$. Assuming $A_k^2 + B_k^2 + C_k^2 = 1$, d_{ij}^k equals the distance from P_{ij} to the k th plane. We now define a new planar surface as:

$$D(u, v) = \sum_i \sum_j (d_{ij}^1, d_{ij}^2) B_{i,d_u}(u) B_{j,d_v}(v).$$

Surface clipping will be conducted over $D(u, v)$, which is merely an orthographic projection of $S(u, v)$ along the ray, if $S(u, v)$ is a polynomial surface. Even if $S(u, v)$ is rational, $D(u, v)$ remains a polynomial.

The solution of $D(u, v) = 0$ corresponding to the intersection between the ray and the original surface $S(u, v)$ is obtained by performing Bézier clipping steps over $D(u, v)$ with respect to some line in the plane. Nishita et al. [20] give more details and also discuss efficiency and timing considerations. They describe how trimmed surfaces are supported by the manipulation of Bézier trimming curves. Computations of the inclusion/exclusion decisions in both the untrimmed and trimmed domains of the surface are also derived through Bézier clipping. These authors discuss efficiency and timing considerations.

30.4.2. Ruled tracing

At the core of every ray-tracing technique is the need to compute the intersection between a ray R and some surface S , the Ray-Surface Intersection (RSI) problem.

Primary rays are rays from the viewer or the eye through each pixel into the scene; they are typically evaluated in scan-line order, exploiting Z-buffer coherence to solve the RSI problem efficiently. Now consider the problem of casting rays from the points found in primary RSI towards the light sources in the scene, in order to detect regions that are in shadow.

Suppose we have in our scene a horizontal triangular polygon, \mathcal{T} , at depth $z = 1$, positioned above a horizontal rectangular polygon \mathcal{P} at depth $z = 0$ (see Figure 30.8 (a)).



Figure 30.6. In (a), several textures are applied to the Utah teapot. A parametric texture mapping is applied to the body of the pot, with a repeating texture pattern. The spout, handle and cap have all received different volumetric texture patterns. In (b), a set of trimmed surfaces, representing a teapot with holes, are rendered. Both images were created with the aid of an AIC-based coverage of the freeform surfaces.

We seek the domain of \mathcal{P} in the current scan-line, L_{y_0} , which is hidden from light source \mathcal{L} due to \mathcal{T} and hence is in shadow. Let $L_{\mathcal{P}} = \{p \mid p \in \mathcal{P} \cap L_{y_0}\}$ be the domain of \mathcal{P} that affects the current scan-line. Traditionally, the domain in shadow along $L_{\mathcal{P}}$ is resolved by emitting one ray per pixel toward \mathcal{L} .

However, one can approach this shadow detection problem differently. Consider the ruled surface R^s that passes through both the light source and the intersecting edge $L_{\mathcal{P}}$ (see Figure 30.8 (b)):

$$R^s(u, v) = v\mathcal{L} + (1 - v)L_{\mathcal{P}}(u), \quad 0 < v < 1.$$

We now compute the intersection of $R^s(u, v)$ with the polygon \mathcal{T} . If $R^s(u, v) \cap \mathcal{T}$ is empty, then clearly \mathcal{T} casts no shadows on \mathcal{P} along the current scan-line L_{y_0} . Alternatively, if $R^s(u_0, v_0) \in R^s(u, v) \cap \mathcal{T}$, then $R^s(u_0, 0)$ is hidden from the light source and hence it is in shadow. Moreover, $R^s(u_0, 0) = L_{\mathcal{P}}(u_0)$ and so we have a direct hit on the portion of $L_{\mathcal{P}}(u)$ that is in shadow.

$R^s(u, v)$ is a ruled surface, by its construction. In fact, because one of the boundary curves of $R^s(u, v)$ vanishes to a point at \mathcal{L} , $R^s(u, v)$ is a developable [8] (and planar) surface. The domain spanned in the u -direction of the set of $R^s(u, v) \cap \mathcal{T}$ corresponds to the domain \mathcal{U} that is in shadow, or alternatively illuminated, on \mathcal{P} for the whole of the current scan-line, L_{y_0} .

In the next section, we take this coherence-based optimization a step further. The objects in the scene are no longer polygons but are now freeform surfaces, and the surface-polygon intersection problem becomes a surface-surface intersection (SSI) problem. In other words, we are about to cast numerous RSI problems into much fewer SSI problems, expressed in the freeform surface domain.

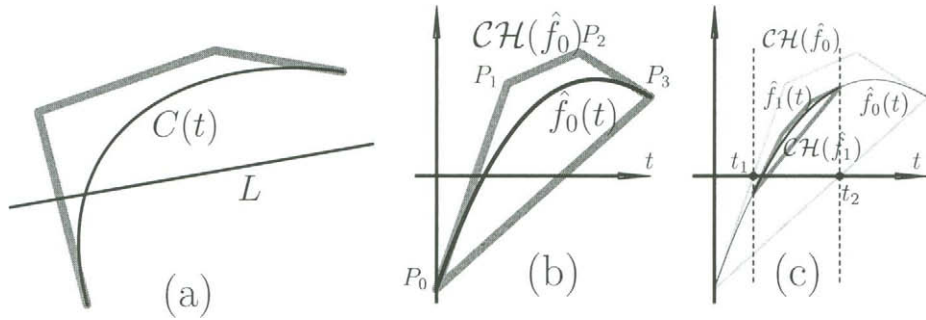


Figure 30.7. In (a), a cubic parametric Bézier curve is to be intersected with line L . The substitution of $C(t)$ into L is shown in (b) as an explicit function $\hat{f}_0(t)$. Clipping the convex hull domain of $\hat{f}_0(t)$ to be between t_1 and t_2 results in $\hat{f}_1(t)$, which is shown in (c).

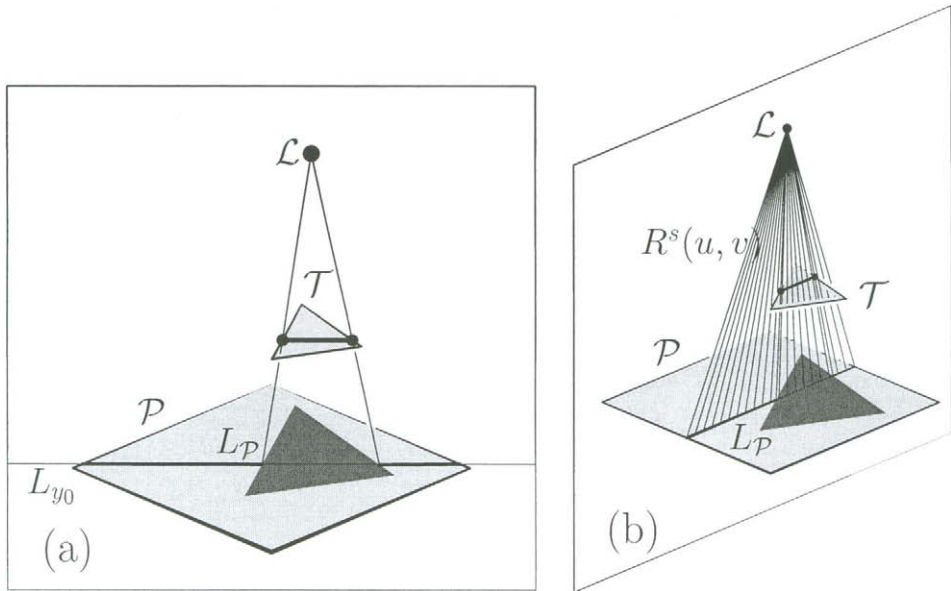


Figure 30.8. In (a), the shadowed regions of \mathcal{P} due to \mathcal{T} are sought. These regions are then separated from the illuminated domains along the current scan-line, L_{y_0} . In (b), a ruled surface $R^s(u, v)$ is constructed between \mathcal{L} and $L_{\mathcal{P}}$, so that $R^s(u, v) \cap \mathcal{T}$ represents the portion of \mathcal{U} that is in shadow.

Ruled tracing of freeform surfaces

The AIC elements \mathcal{C} that cover a freeform surface S do not have to be scan-line oriented univariate functions or even straight lines. Scan-line oriented rendering requires the intersection of a plane and the surface $S(u, v)$, so as to deal with the primary rays traced back to the eye. Using the AIC coverage, we can entirely eliminate the plane-surface intersections that are needed for conventional scan-line oriented rendering.

The intersection of a plane and a freeform surface $S(u, v)$ is not in general an isoparametric curve. Scan-line oriented rendering incrementally constructs a valid coverage for $S(u, v)$ in the image plane, which guarantees that all pixels in the image plane covered by $S(u, v)$ will be visited. Although scan-line oriented ray-tracing can benefit from ruled tracing, it is possible to eliminate all computation of plane-surface intersections for the primary rays that pass through the eye.

Assume that our scene consists of a set of surfaces \mathcal{S} , and recall the AIC coverage \mathcal{C} of a freeform surface S . Let \mathcal{C}^j be a set of isoparametric curves of $S^j(u, v) \in \mathcal{S}$, that forms a valid coverage for $S^j(u, v)$, and let $C_i^j(u) = S^j(u, v_i) \in \mathcal{C}^j$. Consider a light source \mathcal{L} and define the ruled surface between \mathcal{L} and curve $C_i^j(u)$ as:

$$\begin{aligned} R_i^{sj}(u, v) &= C_i^j(u) + v(\mathcal{L} - C_i^j(u)), \\ &= (1-v)C_i(u) + v\mathcal{L}, \quad 0 < v < 1. \end{aligned} \quad (30.6)$$

Clearly, from its construction, $R_i^{sj}(u, v)$ is a generalized cone, or more precisely a *developable surface*. One of the boundary curves of $R_i^{sj}(u, v)$ vanishes to a point, at \mathcal{L} . The following observation is fundamental:

Let $C_i^j(u) \subset S^j(u, v)$. A point $C_i^j(u_0)$ is hidden from \mathcal{L} and hence is in shadow if and only if $\exists S^k(r, t) \in \mathcal{S}$ such that $R_i^{sj}(u_0, v) \cap S^k(r, t) \neq \emptyset$, for some $0 < v < 1$.

If $j = k$, we are testing for a self-occluding surface. Because the domain of v is open, $C_i^j(u)$ itself is excluded from this intersection test. That observation allows one to pose a large set of regular RSI problems, millions in a typical image, in term of much fewer developable-surface-surface intersection (DSSI) problems. A solution to the DSSI of $R^s(u, v)$ against all the surfaces in \mathcal{S} yields all the RSI solutions necessary to determine the domain of $C_i^j(u)$ that is in shadow. Hence, one can form a coverage of isoparametric curves \mathcal{C}^j for the surface $S^j(u, v)$. Then, for each isoparametric curve $C_i^j(u) \in \mathcal{C}^j$, we resolve its shadow domain by constructing a complete ruled surface from $C_i^j(u)$ to the light source \mathcal{L} . Continuing in this way, we solve the DSSI against all surfaces in the scene.

So far we have shown how to delineate the shadow regions from the illuminated ones. Let $\mathcal{V}(u)$ be the unit-size *viewing direction* toward $C_i^j(u)$ and let $\vec{n}_i^j(u) = \vec{n}^j(u, v_i)$ be the unit-size *surface normal* along $C_i^j(u) = S^j(u, v_i)$. Let the unnormalized surface normal be denoted by $\bar{n}_i^j(u) = \left. \frac{\partial S^j(u, v)}{\partial u} \wedge \frac{\partial S^j(u, v)}{\partial v} \right|_{v=v_i}$, with $\vec{n}_i^j(u) = \frac{\bar{n}_i^j(u)}{\|\bar{n}_i^j(u)\|}$. Then the reflection direction, \vec{r} , can be computed as:

$$\begin{aligned} \vec{r}_i^j(u) &= 2 \langle \vec{n}_i^j(u), \mathcal{V}(u) \rangle \vec{n}_i^j(u) - \mathcal{V}(u), \\ &= 2 \frac{\langle \bar{n}_i^j(u), \mathcal{V}(u) \rangle \bar{n}_i^j(u)}{\|\bar{n}_i^j(u)\|^2} - \mathcal{V}(u). \end{aligned} \quad (30.7)$$

Thus, the specular reflection off the univariate domain of $C_i^j(u)$ is formulated as:

$$R_i^{rj}(u, v) = C_i^j(u) + v\bar{r}_i^j(u), \quad v > 0. \tag{30.8}$$

$R_i^{rj}(u, v)$ is a ruled surface. Consider the (piecewise) polynomial or rational freeform surface $S^j(u, v)$. Assuming $\mathcal{V}(u)$ is (piecewise) rational, then $\bar{r}_i^j(u)$ and hence $R_i^{rj}(u, v)$ can be represented as (piecewise) rationals. This holds in at least one degenerate case where $\mathcal{V}(u)$ is a constant unit vector; namely, the case in which the viewing direction is fixed, which holds for primary rays and for rays originating from light sources at infinity. However, closer inspection of Equation (30.7) reveals that $\mathcal{V}(u)$ need not be unit size to generate a rational vector field in the reflection direction (see Elber et al. [13] for more details). Once constructed, $R_i^{rj}(u, v)$ is compared against all the surfaces in \mathcal{S} , and a ruled-surface-surface intersection (RSSI) is computed between each pair. The colors of the reflections can be determined from these intersections.

Algorithm 2 summarizes the *ruled tracing* approach that we have now explained. The algorithm includes computation of the shadowing developable surfaces with respect to the light sources, as well that of the reflecting ruled surfaces, as discussed above. In Line (1) of Algorithm 2, the computation arrives at a single sequence of pixel colors along $C_i^j(u)$. The term \mathcal{U}_i^s represents the domain of $C_i^j(u)$ that is in shadow, and \mathcal{U}_i^r expresses the coloring contribution of the reflections along $C_i^j(u)$, as $R_i^{rj}(u, v)$ is intersected against \mathcal{S} , based on the colors of the surfaces hit by the reflected ruled surface. In Line (1), the reflectance information and the detected shadows are combined with the original surface properties of $S_i(u, v)$ to derive the displayed colors along the entire domain of $C_i^j(u)$. The next section discusses our implementation and presents some examples, drawing from Algorithm 2.

Examples

Given a ‘black box’ that is able to resolve SSI (surface-surface intersection) queries, the AIC algorithm could be extended to support the emulation of ray-tracing by means of ruled tracing. See Elber et al. [13] for some discussion on specially optimized RSSI (ruled SSI) and DSSI (developable SSI) intersection algorithms.

Figure 30.9 shows a computer model of an F16 aircraft, rendered with self-shadowing using ruled tracing. Figure 30.10 shows a king chess piece that was rendered using ruled tracing, with different tolerances in the AIC algorithm.

30.5. EXTENSIONS

The ability to cover a freeform surface using simple primitives has a range of applications and extensions, of which we will consider several here. In Section 30.5.1, we will examine the ability to optimize the layout of parametric textures on freeform surfaces which are in general non-isometric. In Section 30.5.2, we will apply the AIC coverage to the NC machining of freeform surfaces. Finally, in Section 30.5.3, we present line-art rendering techniques of freeform surfaces with the aid of AIC.

30.5.1. Isometric texture mapping

The regular mapping of texture on to parametric surfaces as part of efficient scan-conversion using a set of isoparametric curves is known to be feasible [5,12,27]. Un-

Algorithm 2**Input:**

\mathcal{L}_k : light sources location/direction.
 \mathcal{S} : set of surfaces defining the scene.

Output:

Rendered image.

Algorithm:

```

RuledTrace(  $\mathcal{S}$  )
begin
  for each  $S^j(u, v) \in \mathcal{S}$  do
     $C^j \Leftarrow$  Valid coverage of  $S^j(u, v)$  using AIC (Algorithm 1)
    for each  $C_i^j(u) \in C^j$  do
       $R_i^{sj}(u, v) \Leftarrow (1 - v)C_i^j(u) + v\mathcal{L}_k, \quad 0 < v < 1;$ 
       $\mathcal{U}_i^{sj} \Leftarrow R_i^{sj}(u, v)$ 's  $u$ -domain that intersects  $\mathcal{S}$ ;
       $R_i^{rj}(u, v) \Leftarrow C_i^j(u) + v\mathcal{r}_i^j(u), \quad v > 0;$ 
       $\mathcal{U}_i^{rj} \Leftarrow$  incoming and reflected illumination from  $\mathcal{S}$ ;
    (1) Scan-convert  $C_i^j(u)$  using  $\mathcal{U}_i^{sj}$  and  $\mathcal{U}_i^{rj}$ ;
    endfor
  endfor
end

```

fortunately, due to the fact that parametric surfaces are rarely isometric, the texture is warped as it is mapped on to the freeform geometry. Attempts to alleviate and control these texture distortions have been made [2,3,19] using geodesic curves and relaxation techniques [3,19] to flatten the surface approximately, or by limiting the viewing directions [2]. In Section 30.5.1, we explore the use of the AIC coverage in the search for a less distorted texture mapping.

Texture mapping using AIC

We are interested in the ability to exploit the iso-distance (Definition 4) to produce a *minimally distorted* texture mapping. While it is clear that the texture will always be distorted for non-developable surfaces, we seek to minimize these distortions.

By accumulating the arc-length of a scan-converted curve as it is traversed, it is possible to know the true distance of any point along the isoparametric curve. Further, the scan-conversion process deals serially with the isoparametric curves in the AIC. Hence, with the aid of Algorithm 1 and the computed iso-distances $\Delta_{12}(u)$ we can also accumulate the distance across isoparametric curves.

Measurements of the arc-length of $C(u)$ during the scan-conversion process should be quite accurate. Even so, the (squared) distance represented by $\Delta_{12}^2(u)$ will yield the wrong results if the vectors $C_1(u) - C_2(u)$ are not orthogonal to $\frac{dC_1(u)}{du} \approx \frac{dC_2(u)}{du}$ (see Figure 30.11).

Let us denote the angle between vectors $C_1(u) - C_2(u)$ and $\frac{dC_1(u)}{du}$ by θ . Then $(C_1(u) - C_2(u)) \sin(\theta)$ is a first-order approximation to the true distance between adjacent isopara-



Figure 30.9. A computer model of an F16 aircraft is rendered using ruled tracing, including the effects of (self-)shadowing. Shadows from the right wing and the right elevator are cast on to the fuselage.

metric curves. As the distance between the two adjacent curves converges to zero, the approximation converges to the true distance. Moreover, $\sin^2(\theta)$ can be represented in the following rational form:

$$\begin{aligned} \sin^2(\theta) &= 1 - \cos(\theta)^2 \\ &= 1 - \frac{\left\langle C_1(u) - C_2(u), \frac{dC_1(u)}{du} \right\rangle^2}{\left\langle C_1(u) - C_2(u), C_1(u) - C_2(u) \right\rangle \left\langle \frac{dC_1(u)}{du}, \frac{dC_1(u)}{du} \right\rangle}. \end{aligned}$$

Then, instead of using $\Delta_{12}^2(u)$, we can employ the more accurate

$$\widehat{\Delta}_{12}^2(u) = \Delta_{12}^2(u) \sin^2(\theta). \tag{30.9}$$

With these modifications, we can now get a good estimate of distances along, as well as across, isoparametric curves. Nevertheless, it should be noted that nothing is measured nor guaranteed anywhere other than in the tangent plane T_p of the freeform surface. In other words, we are only considering and preserving the $\left\langle \frac{\partial S}{\partial u}, \frac{\partial S}{\partial u} \right\rangle$ term of the first fundamental form and distance in a direction that is orthogonal to $\frac{\partial S}{\partial u}$ in T_p . Clearly, the preservation of these distances does not yield an isometric mapping—as one would indeed expect with general parametric surfaces. Fortunately, in many instances, the parameterization of the surface is well-behaved and the proposed mapping scheme does indeed alleviate the distortions in the resulting texture.

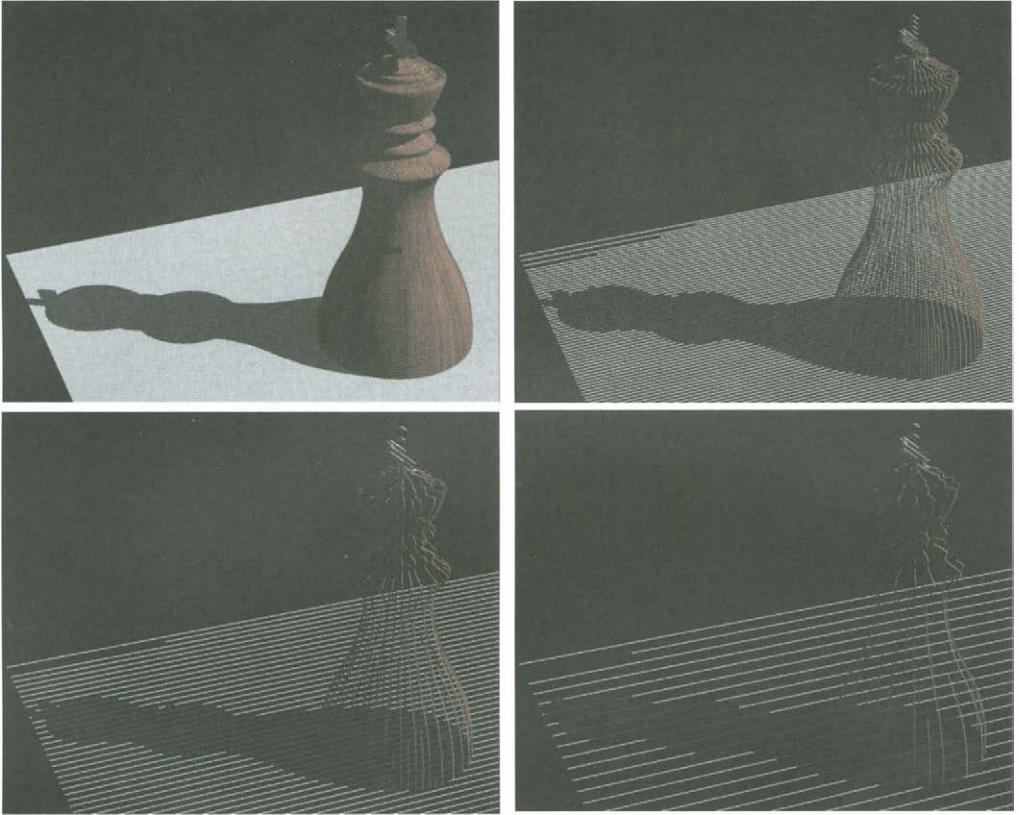


Figure 30.10. A king from a chess set is casting shadows on to the floor. Four different renderings at four different AIC tolerances are presented.

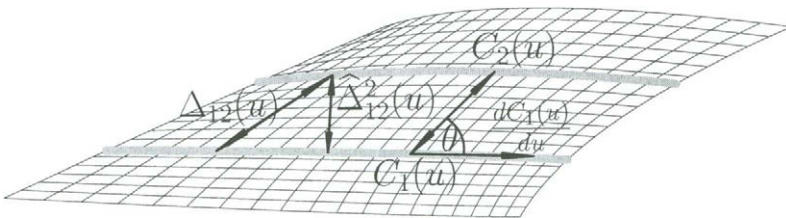


Figure 30.11. The iso-distance as imposed by $\Delta_{12}(u)$ could yield wrong results when the vectors $C_1(u) - C_2(u)$ are not orthogonal to $\frac{dC_1(u)}{du} \approx \frac{dC_2(u)}{du}$.

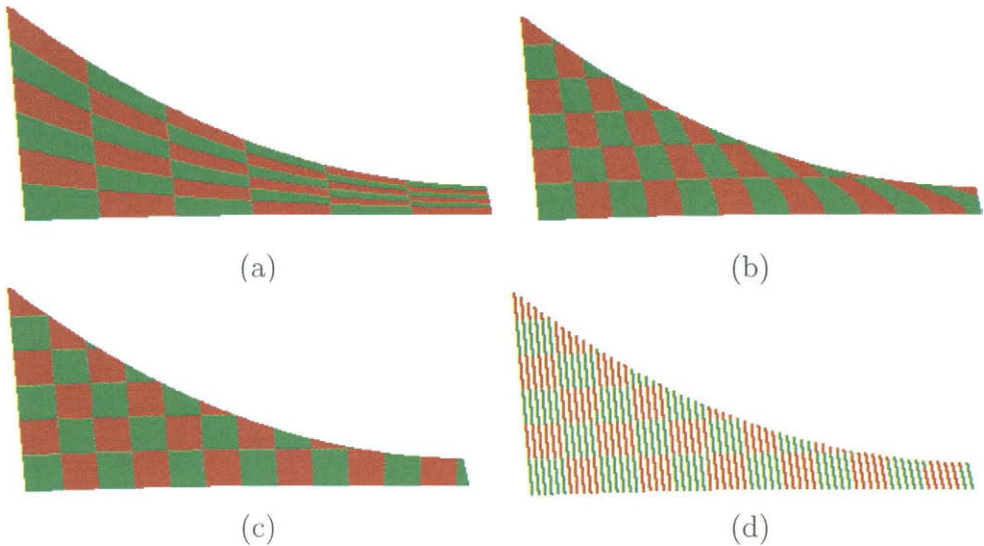


Figure 30.12. A simple planar surface, parametrically texture mapped with red/green checker squares. In (a), traditional texture mapping is exploited using the AIC-based coverage. In (b), the accumulation of arc-length along and across the isoparametric curves is employed. The picture (c) is similar to (a) but with the improved $\hat{\Delta}_{12}^2(u)$ (Equation (30.9)), and (d) shows the same rendering as (c) but with a larger AIC tolerance.

Distances along and across isoparametric curves are measured and accumulated from one or more prescribed corners of the parametric domain of the freeform surface. While the algorithm can select these corners at random, the final result will change when different corners are used. Hence, the user is given the ability to prescribe the corner or corners from which distances are accumulated. Section 30.5.1 demonstrates the effectiveness of this approach using some examples.

Results using AIC texture

Starting with a simple planar yet non-rectangular surface, Figure 30.12 results from the standard texture mapping method as well as from our proposed scheme, with and without the correction of $\hat{\Delta}_{12}^2(u)$, in Equation (30.9).

Finally, in Figure 30.13, a model of an F16 aircraft is rendered using the AIC coverage and almost-isometric mapping. The entire aircraft, except the wings, is rendered using a camouflage-based volumetric texture. In Figure 30.13 (a), checker squares are parametrically mapped on to the wings in the traditional way. In Figure 30.13 (b) the wings of the same F16 are parametrically texture mapped using the distance-preserving scheme presented in this section, yielding an almost isometric texture. Figure 30.13 (c) shows a different texture pattern for the wings.

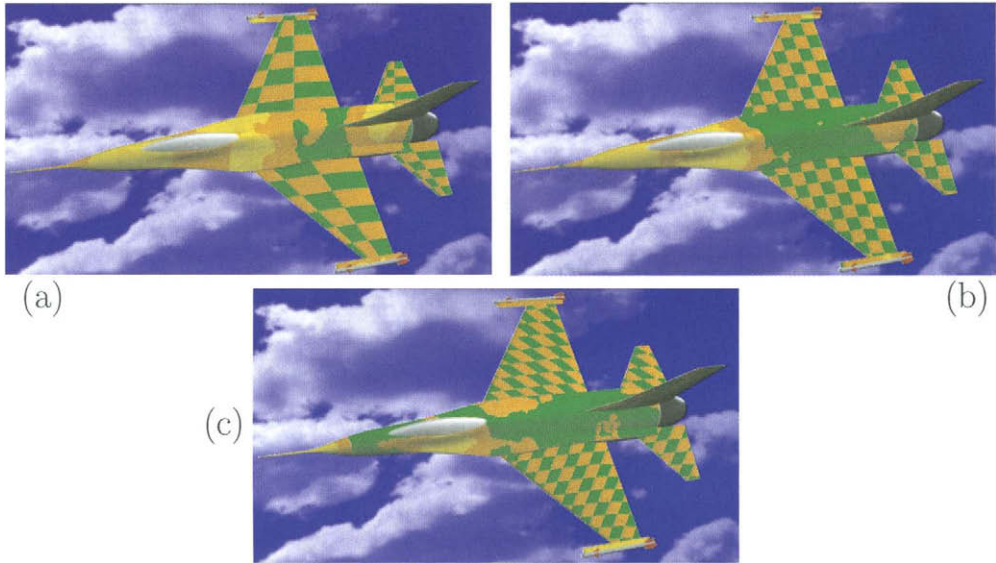


Figure 30.13. A model of an F16 aircraft, with wings that are parametrically texture mapped with a regular pattern. In (a) we see the traditional result that directly follows the (non-isometric) surface parameterization. In (b), almost-isometric texture mapping is employed; and (c) presents a different texture pattern.

More on this almost-isometric texture mapping can be found elsewhere [7].

30.5.2. Machining using adaptive isoparametric curves

Being able to compute the coverage of a surface is a development that holds the key to other problems that require all locations on a surface to be visited to within some prescribed tolerance. An immediate application is Numerically Controlled (NC) machining, in which an AIC coverage can be used to determine a tool path.

Figure 30.14 shows two examples of freeform shapes that were NC machined using a tool path based on an AIC coverage.

30.5.3. Line-art rendering

Recall the tolerance term δ in Definition 1. So far δ has been assumed to be constant. We will now examine the option of making δ a function. The shading $\delta(u, v)$ of a surface can be related to its illumination $S(u, v)$, $\forall u, v$. In particular, δ can now become a function of the viewing direction and the surface normal, as well as of the position and orientation of the light sources. In pursuit of this goal, we will employ a simple shading model that incorporates both diffuse and specular lighting components [15]. In Section 30.5.3, we lay down the necessary background and modifications that are necessary to support $\delta(u, v)$, and in Section 30.5.3 we present some examples.

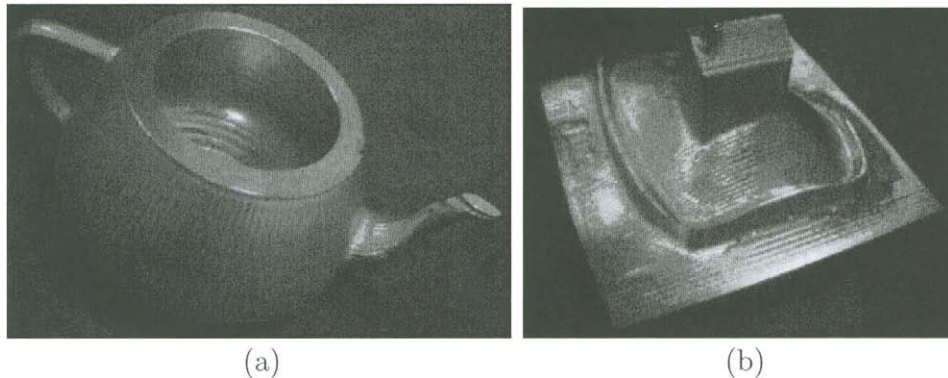


Figure 30.14. Freeform geometry that was NC machined using a tool path that was derived from a surface coverage based on the AIC algorithm.

Background and modifications towards line-art rendering

In previous work on line-art rendering [29], the Phong lighting model has been employed. We will also use the Phong lighting model in the generation of non-photorealistic line illustrations, this time using the AIC approach. We seek a modification to the surface coverage of Definition 1 so that δ can be employed as a simple shader. Let \mathcal{L}_i denote the unit vector of the i th light source and let \mathcal{V} denote the unit viewing direction. Given a surface $S(u, v)$, let $\vec{n}(u, v)$ be the unit normal field of $S(u, v)$. Then

$$\delta(u, v) = \alpha \langle \mathcal{L}_i, \vec{n}(u, v) \rangle, \quad (30.10)$$

where α is a handle controlling the intensity. The term $\delta(u, v)$ in Equation (30.10) prescribes a simple shader with only a diffuse lighting term. Adding a specular term yields

$$\delta(u, v) = \alpha \langle \mathcal{L}, \vec{n}(u, v) \rangle + \beta \langle \mathcal{V}, \vec{r}(u, v) \rangle^c, \quad (30.11)$$

where α and β are handles controlling the different lighting components, and $\vec{r}(u, v)$ is the direction of light from its source and reflected off $S(u, v)$, expressed as:

$$\vec{r}(u, v) = 2 \langle \vec{n}(u, v), \mathcal{L}_i \rangle \vec{n}(u, v) - \mathcal{L}_i,$$

(see Equation (30.7)) and c is the power of the specular term. See Foley et al. [15] for more details.

While neither shading model (Equations (30.10) and (30.11)) is (piecewise) rational due to the square-root normalization factors in the unit normal and reflection fields, $\delta^2(u, v)$ is rational. Because we are actually using $\delta^2(u, v)$ in our algorithm (see Algorithm 1), the use of the squared function $\delta^2(u, v)$ imposes no real difficulty.

Unfortunately, results from Algorithm 1 are likely to appear synthetic and artificial due to its binary subdivision nature (see Figure 30.15 (a)). Nevertheless, there exists a

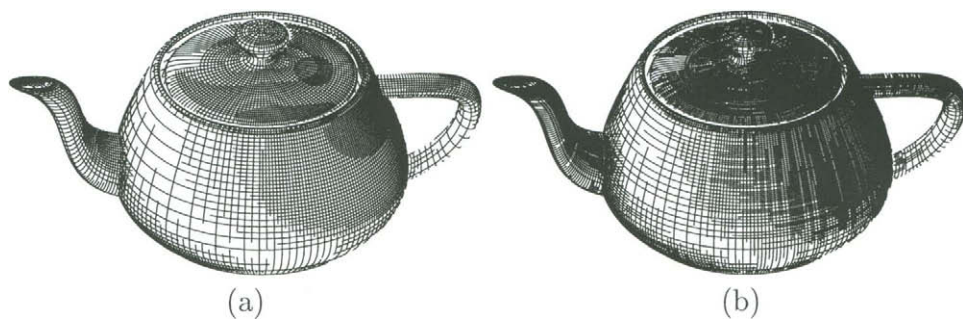


Figure 30.15. The direct adaption of Algorithm 1 to line-art rendering yields results that look synthetic and artificial (a). By adding some random noise to the δ function, these artifacts can be removed (b).

simple remedy. One can introduce white noise into the shader, denoted as $\hat{\delta}(u, v)$, before solving for the roots of $\Delta_{12}^2(u) - \hat{\delta}^2(u, v)$. This masks the pattern introduced by the binary subdivision: compare Figure 30.15 (a) with Figure 30.15 (b).

In order to present the results of Algorithm 1 convincingly, it is necessary to eliminate the invisible portion of the coverage. This can be done simply by using a standard Z-buffer to render the freeform surface model so that a complete depth map of the rendered scene is created, using either traditional polygons or the valid AIC coverage rendering that is described in Section 30.3.1. Then, the line-art coverage is translated a small distance along the z-direction toward the viewer and its visibility is verified against the depth map, isolating and extracting only the visible portion. This visible portion of the coverage is then approximated using piecewise cubic Bézier curves, which can be represented in the Postscript page description language [23] for printing.

In the next section, we go on to demonstrate some other possibilities of applying the line-art rendering technique to scenes of freeform models.

More examples of line-art rendering

So far we have employed a shader that takes into account both diffuse and specular lighting. We now explore two other possibilities. Consider a shader that enhances silhouette areas,

$$\delta(u, v) = (1 - \vec{n}_z(u, v))^c,$$

where \vec{n}_z is the z-component of the unit normal field of surface $S(u, v)$, and c serves as a decay factor as we move away from the silhouettes. See Figure 30.16 (a) for an example.

Yet another possibility is to make the shader respond to the intensity of illumination. The intensity of light decays with the square of the distance from its source; in Figure 30.16 (b), more isoparametric curves are drawn as we get closer to the light, which actually makes the surface look darker!

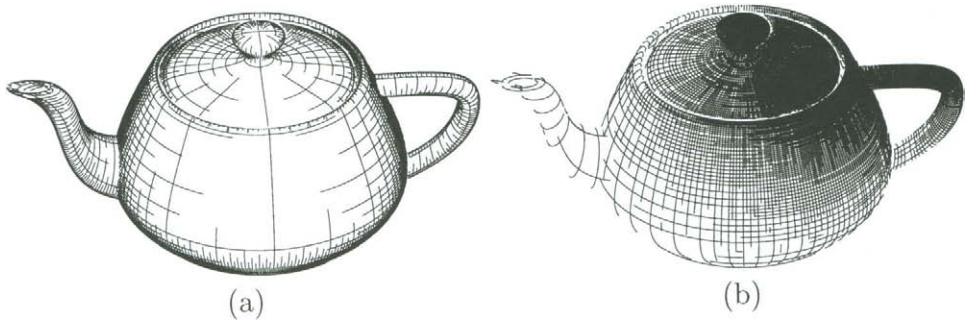


Figure 30.16. The shader can be made to affect $\delta(u, v)$ directly. In (a), this technique is used to enhance silhouette areas, whereas in (b) it is the distance from the light source (above and to the right) that affects $\delta(u, v)$.

Further work has been done on the use of AIC for line-art rendering [11]. In addition, attempts have been made to produce drawings of this sort in real time [14].

30.6. CONCLUSION

This chapter has presented several recent results in the direct scan-conversion and ray-tracing of freeform geometry. The ability to *cover* freeform surfaces optimally and efficiently using primitives which are simple to render, possibly in hardware, has been the key consideration.

In Section 30.5.1, we introduced $\widehat{\Delta}_{12}^2(u)$ with the aim of improving texture mapping capabilities. It can also be used to give a better AIC coverage, at the expense of more elaborate computations.

The adaptation of the Radiosity rendering scheme to the use of covering curves is more difficult. Isoparametric curves have no area and it is difficult to consider light energy exchange among zero-area elements. While the curves represent a finite surface area with width of the order of δ , direct application of the radiosity method to freeform surfaces remains a topic for research.

In recent years, there has been considerable interest in methods to process and render multivariate functions and surfaces directly. Trivariate functions are extensively exploited in medical visualizations and multivariate functions and surfaces are widely used in scientific visualization. This area of research is also expected to grow in the next few years.

The use of polygons to approximate freeform geometry has many benefits. Nonetheless, this approximation is also deficient in many ways. Recent interest in algorithms for directly handling freeform geometry suggests that the disadvantages of using polygonal approximations become ever more evident, in spite of their extreme simplicity. We will see whether direct surface rendering can challenge their lead.

REFERENCES

1. H. Alt and M. Godau. Measuring the resemblance of polynomial curves. In *Proc. the 8th Annual Symposium on Computational Geometry*, pages 102–109, Berlin, Germany, June 1992.
2. N. Arad and G. Elber. Isometric texture mapping. To appear in *Computer Graphics Forum*.
3. C. Bennis, J. M. Vezien, and G. Iglesias. Piecewise surface flattening for non-distorted texture mapping. *Computer Graphics*, 25(4):237–246, Siggraph, July 1991.
4. W. F. Bronsvort. A surface-scanning algorithm for displaying generalized cylinders. *The Visual Computer*, 8:162–170, 1992.
5. E. Catmull and A. R. Smith. 3D transformation of images in scanline order. *Computer Graphics*, 14(3):279–285, Siggraph, July 1991.
6. S. Chang, M. Shantz and R. Rocchetti. Rendering cubic curves and surfaces with integer adaptive forward differencing. *Computer Graphics*, 23(3):157–166, Siggraph, July 1989.
7. E. Cohen and G. Elber. Minimally distorted parametric texture mapping based on rendering of adaptive isoparametric curves. In *Proc. the Second “Korea-Israel Bi-National Conference on Computer Modeling and Computer Graphics in the World Wide Web Era”*, Seoul, Korea, September 1999.
8. M. P. DoCarmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976.
9. G. Elber and E. Cohen. Hidden curve removal for free form surfaces. *Computer Graphics*, 24(4):95–104, Siggraph, August 1990.
10. G. Elber. *Free Form Surface Analysis Using a Hybrid of Symbolic and Numeric Computation*. PhD thesis, University of Utah, Computer Science Department, 1992.
11. G. Elber. Line art rendering via a coverage of isoparametric curves. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):231–239, September 1995.
12. G. Elber and E. Cohen. Adaptive iso-curves based rendering for free form surfaces. *ACM Transactions on Graphics*, 15(3):249–263, July 1996.
13. G. Elber, J. J. Choi, and M. S. Kim. Ruled tracing. *The Visual Computer*, 13:78–94, 1997.
14. G. Elber. Interactive line art rendering of freeform surfaces. *Eurographics 99*, Milano, Italy, September 1999.
15. J. D. Foley and A. van Dam and S. K. Feiner and J. F. Hughes. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley Publishing Company, Second Edition, 1990.
16. A. S. Glassner. *An Introduction to Ray Tracing*. Academic Press, New York, Second Printing, 1990.
17. B. V. Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. *Computer Graphics*, 21(4):103–110, Siggraph, July 1987.
18. S. Lien, M. Shantz, and V. Pratt. Adaptive forward differencing for rendering curves and surfaces. *Computer Graphics*, 21(4):111–118, Siggraph, July 1987.
19. S. D. Ma and H. Lin. Optimal texture mapping. *Eurographics 88*, pages 421–428, September 1988.
20. T. Nishita, T. W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface

- patches. *Computer Graphics*, 24(4):337–345, Siggraph, August 1990.
21. D. R. Peachey. Solid texturing of complex surfaces. *Computer Graphics*, 19(3):279–286, Siggraph, July 1985.
 22. K. Perlin. An image synthesizer. *Computer Graphics*, 19(3):287–296, Siggraph, July 1985.
 23. *PostScript Language Reference Manual*. Adobe Systems Incorporated. Addison-Wesley Publisher Company, Second Edition, November 1994.
 24. A. Rappoport. Rendering curves and surfaces with hybrid subdivision and forward differencing. *ACM Transaction on Graphics*, 10(4):323–341, October 1991.
 25. A. Rockwood. A generalized scanning technique for display of parametrically defined surfaces. *IEEE Computer Graphics and Applications*, 7(8):15–26, August 1987.
 26. A. Rockwood, K. Heaton, and T. Davis. Real-time rendering of trimmed surfaces. *Computer Graphics*, 23(3):107–117, Siggraph, July 1989.
 27. M. Shantz and S. Chang. Rendering trimmed NURBS with adaptive forward differencing. *Computer Graphics*, 22(4):189–198, Siggraph, August 1988.
 28. X. Sheng and B. E. Hirsh. Triangulation of trimmed surfaces in parametric space. *Computer-Aided Design*, 24(8):437–444, August 1992.
 29. G. Winkenbach and D. H. Salesin. Computer-generated pen-and-ink illustration. *Computer Graphics*, pages 91–98, Siggraph, July 1994.

Chapter 31

Modeling and Processing with Quadric Surfaces

Wenping Wang

Quadric surfaces, or *quadrics*, are surfaces defined by algebraic equations of degree two. We will discuss mainly quadrics in 3D space, though quadrics in higher dimensions are also of certain interest to CAGD. In this chapter the following topics concerning quadrics will be discussed.

1. Definition and Classifications
2. Parametric Representation
3. Fitting, Blending, and Offsetting
4. Intersection and Interference

31.1. DEFINITION AND CLASSIFICATIONS

In this section we will define quadrics and discuss their classifications under different groups of transformations, *i.e.*, Euclidean, affine, and projective transformations [41].

31.1.1. Definition

A quadric is defined by a homogeneous quadratic equation $F(x, y, z, w) = 0$, where (x, y, z, w) are the homogeneous coordinates of a point in 3D space, with the corresponding affine coordinates $(x/w, y/w, z/w)$ for a finite point, *i.e.*, $w \neq 0$. The matrix representation of a quadric surface is given by

$$F(x, y, z, w) \equiv X^T M X = 0,$$

where $X = (x, y, z, w)^T$ is the column vector of coordinates and M is a 4×4 real symmetric matrix.

The geometric degeneracy of a quadric is determined by the rank of M . If the rank of M is 4, then the quadric is called *nondegenerate* or *proper*. If the rank of M is less than 4, then the quadric is called *singular*. When the rank of M is 3, the quadric is singular but irreducible, and is therefore called *properly degenerate*. The properly degenerate quadrics are cones and cylinders. When the rank of M is 1 or 2, the quadric degenerates to a pair of planes. The case of $\text{rank}(M) = 0$ is trivial and will not be considered. Since only real points on a quadric surface are of interest in practical applications, we assume, without loss of generality, that the entries of M are real numbers and that M is indefinite, *i.e.*, there exist real points X_0 and X_1 such that $X_0^T M X_0 > 0$ and $X_1^T M X_1 < 0$. Note that this assumption excludes those quadrics that are not real surfaces.

31.1.2. Euclidean classification

A Euclidean transformation is represented by

$$X' = \begin{bmatrix} O_{3 \times 3} & B \\ 0 & 1 \end{bmatrix} X,$$

where O is a 3×3 orthogonal matrix with $\det(O) = 1$, and B is a 3D translation vector. A Euclidean transformation $X' = UX$ transforms a quadric $X^T M X = 0$ to a quadric $X'^T (U^{-T} M U^{-1}) X' = 0$. Under Euclidean transformations an irreducible quadric can be converted to one of the following nine canonical forms.

Nondegenerate quadrics:

$$\begin{aligned} \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} - w^2 &= 0 & (\text{rank}(M) = 4, \text{ ellipsoid}) \\ \frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} - w^2 &= 0 & (\text{rank}(M) = 4, \text{ one sheet hyperboloid}) \\ \frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} + w^2 &= 0 & (\text{rank}(M) = 4, \text{ two sheet hyperboloid}) \\ \frac{x^2}{a^2} + \frac{y^2}{b^2} - zw &= 0 & (\text{rank}(M) = 4, \text{ elliptic paraboloid}) \\ \frac{x^2}{a^2} - \frac{y^2}{b^2} - zw &= 0 & (\text{rank}(M) = 4, \text{ hyperbolic paraboloid}) \end{aligned}$$

The first three of the above are called *central quadrics*.

Properly degenerate quadrics:

$$\begin{aligned} \frac{x^2}{a^2} + \frac{y^2}{b^2} - z^2 &= 0, & (\text{rank}(M) = 3, \text{ cone}) \\ \frac{x^2}{a^2} + \frac{y^2}{b^2} - w^2 &= 0 & (\text{rank}(M) = 3, \text{ elliptic cylinder}) \\ \frac{x^2}{a^2} - \frac{y^2}{b^2} - w^2 &= 0 & (\text{rank}(M) = 3, \text{ hyperbolic cylinder}) \\ ax^2 - yw &= 0 & (\text{rank}(M) = 3, \text{ parabolic cylinder}) \end{aligned}$$

Reducing a quadric surface to one of the above canonical forms by a Euclidean transformation is useful in analyzing quadrics, since this means that only one routine is needed to process each of the canonical forms and it is therefore not necessary to cope with an arbitrary quadratic form.

The fact that any quadric can be converted by a Euclidean transformation to one of these simple forms implies the existence of a local orthogonal coordinate system in which that form is assumed. In the case of a central quadric the coordinate axes of this coordinate system are the principal axes of the quadric. Denote

$$M = \begin{bmatrix} \tilde{M}_{3 \times 3} & m \\ m^T & c \end{bmatrix}.$$

In this case the vectors of the three axes of a central quadric $X^T M X = 0$ are the eigenvectors of $\tilde{M}_{3 \times 3}$, and the center of the quadric is given by $-\tilde{M}^{-1}m$, in Cartesian coordinates. Reducing a quadric surface in E^3 to a canonical form is closely related to the orthogonal reduction of a quadratic form to a diagonal form. The reader is referred to [29] for a discussion of the reduction procedure, as well as a general discussion about the classification of quadrics under different transformation groups.

31.1.3. Affine classification

An affine transformation is represented by

$$X' = \begin{bmatrix} A_{3 \times 3} & B \\ 0 & 1 \end{bmatrix} X,$$

where A is a 3×3 nonsingular matrix and B is a 3D translation vector. Under affine transformations an irreducible quadric can be converted into one of the following nine canonical forms. Since more general transformations are allowed in affine space than in Euclidean space, the coefficients in the following affine canonical forms are less arbitrary than those in the Euclidean canonical forms.

Nondegenerate quadrics:

$$\begin{aligned} x^2 + y^2 + z^2 - w^2 &= 0 && (\text{rank}(M) = 4, \text{ ellipsoid}) \\ x^2 + y^2 - z^2 - w^2 &= 0 && (\text{rank}(M) = 4, \text{ one sheet hyperboloid}) \\ x^2 + y^2 - z^2 + w^2 &= 0 && (\text{rank}(M) = 4, \text{ two sheet hyperboloid}) \\ x^2 + y^2 - zw &= 0 && (\text{rank}(M) = 4, \text{ elliptic paraboloid}) \\ x^2 - y^2 - zw &= 0 && (\text{rank}(M) = 4, \text{ hyperbolic paraboloid}) \end{aligned}$$

Properly degenerate quadrics:

$$\begin{aligned} x^2 + y^2 - z^2 &= 0 && (\text{rank}(M) = 3, \text{ cone}) \\ x^2 + y^2 - w^2 &= 0 && (\text{rank}(M) = 3, \text{ elliptic cylinder}) \\ x^2 - y^2 - w^2 &= 0 && (\text{rank}(M) = 3, \text{ hyperbolic cylinder}) \\ x^2 - yw &= 0 && (\text{rank}(M) = 3, \text{ parabolic cylinder}) \end{aligned}$$

Note that the principal axes of a central quadric are not, in general, mapped to the principal axes of the transformed quadric under an affine transformation. However, the center of a central quadric is mapped to the center of the transformed quadric by an affine transformation.

31.1.4. Projective classification

A real projective transformation in 3D is given by $X' = AX$, where A is any real 4×4 nonsingular matrix. A quadric is mapped to a quadric under a projective transformation and the rank of the coefficient matrix is not changed. Any irreducible quadric can be transformed projectively to one of the following three canonical forms:

$$\begin{aligned} x^2 + y^2 + z^2 - w^2 &= 0 && (\text{rank}(M) = 4, \text{ oval quadric}) \\ x^2 + y^2 - z^2 - w^2 &= 0 && (\text{rank}(M) = 4, \text{ doubly ruled quadric}) \\ x^2 + y^2 - z^2 &= 0 && (\text{rank}(M) = 3, \text{ cone}) \end{aligned}$$

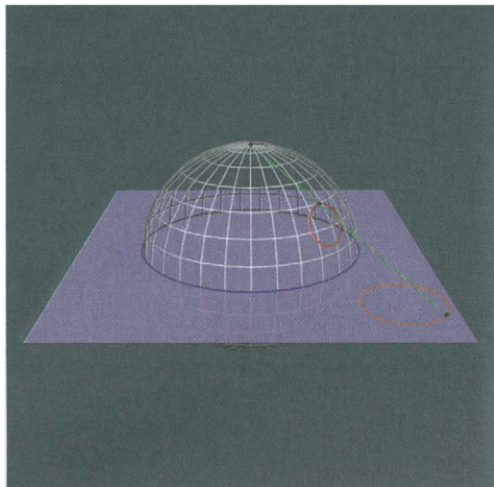


Figure 31.1. A stereographic projection on a sphere.

Other commonly used canonical forms for the doubly ruled quadric and the cone under projective transformations are the hyperbolic paraboloid $xy - zw = 0$ and the cylinder $x^2 + y^2 - w^2 = 0$, respectively.

Any of the affine canonical forms of quadrics can be obtained from one of the above three projective canonical forms by appropriately specifying an ideal plane for an affine realization of projective space. The affine views of quadrics in projective space are discussed in [11].

31.2. PARAMETRIC REPRESENTATION

For reasons of computational efficiency, rational or polynomial representations of curves and surfaces are preferred in CAGD because of their simple analytical properties. So one reason why quadrics are widely used in CAGD is that all quadrics have rational parameterizations of degree two. In this section we shall discuss the rational quadratic parameterization of a whole quadric surface and of a surface patch on a quadric.

31.2.1. Global rational parameterization

A simple way to derive a rational quadratic parameterization of a quadric is to use a stereographic projection of the quadric to establish a birational mapping between points on a plane and points on the quadric [41,43]. See Figure 31.1. Let X_0 be a point on quadric $S : X^T A X = 0$. Let $B_0^T X = 0$ be a plane not passing through X_0 . When S is a cone or cylinder, X_0 is assumed not to be the singular point of S . We will consider the projection that maps points on S through X_0 to points on plane $B_0^T X = 0$.

Let $T = (x, y, z, w)^T$ be a variable point on plane $B_0^T X = 0$, and let (r, s, t) be a projective coordinate system on $B_0^T X = 0$. Then there is a 4×3 matrix M such that

$T = M(r, s, t)^T$, and T or (r, s, t) is called a *parameter point*. Let $P_T(u, v) = uX_0 + vT$ be the line determined by X_0 and T . By Bézout's theorem, there are two points of intersection between S and line $P_T(u, v)$ unless $P_T(u, v)$ is contained in S , and one of these two intersections is X_0 . The exception occurs for the generators of S at X_0 , i.e., the straight lines on S that pass through X_0 . Point X_0 is called the *center of projection* for the projection induced by the line $P_T(u, v)$ from plane $B_0^T X = 0$ to quadric $X^T A X = 0$. Substituting $P_T(u, v)$ for X in $X^T A X = 0$, the parameter pair corresponding to the other intersection is found to be $u : v = T^T A T : (-2X_0^T A T)$. Hence, the other intersection is

$$\begin{aligned} P(r, s, t) &= (T^T A T)X_0 - 2(X_0^T A T)T \\ &= [(r, s, t)M^T A M(r, s, t)^T]X_0 - 2[X_0^T A M(r, s, t)^T]M(r, s, t)^T. \end{aligned}$$

This is a *faithful* rational quadratic parameterization of S ; a faithful parameterization is one that establishes a one-to-one correspondence between all points on the parametric plane and all points on the surface, with the usual exception of points on a finite number of curves on the surface.

Using the above procedure we obtain the following rational quadratic parameterization of the unit sphere $S^2 : x^2 + y^2 + z^2 - w^2 = 0$,

$$x = 2rt, \quad y = 2st, \quad z = r^2 + s^2 - t^2, \quad w = r^2 + s^2 + t^2,$$

with the center of projection at the north pole $X_0 = (0, 0, 1, 1)$ and projection plane $z = 0$. Here the homogeneous parameters (r, s, t) are identified with the homogeneous coordinates (x, y, w) in plane $z = 0$. (See Figure 31.1.) This is the standard stereographic projection of S^2 , which is circle-preserving.

A parametric rational quadratic surface is, in general, a quartic algebraic surface, but degenerates to a quadric in special cases [11]. A rational quadratic surface whose algebraic degree is higher than two is called a *Steiner surface* [37]. Steiner surfaces behave quite differently from rational quadratic curves, which are always conics. It is known from algebraic geometry [7] that the algebraic degree of a surface with a faithful rational quadratic parameterization is $4 - p$, where p is the number of *base points*; a base point of a rational surface $P(r, s, t) = (x(r, s, t), y(r, s, t), z(r, s, t), w(r, s, t))$ is a parameter point $(r_0, s_0, t_0) \neq (0, 0, 0)$ such that $P(r_0, s_0, t_0) = (0, 0, 0, 0)$. It follows that a faithful rational quadratic parameterization of a quadric has two base points. These two base points are distinct if and only if the quadric is nondegenerate [43].

The following properties of rational quadratic parameterizations of a quadric are proved elsewhere [43]. Two parameterizations derived using the above procedure with the same center of projection, but possibly with different projection planes, are related by a rational linear reparameterization; however, two parameterizations derived with two different centers of projection are related by a rational quadratic reparameterization, which is a 2D Cremona transformation [38]; a Cremona transformation is a birational transformation from plane to plane.

Any faithful rational parameterization of a quadric can be obtained using a stereographic projection; but not all rational quadratic parameterizations are faithful. The following is an example of an unfaithful parameterization of the cone $x^2 + y^2 - z^2 = 0$:

$$x = 2rs, \quad y = r^2 - s^2, \quad z = r^2 + s^2, \quad w = t^2.$$

It is unfaithful because, clearly, (r, s, t) and $(r, s, -t)$ yield the same point on the cone. Unfaithful parameterizations can exist only for cones and cylinders and has no base points, and any rational quadratic parameterization of a non-degenerate quadric is faithful.

31.2.2. Generalized stereographic projection

The stereographic projection defined on a quadric surface that we just introduced has some drawbacks for modeling and analyzing rational curves and surface patches on the quadric. Taking the unit sphere S^2 for example, suppose that the center of projection is at the north pole $N = (0, 0, 1, 1)$ and the plane of projection is $z = 0$. Then a quadratic curve C on S^2 , *i.e.*, a circle in this case, is mapped by the inverse of the stereographic projection into a circle on the projection plane $z = 0$ when C does not pass through N ; however, C is mapped to a line on plane $z = 0$ if C passes through N . The fact that the degree of the inverse image of a circle on S^2 depends on the relationship between the circle and the center of projection is inconvenient and caused by the dependence of the definition of stereographic projection on the center of projection, or more fundamentally, by the existence of the bases points of the stereographic projection. This problem can be summarized more generally as follows: since the stereographic projection of a quadric is a quadratic mapping, the image of a degree m rational curve on the projection plane is a rational curve of degree at most $2m$ on S^2 under the stereographic projection; however, one cannot, in general, assert that any rational curve of degree $2m$ on S^2 is the image of a rational curve of degree m under the stereographic projection.

To overcome this problem, the generalized stereographic projection has been introduced by Dietz, Hoschek, and Jüttler [9]. The generalized stereographic projection is a one-to-one correspondence between points on a quadric and a two-parameter family of lines in E^3 , *i.e.*, a line in E^3 is mapped to a point on a quadric, vice versa. Furthermore, this family of lines fills up the entire space E^3 , so the generalized stereographic projection also induces a one-to-many mapping between points on S^2 and points in E^3 . The main advantage of the generalized stereographic projection is that its definition no longer depends on the choice of a particular center of projection; hence, the relationship between a rational curve/surface on a quadric and its inverse image can be stated in a unified manner in the general case. For example, any degree $2m$ rational curve on S^2 is the image of a degree m rational curve in E^3 under the generalized stereographic projection; hence, any circle on S^2 is the image of a line in E^3 under the generalized stereographic projection. This property facilitates the study of rational curves on quadrics [9].

We first introduce the definition of the generalized stereographic projection for a general quadric in E^3 . Since an irreducible quadric is projectively equivalent to either the oval quadric $x^2 + y^2 + z^2 - w^2 = 0$, or the doubly ruled quadric $xy - wz = 0$, or the cylinder $x^2 + y^2 - w^2 = 0$, we just need to consider these three canonical cases. To make the notation consistent with that of [9], we will use homogeneous coordinates (p_0, p_1, p_2, p_3) to denote a parameter point \mathbf{p} in E^3 , the domain of a generalized stereographic projection. The affine coordinates of a finite point $\mathbf{p} = (p_0, p_1, p_2, p_3)$ is given by $(p_1/p_0, p_2/p_0, p_3/p_0)$, where $p_0 \neq 0$.

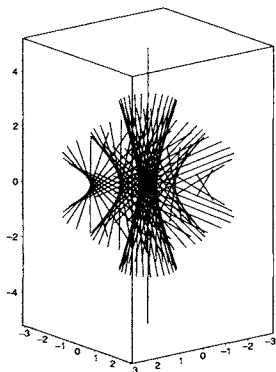


Figure 31.2. The projecting lines of the generalized stereographic projection.

The generalized stereographic projection of sphere $x^2 + y^2 + z^2 - w^2 = 0$ is

$$\begin{aligned} x &= 2p_0p_1 - 2p_2p_3, \\ y &= 2p_1p_3 + 2p_0p_2, \\ z &= p_1^2 + p_2^2 - p_0^2 - p_3^2, \\ w &= p_1^2 + p_2^2 + p_0^2 + p_3^2. \end{aligned}$$

The generalized stereographic projection of the hyperbolic paraboloid is

$$x = p_1p_2, \quad y = p_0p_2, \quad z = p_1p_3, \quad w = p_0p_3.$$

The generalized stereographic projection of the cylinder is

$$x = 2p_0p_1, \quad y = p_0^2 - p_1^2, \quad z = \text{arbitrary}, \quad w = p_0^2 + p_1^2.$$

Below we will only discuss the properties of the generalized stereographic projection for a sphere; the reader is referred to [9] for a more detailed discussion on the other cases. All the points on S^2 are images of a two-parameter family of lines in E^3 , called *projecting lines*, under the generalized stereographic projection. These lines are shown in Figure 31.2. Specifically, let $\mathbf{p} = (p_0, p_1, p_2, p_3)$ be an inverse image point of a point \mathbf{P} on S^2 , and denote $\mathbf{p}^\perp = (-p_3, p_2, -p_1, p_0)$. Then the projecting line that contains all inverse image points of \mathbf{P} is given by $\lambda\mathbf{p} + \mu\mathbf{p}^\perp$. Now define the *hyperbolic projection* by the mapping $E^3 \rightarrow E^2$:

$$h(\mathbf{p}) = (p_0^2 + p_3^2, p_0p_1 - p_2p_3, p_1p_3 + p_0p_2, 0).$$

It can be verified that the hyperbolic projection maps all points on a projecting line to the same point on plane $p_3 = 0$. Then the generalized stereographic projection of S^2 is the composition of the hyperbolic projection and the ordinary stereographic projection centered at the north pole of S^2 . Thus, all point on a projecting line are mapped to the same point on S^2 .

The key results about the generalized stereographic projection on sphere S^2 are the following [9,10]: under the generalized stereographic projection,

1. a degree $2m$ rational curve on S^2 is the image of a degree m rational curve in E^3 ;
2. a degree $(2m, 2n)$ rational tensor-product surface on S^2 is the image of a degree (m, n) rational tensor-product surface in E^3 ;
3. a degree $2n$ rational surface on S^2 is the image of a degree n rational surface in E^3 .

Similar properties also hold for the generalized stereographic projections for other quadrics. These properties of the generalized stereographic projection play a key role in constructing a rational curve interpolating given data points on a quadric and computing a bi-quadratic rational Bézier surface patch interpolating four given conic boundaries on a quadric [10].

31.2.3. Surface patches on quadrics

Using the stereographic projection described in the last section, a triangular patch on a quadric with boundary curves that are conic sections can be represented as a rational quartic surface patch [11,18]. To obtain this form, we start by using the inverse of a stereographic map to project the patch to a triangular region with conic boundaries on the parameter plane. Such a region can in turn be parameterized over a triangular domain with straight line boundaries. The combination of these two steps, each being a mapping of degree two, yields a rational quartic parameterization of the triangular patch on the quadric.

A natural question to ask is: what are the conditions for a triangular patch with conic boundaries on a quadric to be a triangular rational quadratic Bézier surface? If the quadric is nondegenerate, the conditions can be stated neatly [9,21]: a triangular patch on a nondegenerate quadric has a triangular rational quadratic Bézier representation if and only if the boundary curves of the patch are conic segments or line segments and there exists a point \mathbf{P} on the quadric but outside the triangular patch such that three planes each containing one of the three boundary curves are concurrent at \mathbf{P} . This works because the point \mathbf{P} can be used as the center of a stereographic projection to project the triangular patch into a triangle on the projection plane. Lü shows [24] that any triangular patch with conic boundaries on a cone or cylinder is a triangular rational quadratic Bézier surface, which is not necessarily faithful globally.

Rational tensor-product surface patches on quadrics have been studied in [10] using the generalized stereographic projection. Here we will only mention a particular result concerning bi-quadratic patches on a sphere; one may consult [10] for a more general discussion. Given four circular arcs forming the closed boundary of a four-sided region on S^2 , let \mathbf{P}_i be the four consecutive corner points of the region, $i = 1, 2, 3, 4$. See Figure 31.3. Each point \mathbf{P}_i is an intersection of the two circles that contain the two boundary segments meeting at \mathbf{P}_i ; let \mathbf{Q}_i denote the intersection of the two circles that is other than \mathbf{P}_i . It is shown in [10] that there exists a bi-quadratic rational Bézier surface on S^2 interpolating the given boundary if and only if $\mathbf{P}_1, \mathbf{P}_3, \mathbf{Q}_2$, and \mathbf{Q}_4 are on the same circle (*i.e.*, these points are coplanar) and the pair $\mathbf{P}_1, \mathbf{P}_3$ and the pair $\mathbf{Q}_2, \mathbf{Q}_4$ are not interleaved on that circle.

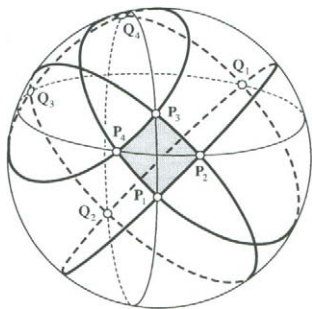


Figure 31.3. A rational bi-quadratic patch on a sphere.

Further studies on the construction of surface patches on quadrics can be found in the literature [4,10,14,35]. Given three homogeneous variables (r, s, t) , the Veronese surface is the 2D surface $(r^2, s^2, t^2, rs, st, st)$ embedded in P^5 , the five-dimensional projective space [16]. By treating a rational quadratic surface in P^3 as a projection of the Veronese surface from P^5 , Albrecht [1] devises a procedure to determine whether a rational quadratic surface is a quadric surface. Determining if a faithful rational quadratic surface is a quadric can also be carried out by generating the implicit equation of the surface via elimination theory and then examining the degree of the implicit equation.

31.3. FITTING, BLENDING, AND OFFSETTING

In this section we discuss the properties of quadrics concerning the following applications in geometric modeling: fitting, blending, and offsetting. By fitting we mean arranging a collection of surface patches with a certain degree of geometric continuity to pass through an array of data points in 3D space. Blending refers to the smooth joining of quadric surfaces by some other simple surfaces, such as part of a cyclide or a low-degree algebraic surface. The low algebraic degree of quadrics makes them valuable for modeling the boundary of 3D objects because it is then relatively easy to check whether a given point lies inside such an object or to perform fast ray-tracing rendering of quadrics. We will also examine some results about the self-intersection and rationality of offset surfaces to quadrics.

31.3.1. Fitting

The application of quadric surfaces to data fitting originated in the use of quadratic functions to fit data points sampled from a bivariate function [33]. Suppose that a bivariate function $f(x, y)$ is sampled at a collection of points (x_i, y_i) , and the function values and gradients $(f(x_i, y_i), \nabla f(x_i, y_i))$ are extracted and associated with (x_i, y_i) . Suppose also that there is a triangulation of the data points (x_i, y_i) . Powell and Sabin [33] study the problem of using quadratic functions to interpolate the data points over each triangle to give a smooth approximation of $f(x, y)$ across the entire domain. Since the graph of a quadratic function is either a paraboloid or a parabolic cylinder, this problem is a special

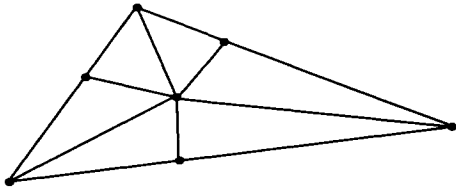


Figure 31.4. Subdivision of a triangle into six smaller triangles.

case of surface modeling with quadric patches.

Each quadratic function has six free parameters, but the data points at the vertices of a triangle impose nine constraints; therefore it is, in general, impossible to use a single quadratic function to interpolate over a triangle. A method is presented in [33] that uses a C^1 piecewise quadratic function over each triangle which is subdivided into six smaller triangles by connecting each vertex and a point on each side of the triangle to an interior point (see Figure 31.4). Six quadratic functions over the six sub-triangles can be constructed to form a piecewise C^1 function over the triangle and this produces a global C^1 approximation of the function $f(x, y)$ over a triangulation of the data points (x_i, y_i) .

Subsequent schemes using triangular quadric patches for scattered data interpolation in 3D space adopt a similar approach using more than one triangular quadric patch to cover each triangle domain, to overcome the insufficient degrees of freedom of a single quadric surface. A collection of triangular quadric patches joined with G^1 continuity over a properly subdivided triangular domain is called a *macro patch*. The problem of using triangular quadric patches to fit 3D data, while also matching specified normal vectors at the data points, was first solved by Dahmen [8], and later independently, by Guo [15], both using an algebraic approach. Their conclusion is that, given a triangle with positions and normal vectors specified at the three vertices, one can construct a macro patch consisting of six quadric triangles to interpolate the given data. However, when this scheme is used over a triangulated polyhedral surface formed by scattered data points in 3D space, additional quadric triangles are needed to join macro-patches over adjacent triangles with G^1 continuity. A geometric description of this scheme has recently been provided by Bangert and Prautzsch [3].

31.3.2. Blending

The problem of blending quadric surfaces is that of finding a surface, called a *blend*, which joins different quadric surfaces, called *primary surfaces*, with a certain degree of continuity. Let an algebraic surface be the zero-set, denoted $S(f)$, of a polynomial $f(x, y, z)$. Let $S(f, h)$ denote the intersection curve between a primary surface $S(f)$ and a *clipping surface* $S(h)$.

Given two quadric primary surfaces $S(f_1)$ and $S(f_2)$, and two clipping surfaces $S(h_1)$ and $S(h_2)$, two curves $S(f_1, h_1)$ and $S(f_2, h_2)$ can be defined on $S(f_1)$ and $S(f_2)$, respectively. The problem of blending $S(f_1)$ and $S(f_2)$ then becomes that of finding a surface $S(g)$ which is tangential to or has higher-order contact with $S(f_1)$ and $S(f_2)$ along the

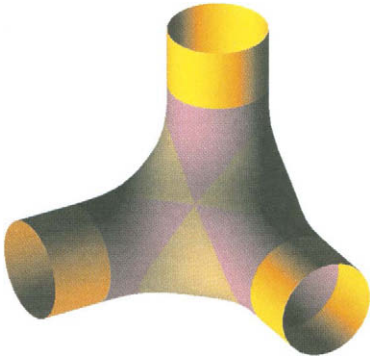


Figure 31.5. A G^2 blend for three cylinders by a piecewise quartic surface (Image courtesy of Falai Chen).

curves $S(f_1, h_1)$ and $S(f_2, h_2)$, respectively.

Hoffmann and Hopcroft [17] present a method for surface blending using potential functions. Given two primary surfaces $S(f_1)$ and $S(f_2)$, consider two families of surfaces $S(f_1 - s)$ and $S(f_2 - t)$, with parameters $s \in [0, \bar{s}]$ and $t \in [0, \bar{t}]$. Let $h_1 = f_2 - \bar{t}$ and $h_2 = f_1 - \bar{s}$. Then a blend surface of $S(f_1)$ and $S(f_2)$ can be defined by the algebraic surface swept out by the space curve $S(f_1 - s, f_2 - t)$, where s and t satisfy some equation $F(s, t) = 0$, and the curve $F(s, t) = 0$ is tangential to the s -axis at $(\bar{s}, 0)$ and the t -axis at $(0, \bar{t})$ in the s - t domain to ensure tangency between the blend surface and the primary surfaces. It is recommended [17] that $F(s, t) = 0$ should be a conic, such as an ellipse, to yield a low-degree blend surface. In this case the blend surface has degree four if $S(f_1)$ and $S(f_2)$ are quadrics.

Warren [46] studies the blending of general algebraic surfaces using the ideal theory of polynomials, and shows that the surface $S(g)$ that is tangential to $S(f)$ along the curve $S(f, h)$ has a special form, *i.e.*, $g \in I(f, h^2)$, the ideal generated by f and h^2 . Using this result, Warren obtains quartic blending surfaces for two quadrics, and surfaces of degree six for three-way blending of three quadrics.

Wu and Zhou [48] report a method that produces a degree $n + 1$ surface for n -way blending of n quadrics, under the condition that the n clipping surfaces are planes that intersect the n primary surfaces in n conics that are contained in a common quadric. C. Chen, F. Chen, and Y. Feng [6] propose a method that uses low degree piecewise algebraic surfaces to blend multiple quadrics. See Figure 31.5 for three cylinders joined by a G^2 blend that is a piecewise quartic surface produced by their method.

There are also other approaches to blending quadrics. Shene [40] uses cyclidal surfaces to blend cones. Wallner and Pottmann [42] use rational surfaces to blend general quadrics.

31.3.3. Offsetting

An *offset surface*, or *offset* in short, of a given surface S is the set of points that have a constant distance d to surface S ; surface S is also called the *progenitor surface*. For a point \mathbf{q} on the offset surface, the corresponding point \mathbf{p} on the progenitor surface that gives rise to \mathbf{q} is called the *foot point* of \mathbf{q} . The offset surface is used in NC machining where a given surface S is milled by a spherical-end cutter of radius d with its center following a path on the offset surface of distance d to surface S . Other applications of offsetting are discussed in [34].

The offset surfaces of general surfaces have been extensively studied; see [31], for example. Here we will only discuss some results concerning the offsets of quadric surfaces in the following two aspects: 1) the self-intersection of an offset surface to a quadric; and 2) the rational parameterization of an offset surface to a quadric.

The following discussions concerning offsets will take place in E^3 , the three-dimensional Euclidean space. We assume that a progenitor surface under consideration is regular and at least twice differentiable; these assumptions are clearly satisfied by nondegenerate quadrics, cylinders, and cones, except at the apex of a cone. By convention, one side of a surface S is assumed to be outside and the other side to be inside, and we assume that the normal vector \mathbf{n} at a point \mathbf{p} of S always points outside. An offset surface of distance d to progenitor surface S lies outside of S if $d > 0$, and inside if $d < 0$. A principal curvature at point \mathbf{p} of surface S is assumed to be positive if its center of curvature lies on the opposite side of surface S as pointed to by normal vector \mathbf{n} ; otherwise, the principal curvature is negative. Let k_{\min} and k_{\max} be the minimum and maximum principal curvatures, respectively, at point \mathbf{p} of surface S . When $d > 0$ and $k_{\min} < 0$, the offset surface of distance d has self-intersection due to the local concavity of the surface if $d > -1/k_{\min}$; when $d < 0$ and $k_{\max} > 0$, the offset surface of distance d has self-intersection if $d < -1/k_{\max}$. We will not discuss here the self-intersection of an offset surface due to the global geometry of the progenitor surface, but refer the reader to the more general treatment in [2].

The computation of offset surfaces of natural quadrics, including spheres, circular cones and cylinders, is given in detail by Farouki [12]. The self-intersection problem of an offset surface to a general quadric is studied by Maekawa in two companion papers [25,26]. The first paper [25] considers the offsets to special quadrics that can be represented by the graph of a quadratic bi-variate function $z = f(x, y)$; these quadrics are the elliptic paraboloid, parabolic cylinder, and hyperbolic paraboloid, which are all the irreducible quadrics tangential to the plane at infinity. It is shown that the self-intersection curve of an offset to such a special quadric is always a segment of a parabola, and this curve degenerates to a straight line in the case of the self-intersecting offset of a parabolic cylinder. When self-intersection occurs, let us call by *foot-point curve* the curve consisting of the foot points corresponding to the points of the self-intersecting curve on the offset surface; then, in this case, the foot-point curve is the intersection between the progenitor surface and an elliptic cylinder when the progenitor surface is a paraboloid, and the projection of the foot-point curve on the $z = 0$ plane is an ellipse. The foot-point curve consists of two lines for a self-intersecting offset surface to a parabolic cylinder. Since a surface S can be approximated to the third order by a quadratic paraboloid or a parabolic cylinder in a neighborhood of a regular point of S , the above results, together with a

surface curvature analysis, lead to an effective method for detecting the self-intersection, especially small loops, of an offset surface to a general surface, as well as a means to compute an accurate initial point on a self-intersection curve [25].

In the second paper [26] Maekawa studies the self-intersection problem of the offset to a general quadric surface and obtains the following results. The intersection curve of a self-intersecting offset surface to a quadric is, in general, a conic; for example, it is an ellipse for an ellipsoid, an ellipse or hyperbola for a hyperboloid or a cone, depending on the sign of offset distance d , and comprises straight lines for a cylinder. Furthermore, when the progenitor surface S is a non-degenerate quadric or a cone, the foot-point curve corresponding to the self-intersection curve on the offset is the intersection curve of S with an ellipsoid concentric with S ; when the progenitor surface S is a cylinder, the foot-point curve corresponding to the self-intersection curve comprises two straight lines that are the intersection between S and an elliptic cylinder. These results are highly useful in understanding and computing the self-intersection of an offset surface to a general quadric.

Next we consider the problem of representing the offset surface to a quadric as a rational surface. Although it is known that any offset surface to a quadric is an algebraic surface, the problem of determining whether such an offset surface is a rational surface has been studied only recently. Lü shows [22] that the offsets to paraboloids and one-sheet hyperboloids are rational, by exploiting the fact that a cubic algebraic surface, except for a cubic cone or cubic cylinder, is a rational surface. Using the same idea, Lü further shows [24] that the offsets to ellipsoids and two-sheet hyperboloids are also rational. It is proven by Pottmann, Lü, and Ravani [32] that the offset to a rational non-developable ruled surface is rational; hence, it also follows from this result that the offsets to a hyperbolic paraboloid and one-sheet hyperboloid are rational.

Although, according to the above results, the offset surface to a non-degenerate quadric is rational, the offset to a properly degenerate quadric, *i.e.*, a cylinder or a cone, is, in general, not rational; the exceptions are parabolic cylinders, circular cones, and circular cylinders, since offset surfaces to these quadrics are easily seen to be rational. These results follow from that the planar offset curve to a hyperbola or an ellipse, except for a circle, is not rational, and that the planar offset curve to a parabola is rational [22,23].

31.4. INTERSECTION AND INTERFERENCE

31.4.1. Computation of intersection curves

Computing the intersection of two quadrics requires deriving an expression for their intersection curve and determining the topological structure of it. Algorithms for intersecting quadric surfaces can be classified into those taking a geometric approach and those taking an algebraic approach. The geometric approach is numerically more stable but is generally limited to natural quadrics [27,28,39]. Below we will confine ourselves to reviewing some algebraic methods developed for general quadrics.

We will refer to the intersection curve of two quadric surfaces as a *QSIC*. A QSIC is a space curve of degree four. When it is nonsingular, a QSIC can have zero, one, or two disjoint connected components in 3D real projective space, and such a curve does not have a rational parameterization. A nonsingular QSIC is always irreducible. A singular

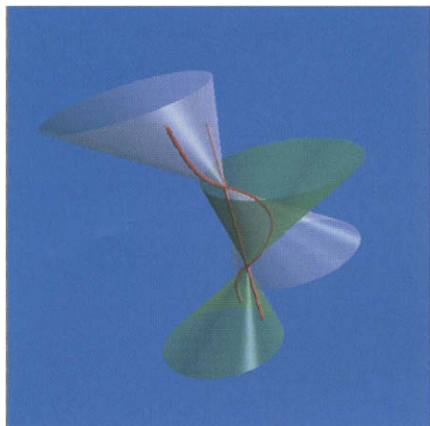


Figure 31.6. Two cones intersecting in a space cubic curve and a line.

QSIC can be reducible or irreducible. A singular but irreducible QSIC has exactly one singular point, which is a cusp, an acnode, or a crunode, and such a QSIC is a rational quartic curve. A reducible QSIC consists of two or more curves of lower degrees that sum to four; for example, in Figure 31.6 two cones intersect in a space cubic curve and a line.

Using the *Segre characteristic*, Bromwich gives a complete classification of degenerate QSICs in complex projective space, in terms of singularity and reducibility [5,41]; for two quadrics $S_0 : X^T A X = 0$ and $S_1 : X^T B X = 0$, the Segre characteristic is defined by the invariant factors of the quadratic form $X^T (\lambda A + \mu B) X$. However, these results need to be interpreted in real projective space in order to be useful to CAGD application.

Levin [19,20] proposes the following algebraic method for computing a QSIC. Let $S_0 : X^T A X = 0$ and $S_1 : X^T B X = 0$ be two distinct quadrics. Levin observes that there always exists a ruled quadric in the pencil $X^T (\lambda A + \mu B) X = 0$; S_0 and S_1 are first transformed to simpler forms simultaneously by an affine transformation to facilitate selecting the ruled quadric. This ruled quadric, called a *parameterization surface*, is parameterized by

$$S(u, v) = R(u) + vT(u),$$

where $R(u)$ is the base curve of the ruling and $T(u)$ is the direction vector of the generating line passing through $R(u)$. Substituting $S(u, v)$ into either S_0 or S_1 , one can solve for v in terms of u to obtain a parameterization of the QSIC of S_0 and S_1 of the form

$$Q(u) = \tilde{R}(u) \pm \sqrt{d(u)} \tilde{T}(u), \quad (31.1)$$

where $d(u)$ is a quartic polynomial, and $\tilde{R}(u)$ and $\tilde{T}(u)$ are some vector functions. Only those values of u for which $d(u) \geq 0$ give rise to real points on the QSIC.

Levin's method is useful for tracing and rendering a QSIC, but does not provide information about the singularity and the structure of the QSIC. In some cases the parameterization computed with Levin's method may not be appropriate; for instance, when

a QSIC is singular or reducible, the method still generates a parameterization involving a square root, although the QSIC has a rational parameterization in this case. Levin's method was later refined and implemented by Sarraga [36] and also revised and extended by Wilf and Manor [47].

Similar to Bromwich's work [5], but by studying the eigenvalues and the generalized eigenspaces of the system $AB - \lambda I$, Ocken, Schwartz, and Sharir show [30] that two quadrics $X^T A X = 0$ and $X^T B X = 0$ can be converted simultaneously by a projective transformation into two canonical forms whose intersection curve can be determined rather easily. The merit of this approach is that a projective transformation is used to convert the pair of input quadrics into forms that are simpler than what are obtained by Levin's method using an affine transformation. However, the link between the algebraic structure of the eigenspaces and the type of singularity or reducibility of the intersection curve is not discussed fully in [30]. Furthermore, the procedures presented there for classifying and computing the intersection curve are not thoroughly analyzed; for instance, the case of two quadrics intersecting in a line and a space cubic curve is not accounted for, and an intersecting curve having two singular points is listed in one of the generic cases and parameterized using a square-root function, although such a curve is reducible and thus comprises a collection of rational curves. These defects come as no surprise since the authors of [30] seem to be unaware of the classical results by Bromwich based on the Segre characteristic and their results on quadrics in 3D real projective space do not reach the same level of depth as attained by Bromwich [5]. However, it is envisioned that, with a thorough analysis and combining Bromwich's results, the ideas of Ocken, Schwartz, and Sharir can be further pursued to yield a reliable and complete algorithm for computing QSICs for practical applications.

Farouki, Neff and O'Connor [13] present an alternative algebraic method that uses rational arithmetic to analyze degenerate QSICs. The degeneracy of a QSIC is detected by testing whether the discriminant of the characteristic equation $\det(\lambda A + \mu B) = 0$ is zero; the discriminant in this case is the resultant of $f(\lambda) \equiv \det(\lambda A + \mu B)$ and its derivative $f'(\lambda)$. When the QSIC is found to be degenerate, a quartic *projection cone* is derived. The reducibility of the QSIC is then determined by factorizing the quartic projection cone.

Wilf and Manor [47] extend Levin's method to classify a general QSIC as well as to find its expression. To classify a QSIC, the roots of the characteristic equation are obtained numerically, and then the Segre characteristic is computed to guide the parameterization of the QSIC, utilizing Levin's method. However, this method cannot compute the number of disjoint connected components of a nonsingular QSIC in real projective space, since this information is not provided by the Segre characteristic.

By exploring a birational mapping between a QSIC and a planar cubic curve under stereographic projection with an appropriately chosen center of projection, Wang, Joe, and Goldman [44] develop a method for classifying and computing the QSIC of two general quadrics, through a topological analysis and parameterization of the planar cubic curve. This method produces complete structural information of a QSIC, including the reducibility, the type of singularity, and the number of disjoint connected components of a QSIC in 3D real projective space.

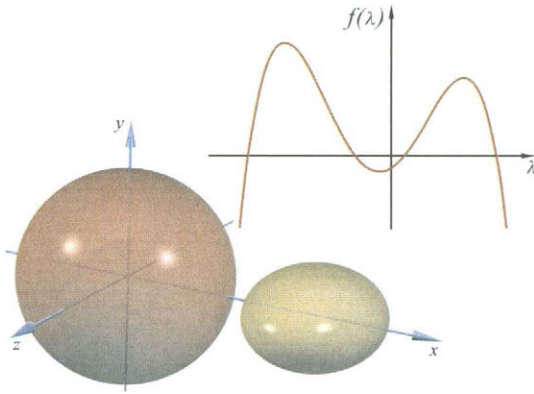


Figure 31.7. Two separate ellipsoids and their characteristic polynomial.

31.4.2. Detecting interference

Interference analysis is about detecting whether two quadrics interfere or intersect with each other. Such problems can be solved by resorting to a general method for computing the intersection curve of two quadrics; however, in interference analysis one is primarily interested in detecting the existence of real intersection points, and thus an expression for the intersection curve is not required. Therefore more efficient methods can be devised for interference analysis.

In light of this, an algebraic condition for the separation of two ellipsoids is proved in [45]. Given two ellipsoids $X^T A X = 0$ and $X^T B X = 0$, with A and B normalized so that their first 3×3 minors are positive, it is shown [45] that the characteristic equation $\det(\lambda A + B) = 0$ has at least two real negative roots, and that the ellipsoids are separated by a plane if and only if their characteristic equation has two distinct real positive roots (see Figure 31.7). Furthermore, the ellipsoids touch each other externally if and only if the characteristic equation has a real positive double root. One advantage of this characterization is that only the signs of the roots matter, rather than their exact values.

This approach to interference analysis is related to the Segre characteristic for classifying a degenerate intersection of two quadrics in complex projective space. More research is expected to obtain conditions for discriminating other configurations of two ellipsoids, such as intersection and containment, as well as to perform interference analysis for other quadric surfaces.

31.5. ACKNOWLEDGMENTS

I would like to thank Barry Joe, Ron Goldman, Bert Jüttler, and Myung-Soo Kim for their helpful comments and assistances in writing this chapter. This work has been supported in part by a grant from the Research Grant Council of HKSAR.

REFERENCES

1. G. Albrecht. Determination and classification of triangular quadric patches. *Computer Aided Geometric Design*, 15:675–697, 1998.
2. S. Aomura and T. Uehara. Self-intersection of an offset surface. *Computer-Aided Design*, 22(7):417–422, 1990.
3. C. Bangert and H. Prautzsch. Quadric splines. *Computer Aided Geometric Design*, 16:497–515, 1999.
4. W. Boehm and D. Hansford. Bézier patches on quadrics. In Farin, editor, *NURBS Curve and Surface Design*, pages 1–14, 1991.
5. T.J. Bromwich. Quadratic forms and their classification by means of invariant-factors. *Cambridge Tracts in Mathematics and Mathematical Physics*, No. 3, 1906.
6. C. Chen, F. Chen, and Y. Feng. Blending quadric surfaces with piecewise algebraic surfaces. To appear in *Graphical Models*.
7. E.W. Chionh and R.N. Goldman. Degree, multiplicity, and inversion formulas for rational surfaces using μ -resultants. *Computer Aided Geometric Design*, 9:93–108, 1992.
8. W. Dahmen. Smooth piecewise quadric surfaces. In T. Lyche, and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 181–194. Academic Press, 1989.
9. R. Dietz, J. Hoschek, and B. Jüttler. An algebraic approach to curves and surfaces on the sphere and on other quadrics. *Computer Aided Geometric Design*, 10:211–229, 1993.
10. R. Dietz, J. Hoschek, and B. Jüttler. Rational patches on quadric surfaces. *Computer-Aided Design*, 27:27–40, 1995.
11. G. Farin. *NURBS: From Projective Geometry to Practical Use*. A.K. Peters, Natick, Massachusetts, 1999.
12. R.T. Farouki. Exact offsets procedures for simple solids. *Computer Aided Geometric Design*, 2:257–279, 1985.
13. R.T. Farouki, C.A. Neff, and M.A. O'Connor. Automatic parsing of degenerate quadric-surface intersections. *ACM Transactions on Graphics*, 8(3):174–203, 1989.
14. G. Geise and U. Langbecker. Finite quadric segments with four conic boundary curves. *Computer Aided Geometric Design*, 7:141–150, 1990.
15. B. Guo. Quadric and cubic bitetrahedral patches. *The Visual Computer*, 11:253–262, 1991.
16. J. Harris. *Algebraic Geometry – A First Course*. Springer-Verlag, New York, 1992.
17. C. Hoffmann and J. Hopcroft. Quadratic blending surfaces. *Computer-Aided Design*, 18(6):301–306, 1986.
18. B. Joe and W. Wang. Reparameterization of rational triangular Bézier surfaces. *Computer Aided Geometric Design*, 11:345–361, 1994.
19. J.Z. Levin. A parametric algorithm for drawing pictures of solid objects composed of quadrics. *Communications of the ACM*, 19(10):555–563, Oct. 1976.
20. J.Z. Levin. Mathematical models for determining the intersections of quadric surfaces. *Comput. Graph. Image Process.*, 1:73–87, 1979.
21. S. Lodha and J. Warren. Bézier representation for quadric surface patches. *Computer-*

- Aided Design*, 22(9):574–579, 1990.
22. W. Lü. Rationality of the offsets to algebraic curves and surfaces. *Appl. Math.*, 9:265–278, Ser. B, 1994.
 23. W. Lü. Offset-rational parametric plane curves. *Computer Aided Geometric Design*, 12:601–616, 1995.
 24. W. Lü. Rational parameterization of quadrics and their offsets. *Computing*, 57:135–146, 1996.
 25. T. Maekawa. Self-intersection of offsets of quadratic surfaces: Part I, explicit surfaces. *Engineering with Computers*, 14:1–13, 1998.
 26. T. Maekawa. Self-intersection of offsets of quadratic surfaces: Part II, implicit surfaces. *Engineering with Computers*, 14:14–22, 1998.
 27. J.R. Miller. Geometric approaches to nonplanar quadric surface intersection curves. *ACM Transactions on Graphics*, 6:274–307, October 1987.
 28. J.R. Miller and R.N. Goldman. Geometric algorithms for detecting and calculating all conic sections in the intersection of any two natural quadric surfaces. *Graphical Models and Image Processing*, 57(1):55–66, 1995.
 29. L. Minsky. *An Introduction to Linear Algebra*. Oxford University Press, 1962.
 30. S. Ocken, J.T. Schwartz, and M. Sharir. Precise implementation of CAD primitives using rational parameterizations of standard surfaces. In J.T. Schwartz, M. Sharir, and J. Hopcroft, editors, *Planning, Geometry, and Complexity of Robot Motion*, pages 245–266. Alex Publishing Corporation, New Jersey, 1987.
 31. B. Pham. Offset curves and surfaces: a brief survey. *Computer-Aided Design*, 24(4):223–229, 1992.
 32. H. Pottmann, W. Lü, and B. Ravani. Rational ruled surfaces and their offsets. *Graphical Models and Image Processing*, 58(6):544–552, 1996.
 33. M.J.D. Powell and M.A. Sabin. Piecewise quadratic approximation on triangles. *ACM Transactions on Mathematical Software*, 3(4):316–325, 1977.
 34. J.R. Rossignac and A.A.G. Requicha. Offsetting operations in solid modeling. *Computer Aided Geometric Design*, 3:129–148, 1986.
 35. J. Sanchez-Reyes and M. Paluszny. Weighted radial displacement: A geometric look at Bézier conics and quadrics. *Computer Aided Geometric Design*, 17:267–289, 2000.
 36. R.F. Sarraga. Algebraic methods for intersections of quadric surfaces in GMSOLID. *Computer Vision, Graphics and Image Processing*, 22(2):222–238, 1983.
 37. T.W. Sederberg and D.C. Anderson. Steiner surface patches. *IEEE Computer Graphics and Applications*, pages 23–36, May 1985.
 38. J.G. Semple and G.T. Kneebone. *Algebraic Projective Geometry*. Oxford University Press, 1952.
 39. C.K. Shene and J.K. Johnstone. On the lower degree intersections of two natural quadrics. *ACM Transactions on Graphics*, 13(4):400–424, 1994.
 40. C.K. Shene. Blending two cones with Dupin cyclides. *Computer Aided Geometric Design*, 15:643–673, 1998.
 41. D.M.Y. Sommerville. *Analytical Geometry of Three Dimensions*, Cambridge University Press, 1947.
 42. J. Wallner and H. Pottmann. Rational blending surfaces between quadrics. *Computer Aided Geometric Design*, 14:407–419, 1997.

43. W. Wang, B. Joe, and R.N. Goldman. Rational quadratic parameterizations of quadrics. *International Journal of Computational Geometry and Applications*, 7(6):599–619, 1997.
44. W. Wang, B. Joe, and R.N. Goldman. Computing quadric surface intersection based on an analysis of planar cubic curves. preprint, 2001.
45. W. Wang, J.Y. Wang, and M.-S. Kim. An algebraic condition on the separation of two ellipsoids. *Computer Aided Geometric Design*, 18:531–539, 2001.
46. J. Warren. Blending algebraic surfaces. *ACM Transactions on Graphics*, 8(4):263–278, 1989.
47. I. Wilf and Y. Manor. Quadric surface intersection: shape and structure. *Computer-Aided Design*, 25(10):633–643, 1993.
48. T.R. Wu and Y.S. Zhou, On blending several quadratic algebraic surfaces. *Computer Aided Geometric Design*, 17:759–766, 2000.

Index

- Δ^1 triangulation, 707
- Δ^2 triangulations, 707
- α -shape, 655
- μ -basis, 373
- k -jet, 199
- 2-stage de Casteljau evaluation, 93
- 3-stage de Casteljau evaluation, 93
- 4-point subdivision, 312

- A-frame lemma, 27
- A-frame, affine, 27
- A-frame, affine representation of the projective, 29
- acnode, 790
- adaptive forward differencing, 752
- additively weighted Voronoi diagram, 441
- adjacency graph, 673
- adjacent isoparametric curves, 754
- affine combination, 24–26
- affine coordinates, 24
- affine coordinates, new, 25
- affine independence, 23
- affine invariance, 112, 120, 127, 131, 168
 - box spline representation, 261
- affine map, 25, 78, 92, 99
- affine parameter transformation, 79, 178, 628
- affine representation of the projective A-frame, 29, 34
- affine scale, 26
- affine space, 23
- affine subspace, 24
- affine system, 23
- AIC coverage, 756
- Aitken's algorithm, 167
- Alfeld, P., 11
- algebraic fitting, 665
- algebraic number, 625

- algebraic quadratic curve, 373
- algebraically-rectifiable curve, 421
- algorithm
 - boundary evaluation, 496
 - Boundary-to-CSG conversion, 497
 - comparison, 493
 - CSG-to-boundary conversion, 496
 - de Casteljau, 115
 - distance computation, 493
 - enabling, 494
 - fundamental, 491
 - mass properties, 498
 - ordering, 493
 - planning and generation, 500
 - PMC, 492
 - point generation, 492
 - polygonization, 495
 - ray casting, 494
 - rendering, 498
 - sampling, 495, 498, 502
 - SMC, 495
- algorithmic complexity, 714
- almost interpolation, 711
- alpha-shape, 655
- analysis
 - of artifacts, 323
- analysis of subdivision support, 315
- analytic surface, 665
- angle of two vectors, 30
- animation, 723, 741
- anticaustic, 420
- antiderivative, 154
- approximation, 712
 - finite element, 301
 - order, 701
 - point data, 170

- shape equations, 172
- spline, 287
 - tensor product, 182
- approximation order, 711
- arbitrary topology, 313
- arc element of a curve, 38
- arc element on a surface, 40
- arc-length, 38, 211, 766
- arc-length function, 407, 420
- arc-length parameterization, 406
- area
 - signed, 86
- Argyris, 283, 291
- ART-algorithm, 289
- artifacts, 323
- artifacts from subdivision, 320
- assembly-centric design, 536
- asymptotic directions, 41
- auxiliary element, 669
- axis of a quadric, 32
- axonometric image, 26

- B stands for basis, 144
- B-form, 150
- B-rep
 - model building, 673
- B-spline, 284, 714
 - cardinal, 285
 - expansion, 715
 - extended, 292
 - inner, 287, 291
 - multivariate, 285
 - normalized, 119
 - outer, 287, 291
 - relevant, 287, 291
 - subdivision, 288
 - tensor product, 284
 - uniform, 285
 - weighted extended, 292
- B-spline curve
 - rational, 118
 - reparameterization, 122
- B-spline defined, 141
- B-spline expansion, 148
- B-spline patch
 - rational, 131
- B-spline recurrence, 142
- Babuska, 284
- Ball, A., 12
- Barnhill, R., 1, 11, 15
- Barsky, B., 14
- barycentric
 - mapping, 660
- barycentric combination, 82, 101, 168
- barycentric coordinates, 11, 24, 33, 97, 702
- base point, 381, 383, 781
- base surface, 659
- basic interval, 145
- basic length unit, 418
- basis, 707
 - Bernstein, 363
 - ideal, 373
 - power, 363
- basis construction, 302
- basis conversion, 86
- basis function
 - Bernstein, 77
 - bivariate Bernstein, 100
 - Hermite, 87
 - monomial, 76
- BB-form, 147
- beautification, 653, 675, 676
- benchmark object, 663, 664, 671, 674
- bending energy, 179, 413
- Bernstein, 147
 - basis, 363
 - polynomials, 702
- Bernstein basis, 146
- Bernstein polynomial, 77, 81, 86
 - bivariate, 98, 100
 - properties, 82
- Bernstein-Bézier, 147
- Bernstein-Bézier coefficients, 702
- Bernstein-Bézier form, 407
- Bernstein-Bézier representation, 702
- Bernstein-Bézier techniques, 702, 706, 708, 712
- Bessel end condition, 177, 178
- Bézier clipping, 759

- Bézier control point, 77
- Bézier curve, 76
 - C^1 conditions, 88
 - circular arc, 587
 - de Casteljau algorithm, 80
 - degree elevation, 85
 - derivative, 83
 - dual, 47
 - interrogation, 85
 - piecewise, 87
 - properties, 77
 - rational, 44, 112
 - subdivision, 81
- Bézier line, 47
- Bézier patch, 90
 - C^1 conditions, 96
 - degree elevation, 95
 - derivative, 93
 - evaluation, 93
 - normal, 95
 - properties, 91, 98
 - subdivision, 96
 - tensor product, 92
 - tensor product property, 91
- Bézier patch, trivariate, 189
- Bézier point, 39, 45, 704
 - of box splines by masks, 267, 269
- Bézier polygon, 77
- Bézier representation
 - box splines, 266
 - of half-box splines, 277
- Bézier techniques
 - industry, 76
 - numerical stability, 75
- Bézier triangle, 97
 - C^1 conditions, 104
 - degree elevation, 103
 - derivative, 102
 - evaluation, 100
 - properties, 98
 - subdivision, 103
- Bézier, P., 1, 5, 188
- Bezout's
 - matrix, 368
 - resultant, 368, 371
 - theorem, 375, 381
- bi-degree, 381
- biarc, 8
- bicubic Hermite patches, 185
- bicubic subdivision, 314
- bifurcation point, 456
- biharmonic problem, 300
- bilinear interpolant, 92, 180
- bilinear patch, 89
- bilinear precision, 92
- biquadratic subdivision, 313
- bivariate
 - polynomials, 701
 - spline theory, 702
- bivariate Bernstein polynomial, 98, 100
 - properties, 101
- black and white, 713
- blend, 657, 663, 664, 672
 - free-form, 662
 - radius estimation, 672
 - reconstruction, 653
 - vertex, 672
- blend surface centre paths, 332
- blending, 588, 785, 786
 - cone/cone, 589
 - cone/cylinder, 590
 - construction, 591
 - one-sided, 588
 - parabolic cyclide, 589
 - supercyclides, 591
 - two-sided, 588
- blending algebraic surface, 384
- blending surface, 63
- Blinn, J., 15
- block-circulant matrices, 316
- blossom, 45, 75, 138, 152
- blossoming, 6
- Blutel surface, 576
- Boehm, W., 1, 5, 8, 10, 12, 14
- Boeing, 7, 10
- Böhm, 156
- Bolton, K., 8
- Boolean operation, 293
- Boolean operations
 - closure under, 476

- regularized, 477
- requirements for, 476
- border point, 636
- boundary
 - as a k -cycle, 479
 - combinatorial, 479
 - evaluation, 496
 - manifold, 480
 - merging, 496
 - representation
 - manifold, 489
 - non-manifold, 490
 - point classification, 489
 - properties, 488
 - winged-edge, 487, 489
 - vertices, 704
- boundary condition
 - essential, 296
 - higher order, 299
 - mixed, 299
 - natural, 298
- boundary curve, 90, 91, 756
- boundary representation, 520
- boundary-value problems, 701
- bounding box, 329
- box spline, 255, 257, 714
 - Bézier representation, 266
 - convexity preservation, 262
 - derivative, 259
 - geometric definition, 256
 - linear precision, 262
 - partition of unity, 261
 - properties of, 258
 - subdivision, 263, 264
 - convergence, 265
 - support, 258
 - center, 262
- box spline representation
 - affine invariance, 261
- box spline surface, 261
 - control points, 261
 - generalized, 269
 - minimal, 270
 - convex hull property, 261
 - derivative, 262
- box-splines, 312
- brain imaging, 189
- break sequence, 141
- Brianchon's theorem, 29
- BSP trees, 486
- Buchberger, 367
- butterfly algorithm, 12
- butterfly subdivision, 314
- C^1 conditions
 - Bézier curves, 88
 - Bézier patch, 96
 - Bézier triangle, 104
- C^1 spline
 - spaces, 702
- C^2 bicubic spline interpolation, 186
 - knot sequence, 187
- C^2 cubic spline
 - minimum property, 179
- C^2 cubic spline interpolation, 175
- CAD/CAM interface, 745
- CAGD, 702
- calculation, 683
- calibrating ball, 654
- canal surface, 56, 136, 411
- cardinal, 145
- cardinal data, 310, 312
- cardinal form, 169, 174
- carrier, of cell, 487
- Cartesian coordinates, 30
- Castelnuovo, 383
- CATIA, 6
- Catmull, Ed., 11
- Catmull-Clark subdivision, 314
- caustic, 420
- Cayley, 368
- Céa inequality, 301
- cell
 - carrier, 487
 - properties, 485
 - representations, 485
- cell complex
 - as a solid model, 478
 - incidence, 487
 - representations, 486

- type of cells, 478
- center of box spline support, 262
- center of projection, 781
- Chaikin's method, 309
- Chaikin, G., 11
- chain rule continuity, 37
- chains, algebraic topological, 479
- characteristic equation, 791, 792
- characteristic function, 141
- characteristic map, 318
 - second order, 318
- chart, 212
- Chebyshev-Demko points, 160
- Chiyokura, H., 9
- circle
 - fitting, 666, 667, 671
- circulant matrices, 316
- clamped end condition, 175
- clamped plate, 300
- Clark, J., 11
- class, 420
- classes of triangulations, 704
- classification
 - point, 481
 - set membership, 495
- Clifford algebra, 423
- Clough, 283
- Clough-Tocher, 291
 - split, 708, 712, 713
- CNC machining, 417
- coalescing points, 36
- cochains, algebraic topological, 509
- Cohen, E., 10
- collinear normal point, 640
- collocation matrix, 160
- coloring, 713
- combinations, projective, 34
- compatibility
 - twist, 206
- completeness, informational, 473, 502, 510
- complex number, 408
- complex representation, 408
- composite surfaces, 38
- computer
 - graphics, 683
 - concurrent engineering, 532
- condition, 151
- condition number, 297
- condition of the B-spline basis, 152
- cone, 28, 131, 664
 - fitting, 666, 667
 - splines, 707
- cone spline surface, 68
- conformal closure, 53
- conformal map, 408
- conic, 117, 374
 - classification, 118
 - complementary segment, 118
 - rho, 118
- conic section, 2, 117
- conjecture, 706
- conjugate gradient, 304
- conjugate net, 576
- conjugate pair of points, 29
- connection function, 134
- constant, 710
- constrained
 - fitting, 653, 668
- constraining, 522
- constraint, 653, 662, 665, 669, 674
 - detection, 675
 - equational, 522
 - geometric, 521, 670
 - inconsistent, 669, 670
 - priority, 669
 - satisfaction, 668, 669, 676
 - semantic, 522
 - topological, 522
- constraint problem
 - parametric, 523
 - variational, 523
- constraint solver, 522
 - constructive, 525
 - degrees of freedom analysis, 526
 - equational, 524
 - graph-based, 526
 - numerical, 524
 - propagation, 525
 - rule-based, 525
 - symbolic, 525

- constraints, 519
- constructive solid geometry, 520, 623
- contact arc, 456
- contact circle, 456
- contact component, 456
- contact element subdivision, 321
- contact of order r , 36
- contact point, 456
- continuity
 - β -matrix, 199
 - ε , 213
 - k -jet, 199
 - coordinate-degree, 203
 - total-degree, 203
 - $n + 1$ Tangent Theorem, 207
 - 3 Tangent Theorem, 207
 - affine space, 122
 - alternating sum constraint, 208
 - arc-length, 211
 - arc-length parametrization, 199
 - compatibility
 - twist, 206
 - composition, 198
 - connecting relation, 209
 - connection matrix, 199
 - constraint
 - compatibility, 206
 - composition, 204
 - vertex-enclosure, 204
 - constructions, 218
 - contact of order k , 200, 217
 - control net, 195
 - triangles, 195
 - coplanarity, 212
 - curvature, 122, 134
 - degree-optimal constructions, 200
 - derivatives, 194
 - differentiation
 - curve, 198
 - Faá di Bruno's Law, 198
 - surface, 198
 - domain, 200
 - enclose with, 202
 - Frenet frame, 200, 209, 210
 - function
 - coplanarity condition, 195
 - geometric, 199, 201
 - patch complex, 209
 - curve, 193
 - surface, 193
 - geometry map, 200
 - higher-order saddle point, 216, 220
 - homogeneous, 123
 - homogeneous coordinates, 213
 - implicit representation, 217
 - interpolation
 - network of curves, 203
 - meet with, 202
 - normal, 212
 - normal angle, 213
 - of curvature, 318
 - of higher derivatives, 320
 - parametric, 199
 - parametrically C^k , 201, 217
 - patch, 200, 217
 - projective space, 122
 - regularity, 194
 - reparametrization, 199, 200
 - subdivision, 217
 - tangent, 122, 194, 212
 - degree bound, 214
 - determinant criterion, 197
 - plane, 203
 - sector, 203
 - tangent plane, 133
 - torsion, 122
 - universal spline, 200
 - visual, 193
- continuity conditions, 415
- continuous
 - spline spaces, 704
- contour line, 37
- control mesh, 309
- control net, 91, 195
- control plane, 50
- control point, 156
 - homogeneous, 115, 121
- control polygon, 156
 - geometric, 46
- control polyhedron, 309

- control structure
 - affine, 737
 - dual, 48, 50, 740
 - intrinsic, 733, 734, 740
- controller, 418
- conventional object, 656, 663
- convex hull, 152, 430, 753, 760
- convex hull property, 112, 120, 128, 131, 168, 329, 738
 - box spline surfaces, 261
 - Bézier curve, 78
 - Bézier patch, 92
 - Bézier triangle, 99
- convexity
 - for box splines, 262
- Coons
 - surface, 659
- Coons patch, 180
- Coons, S., 3, 4, 7–10
- coordinate transformation, 724
- coordinates
 - Cartesian, 724
 - homogeneous, 724, 728
 - homogeneous quaternion, 729
 - measuring machine, 654
 - of rotations, 729
 - Plücker, 60, 668
- coordinates, affine, 24
- coordinates, barycentric, 24
- coordinates, homogeneous, 32, 35
- coordinates, new affine, 25
- coordinates, new Cartesian, 31
- coordinates, projective, 33
- coplanarity condition, 195
- core, 455
- corner cutting algorithm, 120
- corner point interpolation, 128, 131
- Courant, 283
- C^r smoothness, 703
- Cramer's rule, 370, 382
- Cremona transformation, 781
- cross boundary
 - derivative, 708
- cross ratio, 33
- cross-section, 335
- cross-sectioning, 327
- crosscut
 - partition, 707
- crosscut partitions, 711
- crunode, 790
- crystal Voronoi diagram, 446
- CSG (Constructive Solid Geometry), 482
- cubic C^1 splines, 706, 713
- cubic Hermite polynomial, 174
- cubic spline, 415
- cubic surface, 383
- Curry-Schoenberg Theorem, 150
- curvature, 14, 86, 179
 - plot, 86
 - signed, 86
 - surface, 663
- curvature center, 576
- curvature continuous interpolant, 125
- curvature lines, 576
- curvature of a curve, 39
- curvature radius, 576
- curvature sphere, 576
- curvature, normal, 41
- curvature, principal, 41
- curve, 155
 - B-spline
 - rational, 733
 - Bézier, 76, 729
 - rational, 732
 - spherical, 729
 - Bézier
 - dual, 47
 - rational, 44
 - coefficient, 76
 - degree elevation, 85
 - disk Bézier, 55
 - domain, 76
 - evaluation, 80
 - fitting, 668
 - free-form, 667
 - Hermite, 79
 - interrogation, 85
 - Minkowski Pythagorean-hodograph, 56
 - monomial, 76
 - normal

- rational, 44
- NURBS, 46
 - spherical, 63
- offset, 58
- on surface, 90
- parametric, 76
- Pythagorean-hodograph, 58, 744
- quadratic
 - shape defect, 220
- quaternion, 729
- rational
 - spherical, 734, 743
- rational Pythagorean-hodograph, 58
- rational spline, 735
- render, 81
- subdivision, 81
- thick, 668
- curve network segmentation, 657
- curve-plane intersection, 329
- curves and surfaces, osculating, 36
- curves and surfaces, polynomial, 34
- curves and surfaces, rational, 34
- cuspidal cubic, 421
- cycle, 50
- cyclide
 - Dupin, 57
 - generalized, 575
- cyclographic mapping, 51
- cylinder, 35, 131, 136, 184, 664, 665
 - fitting, 666
- DAC-I, 3
- data
 - rectangular structure, 181
- data capture, 652, 653
- data site, 437
- de Boor, 285
- de Boor algorithm, 121, 133
- de Boor point, 119, 715
- de Boor, C., 3, 9, 11
- de Casteljau, 152
 - recurrence, 704
- de Casteljau algorithm, 80, 115, 167, 630, 703, 729
 - Bézier patch, 93
 - Bézier triangle, 100
 - Bézier patch, 93
 - spherical, 729
- de Casteljau's Algorithm, 155
- de Casteljau, P., 10
- de Rham, G., 11, 309
- decimation, 652, 654, 657
- deformation, 188
- degeneracy, 430
- degenerate, 430
 - edge, 706
- degree *vs* order, 141
- degree elevation, 85
 - Bézier patch, 95
 - Bézier triangle, 103
 - repeated, 85
- degree lexicographical order, 365
- degree-optimal constructions, 200
- Delaunay, 431
 - diagram, 431
 - edge, 431
 - polygon, 431
 - tetrahedron, 672
 - triangulation, 431, 655
- density
 - of a shadow, 256
- derivative
 - of half-box splines, 273, 274
 - Bézier curve, 83
 - Bézier patch, 93
 - Bézier triangle, 102
 - directional, 102
 - of box spline surfaces, 262
 - partial, 94
- derivative of a spline, 153
- DeRose, T., 188
- design
 - automatic, 501, 510
 - constraints, 498
 - geometric, 498
 - interactive, 739
 - parametric, 498, 504
- DESIGNBASE, 9
- determining set, 206

- developable surface, 49, 762
- developable-surface-surface intersection, 764
- diameter, 710
- differential geometry
 - projective, 43
- diffuse lighting, 770
- dimension, 704
- dimension of an affine space, 23
- dimensional reduction, 498
- dimensionality filtering, 664
- direct ray-tracing, 759
- direct scan-conversion, 751
- direct segmentation, 663
- directional derivative, 102
- Dirichlet tessellation, 13
- discrete Coons patch, 180
- display, 333
 - ray casting, 327, 333
 - Z-buffer, 327, 333
- distance
 - of planes, 64
- distance function, 665, 669, 671
 - approximation, 666
- distance of points, 30
- division by zero, 153
- Dixon's
 - resultant, 369, 382
- Dokken, T., 15
- domain
 - Bézier curve, 76
 - Bézier patch, 88
 - Bézier triangle, 97
 - periodic, 214
- domain decomposition lemma, 458
- Doo, D., 11
- Doo-Sabin subdivision, 313
- dot product, 30
- double Blutel surface, 576, 583
- double point, 374
- double tracing, 421
- draft angle, 675
- dual control structure, 48, 50
- dual function, 295
- dual functional, 151
- dual representation, 47
- dual subdivision, 321
- duality, 36, 47
- Duchon, J., 12
- dull corner, 456
- Dupin cyclide, 57, 575
 - Bézier representation, 587
 - blending, 588
 - fourth order, 582
 - general, 578
 - mid point curve, 577
 - parabolic, 578
 - parameter representation, 579
 - radius function, 577
 - ring cyclide, 580
 - spindle cyclide, 580
 - symmetry plane, 577
- Dupin's indicatrix, 41
- Dyn, N., 12
- edge
 - detection, 653
 - swapping, 656
- edge swap, 712
- efficient representation, 671
- eigenspace, 791
- elasticity, 299
- elimination theory, 367
- ellipse, 118
- elliptic geometry, 729
- elliptic-distance Voronoi diagram, 445
- end condition, 415
 - Bessel, 177, 178
 - clamped, 175
 - natural, 177
 - Not-a-knot, 178
- endpoint interpolation, 77, 119
- engineer, 683
- envelopes, 740
 - approximate computation of, 738
- enveloping sphere, 577
- equi-angularity, 433
- error estimate, 301
- Euclid's algorithm, 738
- Euclidean metric, 421
- Euclidean motion, 31

- Euclidean space, 30
- Euler
 - characteristic, 480, 493
 - operators, 493
- Euler parameter, 726, 728
- Euler's formulas, 705
- Euler's method, 368
- Euler's theorem, 41
- Eulerian angles, 31
- evaluation
 - Bézier curve, 80
 - Bézier patch, 93
 - Bézier triangle, 100
- evaluation algorithm, 155
- evaluation of a spline, 154
- evolute, 420
- experiment, 683
- extraordinary points, 316
 - limit position, 317
- extrapolation, 78, 81
- extrusion, 653, 664, 666, 667
- extrusion surfaces, 759

- fairing, 658, 659
- faithful parameterization, 781
- faithful representation, 665
- Farin point, 113, 128
 - extended, 128
- Farin, G., 10, 11
- Farouki, R., 15
- fattening, 673
- Faux, I., 14
- feature, 519
 - automatic recognition, 528
 - classes, 532
 - custom libraries, 532
 - dependent, 658
 - detection, 675
 - fitting, 662
 - geometric attributes, 527
 - geometric features, 528
 - instances, 532
 - interactive definition, 528
 - model, 527
 - multiple views, 532
 - nongeometric features, 528
 - recognition, 675
 - surface, 653, 662
 - technological attributes, 527
- feature recognition, 501
- feedrate, 418
- Ferguson, J., 3, 7–9
- fiber bundles, 509
- field modeling, 508
- fillet surface centre paths, 332
- filtering
 - dimensionality, 664
 - planarity, 664
- fine
 - triangulation, 710
- finite element, 702, 708
 - approximation, 301
 - matrix assembly, 302
 - mesh, 290
 - standard, 283, 297
 - triangular, 291
 - web-spline, 292
 - weighted, 284, 289
- fitting, 785
 - algebraic, 665
 - circle, 666, 667, 671
 - cone, 666, 667
 - constrained, 662, 668
 - curve, 667, 668
 - cylinder, 666
 - feature, 662
 - free-form, 659
 - geometric, 665
 - global, 662
 - sphere, 666
 - surface, 653, 665
 - torus, 666, 667
- floating point arithmetic, 385, 631
- floating point number, 625
- forbidden point, 32
- Forrest, A.R., 3, 6, 7, 10
- forward difference, 83, 94, 95, 751
- frame
 - Frenet, 744
 - rational, 744

- rotation-minimizing, 744
- frame line, 47
- frame point, 45
- Frechet metric, 755
- free-form
 - blend, 662
 - curve, 667
 - fitting, 659
 - object, 656
 - surface, 665
- Frenet frame, 39, 200, 210, 411
- frontier condition, 478
- functional, 705
 - Bézier patch, 92
 - Bézier triangle, 99
 - data dependent, 660
 - smoothing, 659, 660
- functional curve, 78
- functional decomposition, 657, 658, 662
- fundamental
 - splines, 710
- fundamental domain, 459
- fundamental Lagrange splines, 713
- fundamental points, 33
- fundamental quantities, Gaussian, 40
- fundamental simplex, 45

- G code, 418
- Galerkin, 283
- Gauss elimination, 370
- Gaussian curvature, 14, 42
- Gaussian fundamental quantities, 40
- Gaussian sphere, 664, 667
- GCD, 366, 378
- Geise, G., 13
- general supercyclide, 585
- generality of topology, 309
- generalized cross validation, 161
- generalized cyclide, 575
- generalized distance, 441
- generalized stereographic projection, 782
- generating point, 429
- generator, 366, 378
- generatrix, 136
- generatrix of a quadric, 28

- generic 2-prong, 457
- generically, 706
- genus, 66, 374
- geometric
 - construction of box splines, 257
 - fitting, 665
- geometric constraint problem
 - parametric, 522
 - variational, 522
- geometric constraint solving, 522
- geometric continuity, 384
- geometric degeneracy, 319
- geometric graph, 453
- geometric Hermite interpolation, 417
- geometric tolerancing, 55
- geometrical optics, 420
- geometry
 - elliptic, 66, 729
 - Galilei sphere, 60
 - isotropic, 67
 - Laguerre, 50
 - Lie, 50
 - line, 43, 60
 - Minkowski, 52
 - Möbius, 50
 - non-Euclidean, 43
 - pseudo-Euclidean, 52
 - sphere, 43, 50
- Gordon, W., 3, 8
- Gouraud shading, 750
- Gouraud, H., 15
- Gröbner
 - basis, 366, 370, 371, 377, 382
- Gröbner bases, 626
- gradient value, 708
- Gram-Schmidt's orthogonalization, 30
- graph of a function, 155
- graph surface, 66
- graph theory, 706
- graphics
 - computer, 683
- greatest common divisor, 366
- Gregory, J., 9, 11, 12
- Greville sites, 148
- grid

- structured, 685
- unstructured, 686
- groupings
 - groupings
 - k -dimensional, 485
 - properties, 486
- guiding polygon, 667, 668
- Hagen, H., 1, 14
- half-box spline
 - Bézier representation, 277
 - geometric definition, 273
 - linear dependency, 276
 - partition of unity, 275
 - properties, 273
 - derivatives, 273, 274
 - differentiability, 274
 - inductive definition, 272
 - linear precision, 276
 - subdivision, 277
- half-box spline surface, 275
 - generalized, 278
 - derivatives, 275
 - minimal, 279
- hardware-based rendering, 773
- harmonic parametrization, 660
- harmonic position, 34
- hat-function, 283, 297, 298
- Hatvany, J., 3
- helical motion, 61
- helix, 411
- Hermite
 - conditions, 702
 - interpolation, 702, 705, 708
 - interpolation operator, 710
- Hermite interpolation, 173, 412
- Hermite polynomial, 79, 87
- heuristic tree search, 333
- hierarchical basis, 288
- hierarchical patch, 657
- hodograph, 84, 406, 415, 421
- homogeneous coordinates, 32, 35, 408, 587, 777
 - continuity, 213
- homological approach, 706
- homotopy method, 416
- Hopf mapping, 63
- Horner scheme, 129
- Hosaka, M., 8
- Hoschek, J., 1, 10, 13, 15
- hyperbola, 118
- hyperbolic paraboloid, 90
- hyperbolic plane, 65
- hyperbolic projection, 783
- hyperplane, 25, 35
 - oriented, 50
- hypersphere
 - oriented, 50
- hypersurface, 366
- ideal, 378
 - basis, 373
- ideal plane, 32
- IGES, 97, 138
- ignore area, 658
- image plane, 755
- immersed C^k surface, 212
- implementation, 716
- implicit
 - surface, 665
- implicit degree, 380
- implicit form, 760
- implicitization, 364, 370
- improper parameterization, 384, 421
- incompressible flow, 299
- indicatrix, 423
- infinite points, 32
- infinite plane, 32
- inflection point, 78, 86
- informational completeness, 473
- inhomogeneizing, the procedure of, 34
- integration, 302
- interference, *see* quadric
- interior
 - edges, 704
 - vertices, 704
- interlacing condition, 711
- interoperability, 508
- interpolant, 437
- interpolated height, 437

- interpolating refinement, 312
- interpolating subdivision, 312
- interpolation, 741
 - C^2 bicubic spline, 186
 - C^2 cubic spline, 175
 - Aitken's algorithm, 167
 - bicubic Hermite patches, 185
 - bivariate splines, 711
 - cardinal form, 174
 - cubic Hermite, 173
 - end conditions, 177
 - end point, 119
 - of keyframes, 723
 - point data, 166
 - tensor product, 181
 - with motions, 741
- interpolation algorithm, 740
- interpolation method, 707
- interpolation schemes, 701
- interpolation sets, 710
- interrogation, 327
 - analysis, 327
 - cross-sectioning, 327
 - display, 327, 333
- intersection, 364, 376, 789
 - algorithm, 377
 - curve to surface, 632
 - curve-plane, 329
 - lattice method, 639
 - line and curve, 81
 - marching method, 640
 - plane-surface, 335
 - subdivision method, 639
 - surface to surface, 634
 - surface-surface, 336
 - tracing method, 636
- intersection curve, 789
 - irreducible, 789
 - nonsingular, 789
 - QSIC, 789
 - reducible, 789
 - singular, 789
- interval arithmetic, 631
- interval number, 625
- Interval Projected Polyhedron (IPP) algorithm, 631
- invariance
 - coordinate, 740
 - parameter, 743
- inverse of the B-spline basis, 151
- inversion, 53, 364, 370
- involute, 420
- irreducible curve, 374
- irregular vertices, 270
- iso-distance, 755
- isolines, 40
- isometric mapping, 769
- isometry, 765
- isoparameter lines, 576
- isoparametric curve, 90, 93, 96
- isoparametric distance, 755
- isophote, 37
- iterative closest point, 654
- iterative refinement, 309
- iterative solution, 303
- Jacobian matrix, 416
- k-cycle, as a boundary, 479
- Kantorovich, 284
- Kantorovich conditions, 416
- keyframe interpolation, 723
- Kimura, F., 8, 9
- kinematics, 723
 - spatial, 60
 - spherical, 66
- Kjellander, J., 13
- Klein quadric, 60
- knee joint, 743
- knot, 119
 - choice, 659
 - insertion, 661
- knot averages, 148
- knot insertion, 156, 310, 328
- knot multiplicity, 148, 150
- knot multiplicity *vs* smoothness, 147
- knot selection rule, 715
- knot sequence, 141, 187
- knot vector, 119, 131

- L_∞ distance, 444
- Lagrange
 - data, 710
 - interpolation, 702, 710
 - interpolation points, 713
 - interpolation sets, 710
- Lagrange interpolation points, 712
- Lagrange polynomials, 169
- Lagrangian multiplier, 665
- Laguerre, 442
 - distance, 442
 - Voronoi diagram, 443
- Laguerre geometry, 50, 420
 - Blaschke model, 52
 - cyclographic model, 52
 - isotropic model, 53
- Laguerre transformation, 50
- Laplace equation, 298
- laser scanner, 654
- Laurent, P.-J., 1
- Lawson, C., 14
- Lax-Milgram Lemma, 298
- leading term, 365
- least squares approximation, 170
- least-squares
 - separable, 666
 - sequential, 665, 666
- least-squares approximation, 161
- left-continuous, 145
- length of a vector, 30
- Levin's method, 790, 791
- Levin, D., 12
- lexicographic ordering, 365, 371
- lexicographically smaller, 433
- Lie cycle, 593
- Lie geometry, 50, 593
- Liming, R., 2
- line complex
 - linear, 61
- line coordinates, 419
- line geometry, 43, 60
- line integral convolution, 691
- line segments, 711
- line-art, 765
- linear
 - functionals, 706
 - spline, 710
- linear complex, 61
- linear dependence, 23
- linear independence, 149
- linear interpolation, 80, 90, 167
- linear precision, 78, 83, 168
 - box spline representations, 262
 - half-box spline representations, 276
 - Bézier triangle, 99
 - bivariate Bernstein polynomial, 102
- linear precision property, 629
- linear space, 23
- Linkage Curve Theorem, 217
- Lipschitz condition, 416
- local
 - Lagrange interpolation, 713
- local frame, 38
- local Lagrange
 - interpolation, 702
- local linear independence, 149
- local support, 713
- lofting, 7
- logarithmic divergence, 319
- Loop subdivision, 314
- Loop, C., 12
- lower and upper bounds, 704
- lower bound, 704
- L_p distance, 444
- Lüroth, 384
- Lyché, T., 1, 10
- MacLaren, D., 7
- macro element, 708
- macro element methods, 702
- manifold, 212
 - definition, 479
 - solid, 480
 - Veronese, 45
- Manning, J, 13
- map
 - affine, 78, 92, 99
- Maple, 367
- mapping
 - affine, 738

- barycentric, 660
- cyclographic, 51
- equiform, 739
- kinematical, 724, 727, 728, 740
- marching cubes algorithm, 655
- Marsden's Identity, 147, 149
- mask, 310
 - Bézier points of box splines, 267, 269
- mass properties, 498
- master model, 533
- Mathematica, 367
- matrix
 - orthogonal, 725
 - special orthogonal, 725
 - Vandermonde, 167
- maximum ball, 672, 673
- mean curvature, 14, 42
- medial axis, 55, 435, 451
 - transform, 55
- medial axis transform, 422, 451, 672
- medial surface, 436
- membrane, 297
- merging
 - point clouds, 654
- meridian, 136
- mesh
 - tetrahedral, 290
 - triangular, 290
- mesh generation, 327
- meshing, 501, 509
- metric
 - isotropic, 67
- Meusnier's sphere, 40
- midpoint of a quadric, 27
- minimal
 - support, 710
- minimal control net
 - box spline surface, 270
 - half-box spline surface, 279
- minimal determining sets, 706
- minimum property, 179
- Minkowski isoperimetric-hodograph curve, 423
- Minkowski metric, 421
- Minkowski plane, 422
- Minkowski Pythagorean-hodograph curve, 422
- mixed partial, 64
- Möbius geometry, 50
- Möbius hyperspheres, 53
- model
 - B-rep, 673
 - incomplete, 674
 - prismatic, 674
 - reconstruction, 673
- modeling, 716
 - assembly, 499
 - continuity, 507
 - deformations, 510
 - heterogeneous materials, 508
 - lumped-parameter systems, 500
 - mechanisms, 499
 - motion, 499
 - NC machining, 500
 - physical fields, 508
 - solid, 473
 - tolerances, 507
 - unit processes, 501
- Moebius transformation, 122, 133
- monomial, 365
- monomial basis function, 76
- monomial form, 704
- monomials, 76, 85, 86
 - Vandermonde matrix, 167
- Morgan-Scott example, 705
- motion
 - camera, 723
 - Darboux, 737, 740
 - elliptic, 737
 - helical, 61
 - of a rigid body, 724
 - of the human knee joint, 743
 - rational, 723
 - planar, 738
 - spatial, 735
 - spherical, 732
 - robot, 723, 743
 - rotational, 726
 - spatial, 724
 - spherical, 726, 729

- motion fairing, 743
- motion fitting, 743
- motion planning, 501
- moving
 - conic, 371
 - curve, 371
 - line, 371
- moving surface method, 382
- MRI, 189
- multi-point approximation scheme, 126
- multiaffine map, 704
- multigrid, 303
- multiple control points, 271
- multiplicatively weighted Voronoi diagram, 442
- multiplicity of a spline zero, 159
- multiresolution analysis, 709
- multiresolution index, 340
- multivariate
 - B-splines, 702, 714
 - polynomials, 704
 - splines, 701
- multivariate functions, 773

- n*-sided patches, 309
- natural configuration, 318
 - second order, 318
- natural end condition, 177
- natural quadric, 651
- NC machining, 327, 745, 765
- neighborhood
 - constant, 478, 486
 - of a cell, 486
 - of a point, 476
- neighbourhood graph, 655
- nested polygon
 - triangulations, 707, 713
- Neumann problem, 299
- Newton method, 377
- Newton-Raphson method, 416
- Nielson, G., 11, 13, 179
- nodal
 - basis, 705
- non-degenerate
 - triangulations, 707
- non-uniform knots, 416
- normal curvature, 41
- normal curve
 - rational, 44
- normal deviation, 661
- normal hull, 331
- normal of a surface, 40
- normal vector, 95, 102
- normalization of a B-spline, 144
- Not-a-knot end condition, 178
- numerical algorithm, 707
- numerical stability, 86, 407
 - extrapolation, 78
- numerically controlled machining, 327, 332
- numerics, 683
- NURBS, 732
- NURBS curve, 46, 119
 - spherical, 63
- NURBS surface, 46, 131

- object
 - auxiliary, 669
 - benchmark, 663
 - conventional, 651, 656, 663, 665
 - free-form, 651, 656
- obstacle-avoidance Voronoi diagram, 445
- occlusion, 654
- octant, 195
- octrees, 486
- odd degree triangulation, 712
- offset, 788
 - rational, 58
- offset curve, 407, 408, 418, 420, 436
- offset surface, 332, 436, 577, 788
 - circular, 60
- offsetting, 788
- optimal
 - approximation order, 706, 710
- optimal approximation order, 714
- optimal valid coverage, 754
- orbifold, 214
- order, 420
- order *vs* degree, 141
- orthogonal projections, 31
- orthogonal vectors, 30

- orthogonalization, Gram-Schmidt's, 30
- osculating circle, 39
- osculating curves and surfaces, 36
- osculating plane, 38, 123
- osculating simplex, 45
- osculating space, 44
- osculatory interpolation, 125
- outlier, 654
- Padé approximant
 - parametric, 126
- Padé approximation, 126
- parabola, 118, 421
- parabolic supercycloid, 585
- parallel manipulator, 62
- parallelism, 25
- parameter
 - correction, 659, 660
- parameter correction, 135
- parameter estimation, 184
- parameter transformation, 79
- parameterization, 659, 742
 - harmonic, 660
 - initial, 659
 - projective, 116
- parametric
 - constraints, 498
 - constructions, 503
 - design, 504
 - families, 505, 507
- parametric curve, 76
 - domain, 76
- parametric degree, 380
- parametric speed, 407
- parametric surface, 88, 364
 - domain, 88
- parametric texture mapping, 757
- parametrics, 519
 - parametric solid, 519
 - parametric solid modeling, 519
- parameterization, 364
 - centripetal, 178
 - chord length, 178
- parameterization, equidistant, 178
- parameterization, uniform, 178
- Parry, S., 9, 188
- partial
 - derivative, 703
- partial derivative, 94
- partition of unity, 82, 142, 148
 - bivariate Bernstein polynomial, 101
- Pascal's theorem, 29
- patch, 89
 - hierarchical, 657
- patch complex
 - geometric continuity, 209
- pathline, 689
- Peixoto, 693
- pencil-of-line, 375, 383
- performance issues, 338
- persistent naming, 485, 504, 507, 533
- PH quintic, 407
- PH quintic spline, 414
- PH space curve, 410
- Phong lighting, 771
- Phong shading, 750
- Phong, B., 15
- Picard-Lindelöf, 692
- piecewise
 - polynomial, 703
- piecewise conic interpolant, 125
- piecewise polynomial, 141
- pipe surface, 136
 - rational parameterization, 137
- pipeline
 - visualization, 684
- Plücker coordinates, 668
- planar cubic curve, 791
- planarity filtering, 664
- plane, 664
 - control, 50
 - hyperbolic, 65
- plane-surface intersection, 764
- Plücker coordinates, 60
- Plücker identity, 60
- PMC (Point Membership Classification), 492
- point
 - Bézier, 45
 - classification, 481, 489, 492

- Farin, 732
- frame, 45
 - generation, 492
- point cloud, 674
- point light source, 772
- Poisson equation, 283, 296, 301
- polar form, 152, 704, 715
- polarity, 35
- pole and polar plane, 29
- polyhedral
 - splines, 714
- polyhedron, 712
- polyhedron, topological, 479
- polynomial
 - Bernstein, 77
 - monomial, 76
 - surfaces, 704
 - trivariate, 217
- polynomial ideal, 366
- polynomial interpolation
 - oscillate, 170
- polynomial solver
 - algebraic technique, 626
 - homotopy method, 627
 - hybrid technique, 626
 - subdivision method, 627
- polynomials, 707
- positions
 - control, 738
 - interpolation of, 741
- positivity of a B-spline, 143
- Postscript, 772
- Powell, M., 14
- Powell-Sabin
 - splits, 709
 - triangulation, 709
- Powell-Sabin element, 709
- power
 - basis, 363
- power basis, 76
- power diagram, 443
- power distance, 442
- Pratt, M., 14
- Prautzsch, H., 12
- predictor-corrector method, 416
- primary rays, 761
- primary surface, 653, 658, 659, 662
- primitives, candidate, 491
- principal axis direction, 32
- principal curvature, 41
- principal directions, 41
- principle of degree reduction, 711
- priority
 - of constraint, 669
- priority queue, 333
- prismatic reconstruction, 674
- projecting line, 783
- projection cone, 791
- projections, orthogonal, 31
- projective A-frame, affine representation, 29
- projective canal surface, 586
- projective combinations, 34
- projective coordinates, 33
- projective extension, 32
- projective frame, 33
- projective invariance, 113
- projective line, 34
- projective map, 33
- projective space, 32
- projective translation surface, 584
- prong, 456
- Pythagorean theorem, 405
- Pythagorean triple, 406
- Pythagorean-hodograph curve, 406
- Pythagorean-hodograph curve, 58
- Pythagorean-normal (PN) surface, 59
- QSIC, *see* quadric
- Qu, R., 11
- quadrangulations, 708, 713
- quadric, 137, 655, 665, 777
 - affine classification of, 779
 - classification, 777
 - ellipsoids, 792
 - separation of, 792
 - Euclidean classification of, 778
 - interference analysis, 792
 - irreducible, 778
 - natural, 651

- nondegenerate, 778, 779
- parameterization, 780
- projective classification of, 779
- properly degenerate, 778, 779
- QSIC, 789
- singular, 778
 - triangular patch on, 784
- quadric in affine space, 27
- quadric in projective space, 34
- quadric surface, 137, 383
 - intersection curve of, 789
- quadric, equation of a, 35
- quadric, parametrization of a, 35
- quadrilateral, 708
- quantized hull, 330
- quasi
 - crosscut
 - partitions, 707
- quasi-interpolation, 712
- quaternion, 411, 727
 - dual, 723, 740
 - multiplication of, 728
 - unit, 723, 728
- quaternion sphere, 729, 743

- R-function, 284, 293, 481
- radial line, 103
- radiosity, 750
- Ramshaw, L., 6, 10, 152
- range data, 652, 653
- ratio, 26, 80
- ratio of parallel distances, 25
- rational arithmetic, 632
- rational B-spline, 739
- rational B-spline curve
 - degree elevation, 120
 - derivative, 120
 - knot insertion, 120
- rational B-spline patch, 131
 - derivative, 131
 - evaluation, 133
- rational B-spline surface, 131
- rational Bézier, 739
- rational Bézier curve
 - degree elevation, 116
 - derivative, 114
 - reparameterization, 116
- rational Bézier patch, 127
 - triangular, 137
- rational Bézier surface, 127
 - derivative, 129
 - reparameterization, 131
- rational curve, 39, 363, 782, 790
 - approximation, 123, 126
 - geometric continuity, 122
 - interpolation, 123, 124
- rational curves and surfaces, 34
- rational linear transformation, 122
- rational number, 625
- rational parametric surface, 380
- rational patch
 - continuity, 133
- rational PH curve, 419
- rational quartic curve, 790
- rational surface
 - approximation, 134
 - interpolation, 134
- rational triangular Bézier patch, 137
- ray casting, 327, 333
- ray casting, tracing, 494
- ray-surface intersection, 759
- ray-tracing, 384, 750
- real analytic curve, 454
- real-time interpolator, 418
- rectangular Bézier patch, 88
- rectification, 405
- rectilinear
 - partitions, 705
- recurrence for B-splines, 142
- recurrence for Bernstein basis, 146
- recurrence for cardinal B-splines, 146
- recursion relation, 82
 - bivariate Bernstein polynomial, 101
- recursive refinement, 309
- recursive subdivision, 309
- reference point, 418
- reflection direction, 771
- reflection line, 37
- region
 - adjacency, 673

- growing, 663
 - multiple, 664
 - simple, 664
- registration, 654
- regular curve, 36
- regular sets, regularization, 476
- regular subdivision, 315
- regular vertices, 270
- renormalization, 44, 583
- reparametrization
 - global, 214, 215
 - local, 215
 - regional, 215
- repeated subdivision
 - Bézier triangle, 103
- representation
 - boundary, 488
 - BSP trees, 486
 - cell complexes, 486
 - constructive, 481, 483
 - conversion, 494, 497
 - CSG, 482
 - efficient, 671
 - enumerative, 481, 485
 - faithful, 665
 - grouping, 485
 - implicit, 480
 - implicit, functions, 481
 - octree, 486
 - of cells, 485
 - resolvable, 676
 - scheme, 474
 - STL, 502
 - sweep, 484
 - unified, 490
 - winged-edge, 487, 489
 - with R-functions, 481
- resolvable representation, 676
- resultant, 367, 370, 377, 382, 626
 - Bezout, 368
 - Dixon, 369
 - Sylvester, 367
- reverse engineering, 62, 502, 651
- Riesenfeld, R., 1, 10, 11
- right-continuous, 141
- ring cyclide, 580
- Ritz, W., 283
- robot controller, 743
- robot motion, 723, 743
- robustness, 423, 506
- rolling-ball blend, 672
- rotary table, 654
- rotation index, 413
- rounded interval arithmetic, 631
- rounded interval number, 625
- ruled surface, 62, 762
- ruled tracing, 759
- ruled-surface-surface intersection, 765
- Runge phenomenon, 170
- Rvachev, V.L., 284, 293
- Sabin, M., 3, 8, 11, 15
- saddle point
 - higher-order, 216, 220
- sampling interval, 418
- Sbounds, 339
- scan-conversion, 749
- scan-line algorithm, 384
- scanner, 654
- scattered data
 - fitting, 701
- scattered points, 712
- Schmidt's orthogonalization, 39
- Schoenberg, 145
- Schoenberg's operator, 158
- Schoenberg-Whitney conditions, 160
- Schoenberg-Whitney Theorem, 160
- Schoenberg-Whitney type conditions, 711
- Schumaker, L., 1, 11
- scientific computing, 745
- scientist, 683
- second-order
 - derivatives, 708
- Sederberg, T., 9, 188
- segmentation, 653, 656, 657, 663
 - direct, 663
 - two-dimensional, 664, 667
- Segre characteristic, 790
- self-conjugate, 29
- self-occluding surface, 764

- semantic feature, 533
- semi-parabolic supercyclide, 585
- semi-singular
 - vertices, 713
- sequential least-squares, 665, 666
- sets
 - closed regular, 476
 - semi-analytic, 476
- shadow detection, 761
- shape classification, 45
- shape equations, 172, 184
- shape handle, 114
- shape-preserving, 159
- sharp corner, 456
- Sharrock, T., 8
- shift operator, 81
- shoulder point, 118
- shoulder tangent, 118
- Sibson, R., 13
- silhouette, 337
- silhouette area, 772
- silhouette curves, 757
- silhouette edges, 750, 755
- simplex
 - splines, 714
- singular
 - vertex, 709
 - vertices, 705
- singular point, 577, 637
- singular point of a quadric, 28
- singularity, 316, 374
- skeleton, 452
- sketching, 522
 - sketched features, 530
- skin thickness, 332
- slerping, 723, 729
- slopes, 704
- smallest angle, 710
- SMC (Set Membership Classification), 495
- smooth surfaces, visual, 37
- smoothing, 170
 - functional, 660
- smoothing spline, 14, 160, 161
- smoothness conditions, 147, 149, 706
- Sobolev space, 284
- solidity
 - combinatorial model, cell complex, 477
 - generalized model, 480
 - manifold, 480
 - point set model, closed regular sets, 476
 - postulates, 475
 - semi-analytic sets, 476
- space
 - elliptic, 729
 - of geometric objects, 63
- space cubic curve, 790
- space-time, 421
- spatially addressable, 485, 504
- specular highlights, 757
- specular lighting, 770
- specular reflection, 765
- sphere, 131, 136, 664
 - fitting, 666
- sphere geometry, 43, 50
 - Galilei, 60
- spherical point, 41
- spindle cyclide, 580
- spline, 7, 286
 - approximation, 287
 - cubic, 742
 - cubic Hermite, 743
 - hierarchical, 288
 - stability, 296
 - uniform, 285
 - web-, 292
- spline curve, 137, 155, 411
- spline interpolation, 159
- spline of order k with knot sequence \mathbf{t} , 144
- spline space, 710
 - dimension, 206
- spline subdivision, 310
- splines on surfaces, 248
- splitting procedure, 708
- sqrt3 subdivision, 321
- stability, 296
- stable, local bases, 710
- standard representation, 116
- standards, data exchange, 505

- star, 705
- star-supported
 - splines, 706
- Steiner surface, 781
- stencil, 310
- STEP, 138
- stereographic projection, 743, 780–782, 791
 - generalized, 63, 734, 743, 782
- stitching, 674
- stratification
 - computing, 491, 502
 - definition, 478
 - intersection, 491
 - sign-invariant, 490
 - Whitney regular, 490
- streakline, 690
- stream
 - surface, 690
- structural analysis, 327
- structured
 - grid, 685
- subdivision, 81, 303, 377, 714
 - 4-point, 312
 - analysis, 315
 - Bézier patch, 96
 - Bézier triangle, 103
 - bicubic, 314
 - biquadratic, 313
 - butterfly, 314
 - Catmull-Clark, 314
 - Chaikin, 309
 - cubic, 310
 - display, 328
 - Doo-Sabin, 313
 - for box splines, 263, 264
 - convergence, 265
 - for half-box splines, 277
 - interpolating, 312
 - Loop, 314
 - of B-splines, 310
 - over semi-regular lattices, 321
 - precision set, 320
 - quadratic, 309
 - regular, 315
 - repeated, 81
 - square root of 3, 321
- subdivision interrogation, 327
- subdivision property, 730, 739
- super
 - spline, 709
 - spline spaces, 706
 - splines, 701
- supercyclide, 576, 583
 - central projection, 585
 - classification, 585
 - first axis, 584
 - first pencil, 584
 - fourth order, 585
 - general, 585
 - invariant, 586
 - parabolic, 585
 - second axis, 584
 - second pencil, 584
 - semi-parabolic, 585
 - third order, 585
- supersmoothness, 710
- support analysis, 315
- support of a B-spline, 143
- surface, 701
 - analytic, 665
 - base, 659
 - bilinear, 89
 - blending, 63
 - canal, 56
 - cone spline-, 68
 - Coons, 659
 - developable, 49
 - domain, 88
 - evaluation, 93
 - feature, 653, 662
 - fitting, 653, 665, 714, 716
 - constrained, 653, 668
 - free-form, 220, 665
 - implicit, 665
 - interrogation, 96
 - NURBS, 46
 - of revolution, 653, 664–667
 - offset, 58
 - partial derivative, 93
 - patch, 89

- primary, 653, 659, 662
- Pythagorean-normal (PN), 59
- quadric, 665
- rational, 127
- rectangular Bézier patch, 88
- ruled, 62
- stream, 690
- sweeping, 723, 744
- tensor product, 91, 127
- surface coverage, 750
- surface normal, 757
- surface of revolution, 136, 759
- surface rendering, 749
- surface-surface intersection, 762
- Sutherland, I., 4
- sweep, 653, 664, 666, 667, 672
 - applications, 499, 500
 - definition, 484
 - point classification, 484
- sweep surfaces, 759
- sweeping, 723, 744
- swept surface, 411
- Sylvester's
 - dialytic method, 367
 - resultant, 367, 370
- symmetry, 653
 - Bézier curve, 77
 - Bézier patch, 92
 - Bézier triangle, 99
 - Bernstein polynomials, 82
 - bivariate Bernstein polynomial, 101
 - detection, 675
- systems
 - classical, 502
 - dual-representation, 503
 - parametric, 503
- TABCYL, 8
- tangent hull, 331
- tangent of a curve, 38
- tangent of a quadric, 28
- tangent plane, 93, 95, 102
- tangent plane of a quadric, 28
- tangent vector, 83, 93, 102, 730
- target point, 437
- Taylor series, 418
- template, 310
- tensor product, 312
 - approximation, 182
- tensor product interpolation, 181
- tensor product surface, 91, 92, 381
- term order, 365
- terminal point, 456
- texture, 691
- theorem
 - Bezout's, 381
- theorem of Sabin, 589
- thick curve, 668
- thin plate energy, 660
- thin plate spline, 12
- timeline, 690
- tolerance region, 55
- tolerances, 507
- topological generality, 309
- topology
 - of B-rep model, 673
- torsion, 39, 86
- torus, 131, 136, 578, 651, 664
 - fitting, 666, 667
- total curvature, 14
- total degree, 92
 - Bézier triangle, 99
- total parametric degree, 381
- totally positive, 160
- trajectories
 - of planes, 740
 - of points, 739
- transfinite interpolation, 179
- translational surface, 184
- triangular
 - B-splines, 714, 716
- triangular Bézier patch, 97
- triangular de Casteljau algorithm, 100
- triangular diagram, 80
- triangular patch on, 784
- triangulation, 184, 476, 478, 652, 654, 655, 672
 - 2D, 660
 - Delaunay, 655
- tridiagonal system, 415

- trimmed surfaces, 757, 761
- trivariate B-splines, 189
- trivariate functions, 773
- truncated power, 148
- truncated power functions, 707
- Tschirnhausen cubic, 407
- turning point, 637
- twist vector, 94
- two-sided offset, 421

- umbilic, 577
- umbilical point, 41
- unequal interval subdivision, 322
- uniform type
 - triangulations, 707
- unimodular, 261
- UNISURF, 6
- unit point, 33
- unit quaternion sphere, 729, 743
- unit vector, 703
- univariate
 - spline, 701
 - spline spaces, 711
- unsteady
 - vector field, 689
- unstructured
 - grid, 686
- unsweep, 484, 499
- upper bound, 705
- usefulness of B-spline basis, 151

- valid coverage, 751
- Vandermonde matrix, 167, 182
- variant
 - design, 521
 - modeling, 521
- variation diminishing, 78, 158
- variation diminishing property, 112, 120, 168, 329
- variational
 - class, 520
 - family, 521
- variational approach, 659
- variety, 366, 377
- vector field
 - unsteady, 689

- Vernet, D., 6
- Veronese manifolds, 45
- Veronese surface, 785
- Versprille, K., 10
- vertex
 - blend, 672
- viewing direction, 755, 764
- viewing space, 754
- virtual reality, 723
- visualization, 683
 - pipeline, 684
- volume deformation, 9, 188
- volumes
 - Bézier patch, 96
 - Bézier triangle, 104
- volumetric texture mapping, 757
- Voronoi, 429
 - diagram, 429
 - edge, 430
 - point, 430
 - region, 429
- Voronoi diagram, 13, 451
 - curved, 655

- walking the dog, 755
- Watson, D., 15
- web-spline, 292
- Weierstrass approximation theorem, 85
- weight, 46, 112, 119, 128, 131, 441
- weight function, 120, 291, 293, 295
 - singular, 125
- weight point, 113, 128
- Whitney regular stratification, 490, 491
- Whittaker's Cardinal Series, 145
- Whitted, T., 15
- Wilson-Fowler spline, 8
- winged-edge, data structure, 487, 489

- Z-buffer, 327, 333, 749
- Z-depth map, 772
- zero of a spline, 159
- zero-set, 217, 759, 760
- Zwart-Powell element, 260