SYMBOLIC COMPUTATION

Computer Graphics – Systems and Applications

Managing Editor: J. L. Encarnação

Editors: K. Bø J. D. Foley R. A. Guedj
P. J. W. ten Hagen F. R. A. Hopgood M. Hosaka
M. Lucas A. G. Requicha

Springer Series
SYMBOLIC COMPUTATION
Computer Graphics – Systems and Applications

J. L. Encarnação, R. Schuster, E. Vöge (eds.):
Product Data Interfaces in CAD/CAM Applications.
Design, Implementation and Experiences.
IX, 270 pages, 147 figs., 1986

U. Rembold, R. Dillmann (eds.):
Computer-Aided Design and Manufacturing.
Methods and Tools. Second, revised and enlarged edition.
XIV, 458 pages, 304 figs., 1986

G. Enderle, K. Kansy, G. Pfaff:
Computer Graphics Programming. GKS – The Graphics
Standard. Second, revised and enlarged edition.
XXIII, 651 pages, 100 figs., 1987

Y. Shirai:
Three-Dimensional Computer Vision.
XII, 297 pages, 313 figs., 1987

D. B. Arnold, P. R. Bono:
CGM and CGI. Metafile and Interface Standards
for Computer Graphics.
XXIII, 279 pages, 103 figs., 1988

J. L. Encarnação, P. C. Lockemann (eds.):
Engineering Databases. XII, 229 pages, 152 figs., 1990

P. Wisskirchen:
Object-Oriented Graphics. From GKS and PHIGS
to Object-Oriented Systems.
XIII, 236 pages, 83 figs., 1990

J. L. Encarnação, R. Lindner, E. G. Schlechtendahl:
Computer Aided Design. Fundamentals and System
Architectures. Second, revised and extended edition.
XII, 432 pages, 240 figs., 1990

José L. Encarnação Rolf Lindner
Ernst G. Schlechtendahl

Computer Aided Design

Fundamentals and System Architectures

Second, Revised and Extended Edition

With 240 Figures, Including 34 in Color

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona

Prof. Dr. José L. Encarnação
Dr. Rolf Lindner
TH Darmstadt
Institut für Informationsverwaltung
und Interaktive Systeme, FB 20
Wilhelminenstraße 7
D-6100 Darmstadt, FRG

Dr. Ernst G. Schlechtendahl
Kernforschungszentrum Karlsruhe
Postfach 3640
D-7500 Karlsruhe, FRG

CR Classification (1987): I.3, J.6, B.4.4, A.1

ISBN-13: 978-3-642-84056-2 e-ISBN-13: 978-3-642-84054-8
DOI: 10.1007/978-3-642-84054-8

Library of Congress Cataloging-in-Publication Data
Encarnação, José Luis.
Computer aided design : fundamentals and system architectures / J. Encarnação, R. Lindner, E. S. Schlechtendahl.–2nd, rev. and extended ed. p. cm.–
(Symbolic computation. Computer graphics–systems and applications)
Includes bibliographical references.

1. Engineering design–Data processing. 2. Computer-aided design. 3. Computer architecture. I. Linder, R. (Rolf) II. Schlechtendahl, E. G. (Ernst G.), 1938– . III. Title. IV. Series.
TA174.E47 1990 620'.0042'0285–dc20 90-9721 CIP

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in other ways, and storage in data banks. Duplication of this publication or parts thereof is only permitted under the provisions of the German Copyright Law of September 9, 1965, in its current version, and a copyright fee must always be paid. Violations fall under the prosecution act of the German Copyright Law.

The use of registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

© Springer-Verlag Berlin Heidelberg 1983, 1990

Softcover reprint of the hardcover 2nd edition 1990

Typesetting: K & V Fotosatz, Beerfelden
2145/3140-543210 – Printed on acid-free paper

Preface

This second edition is a thorough revision and extension of the first edition, published in 1983. Many minor corrections have been made to eliminate errors or to adapt to new circumstances. Several subchapters have been extensively revised to preserve their topicality. Some major additions have been made to eliminate deficiencies in the first edition and to reflect new developments in science and technology. Several topics have been moved within the book so that they are nearer to corresponding topics. And of course the bibliography has been updated.

Chapter 1, *Introduction*, has been adapted to the new contents of this edition.

Chapter 2, *History and Basic Components of CAD*, is almost new. The subchapter *Graphics Standards* is a revised and substantially extended version of the former subchapter on the Graphical Kernel System and now includes sections on the computer graphics reference model, GKS, CGI, CGM, GKS-3D, PHIGS, and PHI-GKS. The subchapter *The Graphical Dialogue System* has been substantially revised and extended to discuss window managers, user interface toolkits and user interface management systems, and now includes a detailed example (THESEUS). The former subchapter on data bases for CAD has been replaced by a new subchapter *Application Interfaces to Engineering Databases* including much more detailed information. The former subchapter on economical aspects of CAD has been replaced by an updated and more detailed subchapter *Evaluating and Choosing CAD Systems.* These three subchapters all contain revised parts of the fifth chapter of the first edition.

Chapter 3, *The Process Aspect of CAD*, and Chapter 4, *The Architecture of CAD Systems*, have remained almost unchanged.

Chapter 5, *Implementation Methodology*, is mostly new. It now focuses on the hardware, the system architecture and the system/application interface of graphics. (Discussions of CAD system development, data bases, integrated systems, methods bases, and AI, in the fifth chapter of the first edition now appear in Chapter 2.) The former subchapter on computer graphics has been completely rewritten and now covers graphics peripherals, graphics workstations and network aspects in much more detail. The description of GKS as the system/application interface has been revised.

Chapter 6, *Engineering Methods of CAD*, now includes the new subchapter *Mathematical Description of Curves and Surfaces.*

Chapter 7, *CAD Data Transfer*, is completely new. (The former seventh chapter is now Chapter 8.) Data exchange standards and neutral formats are described, including IGES, SET, VDA-FS, XBF, ESP, CAD*I, PDES, and STEP.

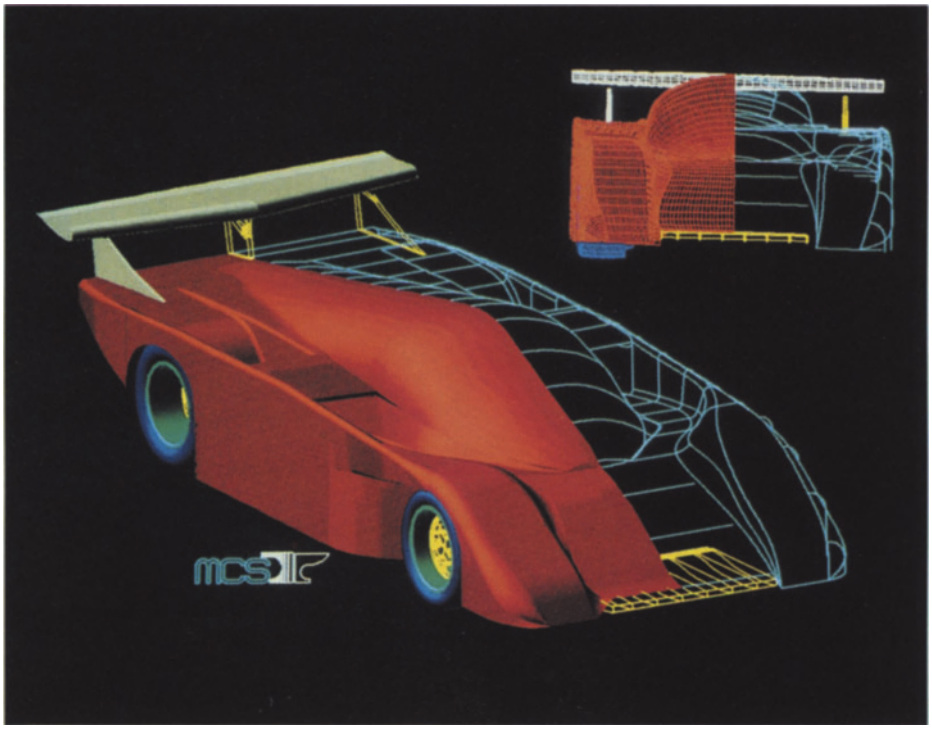
Chapter 8, *CAD Application Examples*, (the former seventh chapter) has been substantially revised and includes several new examples. The eighth chapter of the first edition on trends has not been retained. Though many of the global trends described in the first edition are still evident, the authors have the impression that trends are an appropriate subject for special periodicals but are out of place in a book.

Darmstadt, Karlsruhe
August 1990

J. L. Encarnação
R. Lindner
E. G. Schlechtendahl

Contents

1 Introduction



Combination of two different presentation modes for a coachwork
(courtesy of Tektronix, Beaverton, USA)

1.1 Purpose of this Book

The intention of this book is to describe principles, methods and tools that are common to computer applications for design tasks, independent of a particular product. It does not present cookbook recipes on how to select a commercially available CAD system for this purpose. When we consider CAD as a discipline lying somewhere between engineering and computer science, the tendency towards generalization inevitably leads us to emphasize the computer aspects. But the book is primarily for engineers who plan to work in CAD or who already do. They will recognize experiences they may have had, placed in a more general context. They should also find useful ideas which they can put into practice in their own environment. The book is also intended for students who want to give themselves a broader fundamental background in CAD.

1.2 Scope of CAD

The meaning of "computer-aided design" (CAD) has changed several times in its past twentyfive years or so of history. For some time, CAD was almost synonymous with finite element structural analysis. Later, the emphasis shifted to computer-aided drafting (most commercially available CAD systems are actually drafting systems). Handling smooth surfaces, as required in ship-building and the automobile industry, became another key issue. More recently, CAD has been associated with the design of three-dimensional objects (this is typical in many branches of mechanical engineering). In this book, we consider CAD as a discipline that provides the required know-how in computer hardware and software, in systems analysis and in engineering methodology for specifying, designing, implementing, introducing, and using computer-based systems for design purposes.

Computer-aided design is often treated together with computer-aided manufacturing (CAM). We are not including CAM in this book, since CAM starts from data – preferably machine-readable data – that are produced in the design process, but CAM is not part of the design process itself. The same applies to computer-aided testing (CAT), computer-aided work planning (CAP), computer-aided quality assurance (CAQ), computer-aided maintenance, and the other areas which constitute computer-integrated manufacturing (CIM). Knowledge about the available manufacturing, testing, and maintenance capabilities certainly influences the design; but the methods applied in these other CA's are not the concern of this book.

Recently the term computer-aided engineering (CAE) has been used for summarizing all computer aids in design, while restricting CAD to computer-aided drafting. Here, however, we will continue to associate the term CAD with the wider meaning defined above.

Design is not only the more-or-less intuitively guided creation of new information by the designer. It also comprises analysis, presentation of results, simulation, and optimization. These are essential constituents of the iterative process, leading to a feasible and, one hopes, optimal design.

1.3 Content of the Book

In Chapter 2 we present briefly the history of CAD. The main components of CAD systems are identified, and their principal functions described. Economical and interdisciplinary aspects are discussed.

Chapter 3 starts with a systems analysis of the design process. The notion of a process is introduced as a fundamental tool to describe activities like design as a whole, computer-aided design, program executions, terminal sessions, etc. The environment and the resources which the environment must supply for the successful execution of any process are discussed. The problem of modeling the design objects in an abstract schema and the interrelation between the schema and the planning of the individual step in the design are analyzed.

Chapter 4 concentrates on the interfaces among the components of a CAD system, including the human operator. The problem of mapping an abstract schema onto the capabilities of various programming, command, or data description languages in described in detail. Emphasis is laid upon the resource aspect and its influence on the design of CAD systems. The concept of a CAD software machine is introduced, and rules for designing such machines are given.

Chapter 5 presents graphical input and output devices and describes their capabilities based on their principle of operation. Graphics workstations are described in terms of hardware architecture and graphics support. The most important features of computer networks are outlined as far as CAD systems are concerned. Finally, an introduction to a standard software interface to computer graphics hardware, the graphical kernel system (GKS), is given.

Chapter 6 presents selected engineering methods for CAD. Various numerical analysis methods (such as the treatment of geometry with numerical techniques, finite elements, simulation and optimization) are treated only to the extent that the reader may obtain select entry points into the extensive literature on these subjects. Not the methods themselves but rather their embedding into CAD lies within the scope of this book. Graphic techniques for presentation of numerical results are described in more detail.

Chapter 7 is concerned with CAD data exchange. This area has become extremely important since industry has realized that the greatest benefits from using CAD can be obtained only when CAD data can be exchanged without (significant) loss of information both between CAD systems and from CAD to subsequent processes in manufacturing. Various interface techniques have been developed, some of them have become national or international standards, for this purpose.

Chapter 8 gives selected examples of CAD applications taken from industrial practice.

1.4 Summary

The aim of this first chapter was to give the reader an impression of what he may expect to learn from this book. We have also explicitly indicated important areas which

will not be covered at all, or will be touched on only briefly (computer-aided manufacturing or finite elements, for instance). Readers who prefer to study just one topic or another are invited to jump to the pertinent chapters immediately.

1.5 Comments on the Second Edition

In the seven years since the first publication of this book an impressive expansion of CAD has taken place. The dominating evolutions may be summarized as follows:

- CAD has matured to an important factor influencing the economy of the industrial countries. On the user side, the need for justifying the introduction of CAD in a company is now practically replaced by the need to justify when CAD is not used for design applications;
- the tremendous development in computer hardware technology has led to a multiplication of the processing power, the storage capacity, and the graphic interaction potential that is now available to the user at his desk, at much reduced cost;
- CAD systems are now available in an investment range from a few thousand US dollars up to many millions. Hence, there is a solution for utilizing CAD for all companies;
- CAD is seen more and more as a central part of the whole manufacturing process (an issue which we deal with specifically in Chapts. 3 and 4) with the need for communicating the results of a CAD process to other processes. Hence, the development of standards for communicating CAD information, and the development of processors which perform such communication conforming to these standards has become vitally important.

1.6 Acknowledgements

The authors gratefully acknowledge the support of their organizations, Technische Hochschule Darmstadt and Kernforschungszentrum Karlsruhe, which have made their facilities available for the preparation of this book. Several experts have contributed to the contents of this book: The sections listed were written by the following contributors: R. Güll (Sect. 5.4), W. Hübner (Sect. 2.4), K. Klement (Sects. 2.1 and 6.1), D. Köhler (Sect. 2.5), L. A. Messina (Sect. 2.7), S. Noll (Sect. 2.3), J. Poller (Sect. 2.3), and M. Ungerer (Sect. 2.5).

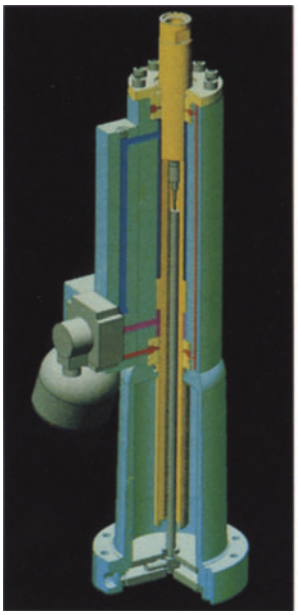
The laborious work of preparing most of the illustrations for this book was performed with great care by Mrs. U. Maier. We are much obliged to Mr. G. Becker who copy-edited the whole manuscript for the first edition of this book, and to Mrs. M. Christ for editing the complete second edition. Finally, we very much appreciate the ideal support we received from many experts in the field, both from Germany and other countries. Communication with them has been an invaluable help in collecting

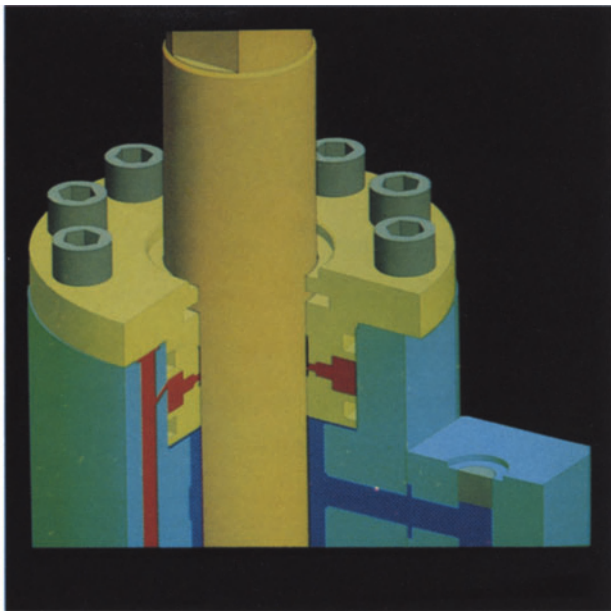
together the great variety of thoughts in the CAD world and presenting them here. We also wish to express our gratitude to our families, who had to spend many weekends without husband and father during the preparation of the manuscript.

1.7 List of Frequently Used Abbreviations

2D	two-dimensional
$2\frac{1}{2}$D	two-and-a-half-dimensional
3D	three-dimensional
AI	artificial intelligence
AP, APP	application program
B-REP	boundary representation
CAD	computer-aided design
CAE	computer-aided engineering
CAM	computer-aided manufacturing
CAP	computer-aided planning
CAQ	computer-aided quality assurance
CAT	computer-aided testing
CCD	charge-coupled device
CELL-D.	cell decomposition
CGI	computer graphics interface
CGM	computer graphics metafile
CIM	computer-integrated manufacturing
CODASYL	conference on data system languages
COM	computer output on microfilm/microfiche
CRT	cathode ray tube
CSG	constructive solid geometry
DB	data base
DBMS	data base management system
DBTG	data base task group
DC	device coordinates
DDL	data definition language
DIS	draft international standard
DML	data manipulation language
DP	draft proposal, design process
EP	economical parameters, environment process
GDS	graphics data structure
GKS	graphical kernel system
GKSM	graphical kernel system metafile
H/S	hardware/software
HPAT	hyperpatches
I/O	input/output
IS	international standard
LAN	local area network
MFLOPs	million floating-point operations per second
MIPs	million instructions per second
MIT	Massachusetts institute of technology
NC	numerical control
NDC	normalized device coordinates
NWI	new work item
OP	organizational parameters
PCB	printed circuit board
PHI-GKS	programmer's hierarchical interactive graphical kernel system
PHIGS	programmer's hierarchical interactive graphics system
PLA	programmable logic array
POL	problem-oriented language
TP	technological parameters
UI	user interface
UIM	unser interface manager
UIMS	user interface management system
VDS	virtual device surface
VLSI	very-large-scale integration
WC	world coordinates
WCS	world coordinate system
WD	working draft
WDSS	workstation dependent segment storage
WI	work item
WISS	workstation independent segment storage

2 History and Basic Components of CAD





Solid modelling of a hydraulic component
(courtesy of Control Data, Minneapolis, USA)

2.1 History

We will first give a brief review of the historical background of CAD. Knowledge about the history provides a better understanding of the present state of the art, and may even enhance the creativity of those planning to work in this field [ALLA73]. Up to 1978, this review is based on [BENE79]. Another summary on the computer graphics part of the history of CAD is given in [CHAS81]. Early in the 1950s, the Servomechanisms Laboratory at the Massachusetts Institute of Technology (M. I. T.) developed the first automatically controlled milling machine using the Whirlwind computer [PEAS52]. This led to the evolution of the Automatically Programmed Tool (APT) [BROW63]. We note that computer-aided manufacturing is not a descendant of CAD, but has a distinct origin of its own. The step from APT to design programs including computer graphics functions was outlined by Coons [COON63]. Sutherland, one of the first CAD pioneers, envisaged the designer sitting in front of a console using interactive graphics facilities developed at the Massachusetts Institute of Technology; he developed SKETCHPAD in 1963 [SUTH63]. The software principles of rubber band lines, circles of influence, magnification, rotation, and subframing were born in those days.

In 1964, General Motors announced the DAC-1 (Design Augmented by Computer) system [JACK64]. The hardware was built by IBM according to the specifications of General Motors Research Laboratories. DAC-1 was more concerned with producing hard-copies of drawings than with interactive graphical techniques. In 1965, Bell Telephone Laboratories announced the GRAPHIC1 remote display system [NINK65]. GRAPHIC1 utilized a modified DEC 340 display and a PDP5 control processor, connected to an IBM 7094. The system was used for geometrically arranging printed-circuit components and wirings, for the schematic design of circuits or block diagrams, for the composition and editing of text, and for the interactive placement of connective wiring. It was a very early implementation of the important idea of having the CAD processing power distributed among local interactive workstations and a central host computer.

In 1966, IBM Components Division described a system which was an aid to the design of hybrid integrated-circuit modules, as used in IBM's System 360 machines [KOFO66]. Freeman suggested, in 1967, an algorithm for the solution of hidden-line problems [FREE67]. A system called GOLD was developed in 1972 at RCA for integrated circuit mask layout [FREN70]. GOLD was implemented on a custom-made refresh display, driven by a small computer (Spectra 70/25) with a single disk, and was capable of interacting with a large time-shared computer. The first half of the 1970s was a time of much enthusiasm among the early CAD scientists and system developers [CLAU71]. Much theoretical work was done, laying down the fundamentals of CAD as we know it today. The Integrated Civil Engineering System (ICES) was developed [ROSS76], followed by a number of systems [SCHL74] which implemented many principal ideas regarding a CAD methods base. The theory of finite elements and associated programs started a booming development [PILK74]. At the same time, considerable research activity was going on in the areas of hidden line and surface removal [ENCA72].

The University of Rochester started the Production Automation Project in 1972. As a result of this project, two geometric modeling systems PADL-1 and PADL-2 were

developed [REQU82]. In 1973, a Lockheed review demonstrated that computer graphics will not only be practicable in the design process, but also cost effective [NOTE73]. In 1975, Chasen from Lockheed Aircraft Corporation published an analysis of the financial benefits of computer graphics in CAD systems [CHAS73], and Eastman described a data base for CAD. As a specialty within CAD, computer-aided drafting began to appear. It soon had such an impact on the field that, more recently, the term CAD seems to have become associated with the drafting part of design alone, while CAE (computer-aided engineering) has been used to include the analysis and optimization aspects of design. In this book, however, CAD is considered as a supporting discipline for the whole design process, which includes synthesis, analysis, and evaluation.

Hewlett-Packard announced in 1978 a microprocessor-based raster scan display terminal [DICK78]. Several publications by General Motors [RENO78] and Boeing [INMA78] in 1978 confirmed the usefulness of CAD/CAM technology and described how to bridge the gap between CAD and CAM (computer-aided manufacturing). The late 1970s may be characterized as the time of CAD's breakthrough from a scientific endeavor to an economically attractive and – in many areas – indispensible tool in industry. Governments became aware of this fact, provided funding and initiated projects to promote the integration of CAD technology, particularly with respect to medium- and smaller-sized industries.

Since the beginning of the 1980s, CAD has been fully developed in the market place, increasingly becoming a standard tool in design offices, and progressing in tandem with a steady correlation and adaptation of the work procedures there [JOHN86].

The efforts in research and industrial development have concentrated since the mid 1980s on the integration aspects of CAD as central part of CIM (computer-integrated manufacturing, which is also known emphatically as "factory of the future" [VERN84]. This embedding of CAD into a complex industrial automated process required a more general concept of internal model data. The term product definition data has been established. These product definition data contain as extension of the product geometry data application-dependent information like product material data, product tolerance data, and life-cycle information like product planning data, product manufacturing data, and product test data [STEP88].

In parallel and in addition to this integration an improved reference model for CAD systems has been worked out, which associates the construction-related tasks and the computational aspects of CAD in a reference matrix [RCAD88].

Not only the industrial nations but also the developing countries realized that CAD will be an essential constituent of practically all industrial enterprises in the near future [ENCA81], [LAST88], [CHIY88].

2.2 Modules, Functions, Components

Computer-Aided Design (CAD) means the usage of computer hardware and software for the design of products that are needed by society [DIEB76]. Products, in the

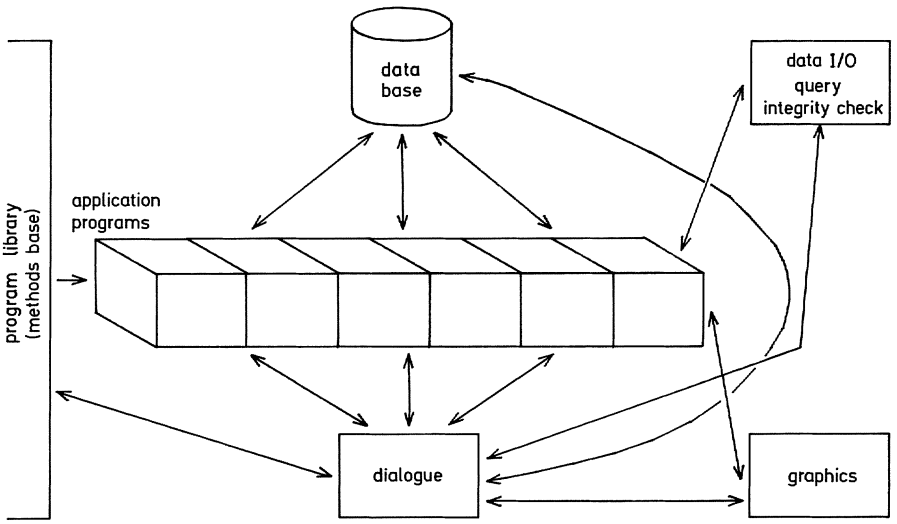


Fig. 2.1. The basic components of a CAD system

widest sense, are elements of some larger system: a transport system, a medical center, a factory planning project, etc. CAD means the integration of computer science methods and engineering sciences in a computer-based system, providing a data base, a program library (sometimes called a program chain or methods bank), and a communication subsystem (Fig. 2.1). The program library contains both the modules used for the elementary system functions (data base, dialogue, data I/O, graphics) and the modules that represent the algorithms of the application area. Data I/O implies the functions of inputting data, performing tests to guarantee the integrity and consistency of the data in the base, and querying the data base for data. Some of these application modules may be very large programs (for example, a module for finite elements). The communication subsystem includes modules for dialogues (commands addressed to the subsystem CAD and messages returned to the designer), for input and output of data, and for graphical information processing.

The dialogue modules comprise the command language of the operating system to the extent that is required to set up the appropriate environment for its operation. The CAD system itself supports a special command language for dialogue with the user. The data acquisition, the integrity check and the query language are the modules for data I/O. The graphical I/O and the interactive graphic dialogue are processed in the graphical information processing module. It has become a generally accepted practice to distinguish between modeling functions and viewing functions. While modeling, the user is actually communicating with a part of the application program chain for the definition of the problem, its topology, its geometry, and other properties. For viewing, the user communicates with a set of functions for the display and manipulation of graphical data, independent of the particular application. Hence, viewing functions may be collected in an independent graphical module [ENCA79].

The interface between model data and graphical data is realized by the so-called presentation module; this contains all information about the intended visual ap-

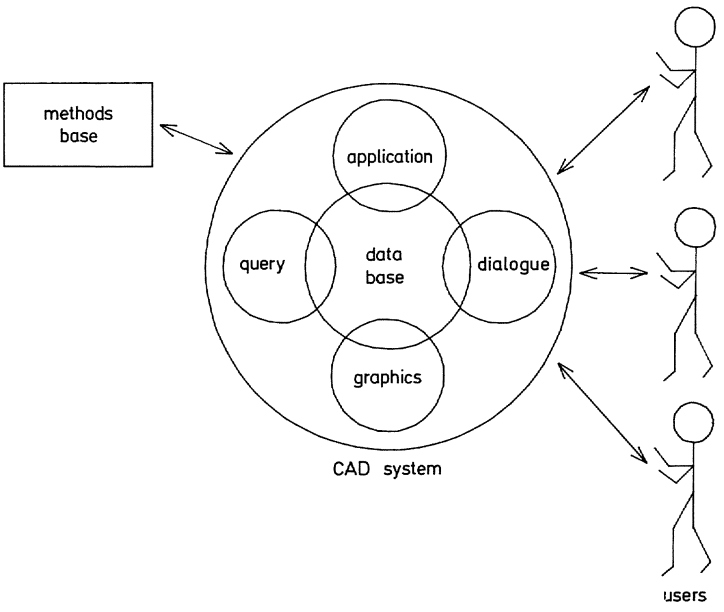


Fig. 2.2. The functional structure of a CAD system

pearance of the product definition data on an output device. This is information about, e.g., spatial and non-spatial selection of product definition data for visualization; presentation accuracy, which determines the approximation accuracy of the product geometry with graphical output primitives; inheritance schema for the related graphical attributes; and definition of an intended shading [KLEM88].

The border lines between modeling, presentation and graphical functions cannot be drawn unambiguously. They depend on the CAD systems capability.

Development in the graphics area itself is providing the graphical systems with more and more low level modeling and presentation functionality [PHBR88].

Figure 2.2 illustrates the functional structure of a CAD system, with emphasis on the central role of the data base. From a system point of view, CAD systems can be classified as:

- Time sharing
  Several applications: programs run on the same computer. They are independent from each other. The users only share the CPU usage.
- System sharing:
  Several users work with one common application program using the same computer, the same data base, etc. The users share the CAD system resources.

From a hardware point of view there are two principal CAD system structures (Fig. 2.3, [NEES78]):

- single programs; independent "stand-alone" systems; and
- terminal systems linked to a host computer and its data base.

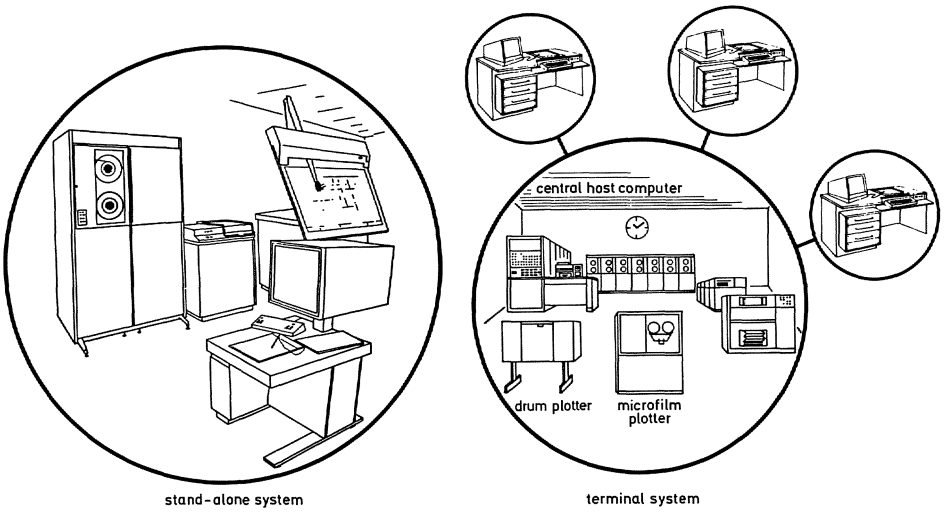


Fig. 2.3. Stand-alone and terminal systems

With the spread of open computer networks in recent years, four major types of modeling in networks have been established [DFN__86]:

- *Local modeling*
  All modeling work is performed on the local CAD system, but all or part of the product definition data is retrieved from remote systems via the network.
- *Remote modeling*
  All modeling activities are performed on a remote CAD system. The local system acts only as a detached workstation for the remote one. Only the initial and final results may be distributed via the network.
- *Resource-sharing distributed modeling*
  This requires homogeneously distributed modeling environments. Any resources are directly available to all workstations on the network and are shared between the different modeling processes.
- *Cooperating distributed modeling*
  This requires a common command interface and common neutral, but interactively processable product definition data for all CAD systems involved. This neutral product definition data is continuously updated on all systems participating in a particular modeling session.

From a software point of view, one may distinguish between [SPUR81]:

- Black-box (turn-key) CAD systems:
  These have a predefined set of functions, generally operating only on specific hardware. They possess a menu-driven user interface and a command language for sequential call-up of the predefined functions. There exists no other way for communicating with black-box systems;

- Freely programmable (open) CAD systems
  These possess in addition a programing language, which allows extension of the set of predefined functions. The programming language is generally still FORTRAN [CCRE87].

As a result of the ongoing integration of CAD into CIM the functionality of CAD systems is increasing in two independent directions. On one side they tend to serve several mostly similar application areas. On the other side they consider the traversal tasks arising from CIM integration [CIM_88]. This extended funtionality like, e.g., CAP (computer-aided planning), CAQ (computer-aided quality assurance), CAT (computer-aided testing) is interfaced and handled with the help of additional programs and modules [ISIS86].

2.3 Graphics Standards

Standards specify interfaces for exchanging information between parts of a graphics system. The usage of standards provides portability, extendability, longevity, device-independence and functionality of the applications and system components.

Computer graphics standards are developed within the International Standardization Organization (ISO) in Sub-Committee 24 (SC24), which belongs to the Technical Committee 97 (TC97) for information processing. Sub-Committee 24 has several Working Groups (WG's), which develop the several Graphics Standards.

The ISO process starts when a New Work Item (NWI) proposal for a graphics standard is drafted by a member body or a Sub-Committee. After a letter ballot (each member country has one vote) the NWI can be accepted as a Work Item (WI). The Work Item (WI) or project is then assigned to a Rapporteur Group, created by the working group (WG). In international meetings the Rapporteur Group prepares Working Drafts (WD) from the base document. The Working Group (WG) can recommend that SC24 register the Working Draft (WD) as a Draft Proposal (DP). If the letter ballot that follows is successful, the Draft Proposal (DP) is registered and gets an ISO number. When the DP is technically stable and international consensus is reached it can become a Draft International Standard (DIS) after another letter ballot.

The final status is International Standard (IS). Only an IS is a valid standard.

2.3.1 Reference Model for Computer Graphics

The computer graphics committee of ISO is developing a basic reference model for computer graphics. This model is needed to define the relations between the several graphics standards. The reference model is also used to clarify the relations between the graphics standards and other related ISO standards, such as the Open System Interconnection (OSI) reference model for computer communication.

The reference model described here is based on a classical view of graphics system architecture, which sees a high-level interface, with a completely device-independent

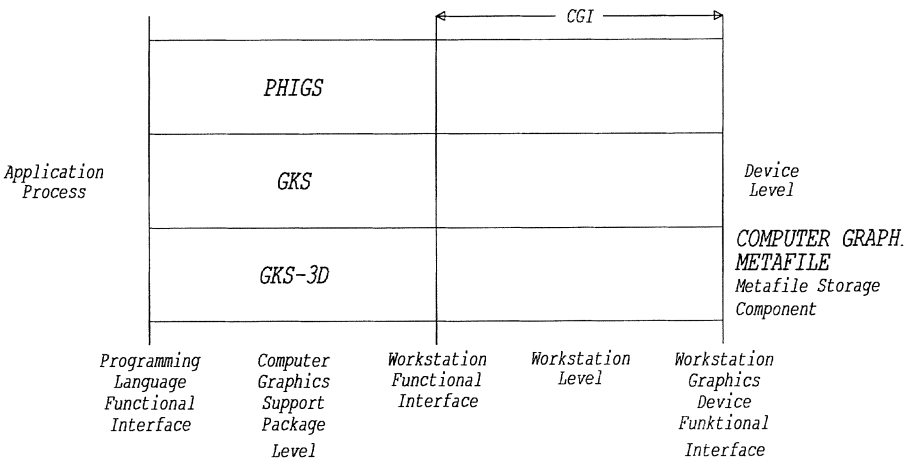


Fig. 2.4. Computer graphics reference model and graphics standards

view of graphics, provided to application programmers. This level is followed by a series of lower level interfaces, such as workstation and device level, across which would flow device-dependent information.

The reference model contains the following components and interfaces (Fig. 2.4):

- application process,
- programming language binding functional interface,
- computer graphics support package level,
- workstation functional interface,
- workstation level,
- workstation graphics device functional interface,
- device level, and
- metafile storage component.

Classes of processing, such as attribute, transformation, clipping, and dimensionality, are each described as a sequence of processing steps. The collection of all processing steps of a class is called a strand. Elements from various strands are then interleaved into streams, which describe the sequence of processing associated with each graphical function.

Currently there are some work documents developed within ISO. The final disposition of the reference model is unclear, but doing good progress.

2.3.2 Graphical Kernel System (GKS)

GKS, ISO IS 7942, was the first international standard for computer graphics. It was developed between 1980 and 1984 and became an IS in 1985.

GKS is a 2D graphics system that provides graphical output and input in a device- and language-independent manner.

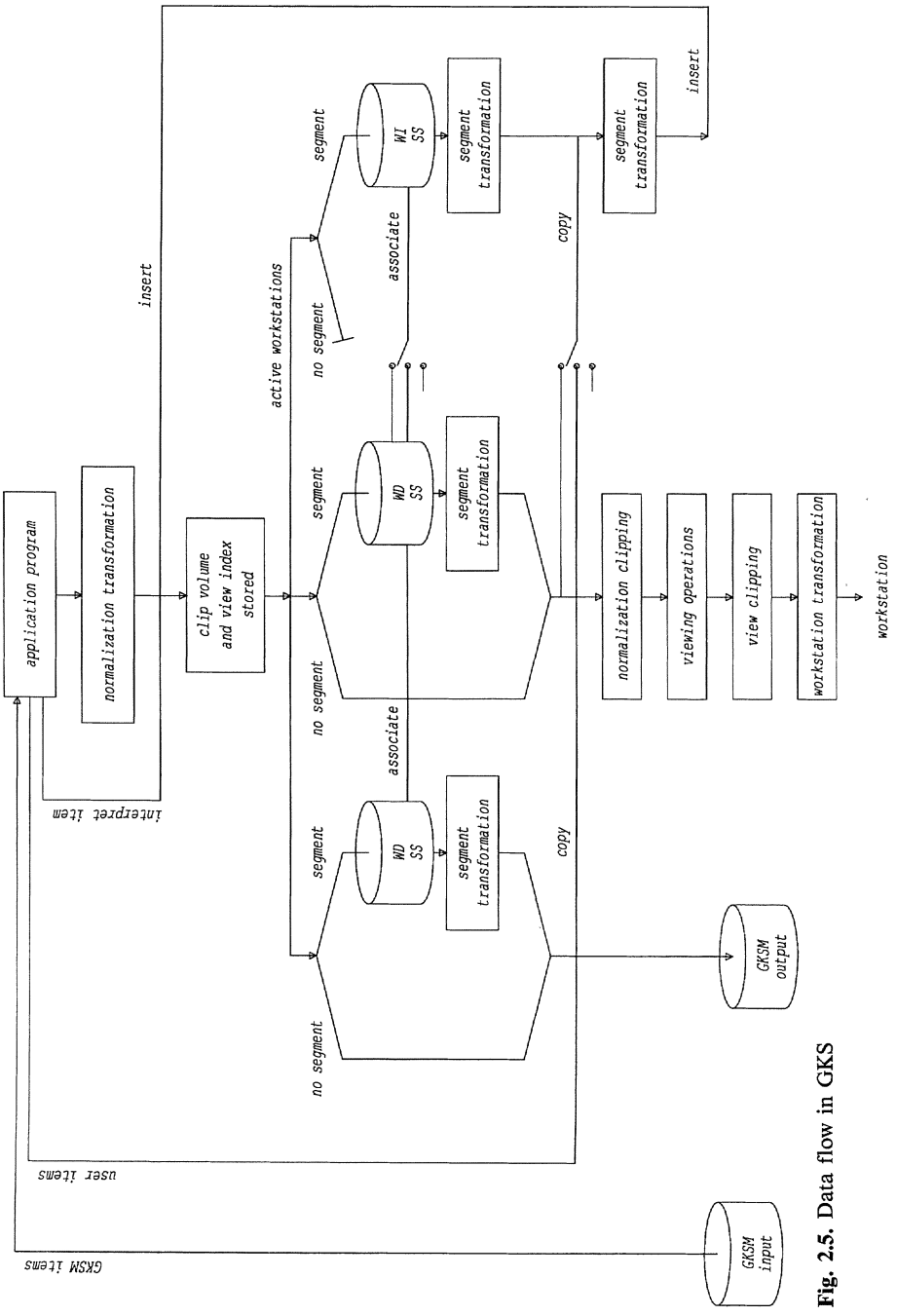


Fig. 2.5. Data flow in GKS

Six output primitives are defined:

- Polyline,
- Polymarker,
- Fill Area,
- Text,
- Cell Array, and
- Generalized Drawing Primitive.

GKS uses the concept of a workstation, which is an abstraction from physical devices. More than one workstation can be used simultaneously for output and input (Fig. 2.5).

Transformation to the coordinate system of the display device is accomplished in two stages:

- normalization transformation, maps from world (user) coordinates (WC) to normalized device coordinated (NDC), and
- workstation transformation, maps from NDC to device coordinates (DC).

Both transformations are window-to-viewport mappings. Graphical output within segments (see below) is also transformed via the segment transformation matrix (2×3).

The appearance of the output primitive on the display is controlled by aspects. For each graphical primitive a set of aspects is defined, such as linewidth, colour, and linestyle for polylines. These aspects are bound in several ways to the primitive.

In GKS, output primitives and primitive attributes may be grouped together in a segment. Segments are the units for manipulation and change. Manipulation includes creation, deletion, and renaming of segments. Change includes transforming a segment, making a segment invisible, highlighted or detectable. Segments also form the basis for workstation-independent storage of pictures. Via this storage, which is set up as a special workstation called workstation-independent segment store, segments can be inserted, copied, and associated to other workstations.

For graphical input there are six logical input devices defined:

- Locator,
- Stroke,
- Valuator,
- Choice,
- Pick, and
- String.

The logical input devices can operate in request, sample, and event mode.

For implementations GKS defines nine levels of functionality to support. The simplest (0a) has no segments and no input, while the most complex one (2c) has all facilities including workstation-independent segment storage and event input.

2.3.3 Computer Graphics Interface (CGI)

CGI, ISO DP 9636, has been developing since 1985 to provide a standard range of facilities to be used in communicating with graphics devices. CGI is useful for the

device-independent/device-dependent interface between functional computer graphics support packages, such as GKS/GKS-3D and PHIGS, and graphics devices.

The output primitives and aspects are very similar to GKS. Some new primitives have been added to support special devices: Circle, Rectangle, and Pixel Array. Another extension to GKS/PHIGS is the possibility of closed figures. A closed figure is a set of connected output primitives which define a filled area. Other CGI concepts for segments and input are also very similar to GKS.

One main difference with aspect to GKS is the raster part of CGI. This CGI part defines the usage of bitmaps for graphical output. Bitmaps are rectangular arrangements of pixels, and are used for storing and modifying graphics output data via raster operations. Unlike GKS/PHIGS, CGI has no workstation concept, because CGI is conceptually a single-device interface.

Like GKS/GKS-3D, CGI is a configurable standard. The functionality of a specific CGI implementation is defined via a profile which can define a lot more levels, as in GKS.

2.3.4 Computer Graphics Metafile (CGM)

CGM for the storage and transfer of picture description information, ISO IS 8632, was the second international standard for computer graphics. It was developed between 1982 and 1987 and became an IS in 1987.

The CGM standard has two distinct roles: the first is to define the functions that need to appear in the metafile and the order and position of the various elements. The second role is to define the way that these functions are recorded in the metafile.

CGM is very closely related to the CGI definition in its output functions and primitive attributes. Graphics metafiles provide:

- a data format for picture archiving,
- a graphical protocol for off-line and off-site plotting,

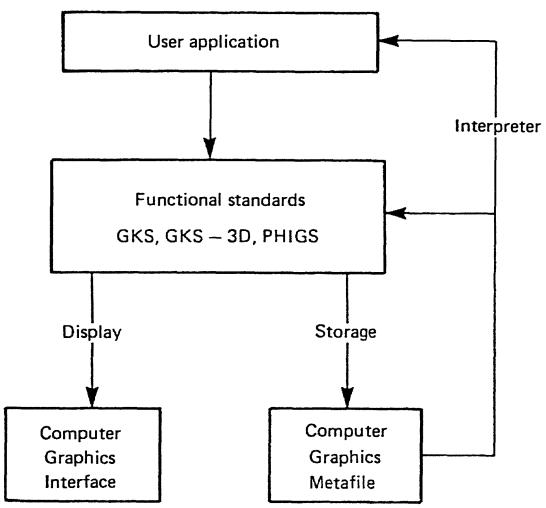


Fig. 2.6. Relationship of CGM to other graphics standards

- a single format for spooling to multiple dissimilar plotting devices,
- the possibility and impetus for a single standard interface to picture-generating devices,
- a way to reuse the same picture without recomputing it,
- a basis for session save/restart mechanisms, and
- the glue for unifying and integrating distinct graphics applications and hardware/software systems in a distributed environment.

The CGM is closely related to the standards GKS, GKS-3D, and PHIGS (Fig. 2.6) and also to CGI.

2.3.5 Graphical Kernel System for Three Dimensions (GKS-3D)

After the definition of GKS the development of a 3D extension, GKS-3D, was started in 1982. This is a fully upward-compatible system that allows 2D-GKS programs to run in the 3D environment. GKS-3D is already a Draft International Standard, ISO DIS 8805.

The main features added to GKS are:

- 3D primitives,
- fill area set primitives,
- edge aspects for fill areas,
- 3D geometric aspects,
- 3D transformation pipeline with viewing operations,
- 3D input pipeline, and
- access to hidden-line/hidden-surface removal.

GKS-3D extends each GKS primitive to three dimensions, and it adds one new primitive, fill area set. Fill area set allows one to define a filled area with more than one polygon.

The same aspects for output primitives as in GKS are used in GKS-3D. Extensions are the edge control for fill area sets, and the 3D geometric aspects for text and fill area/fill area set.

The workstation-independent part of the transformation pipeline contains the normalization transformation (3D-window to 3D-viewport), the segment transformation (3×4 matrix) and the normalization clipping (rectangular parallelepiped oriented with edges parallel to the normalized device coordinate axes).

Viewing transformations are defined in a table at each workstation. In the first viewing stage objects are transformed by a 4×4 view orientation matrix. In the second stage the objects are transformed by a 4×4 view mapping matrix and clipped at a 3D volume. The projection type of the view mapping may be parallel or perspective.

After the viewing transformation, the workstation transformation maps from a 3D window to a 3D viewport in device coordinates (Fig. 2.7).

The input devices are extended to three dimensions analogous to the output. The inverse of each transformation in the whole transformation pipeline is applied to input coordinates.

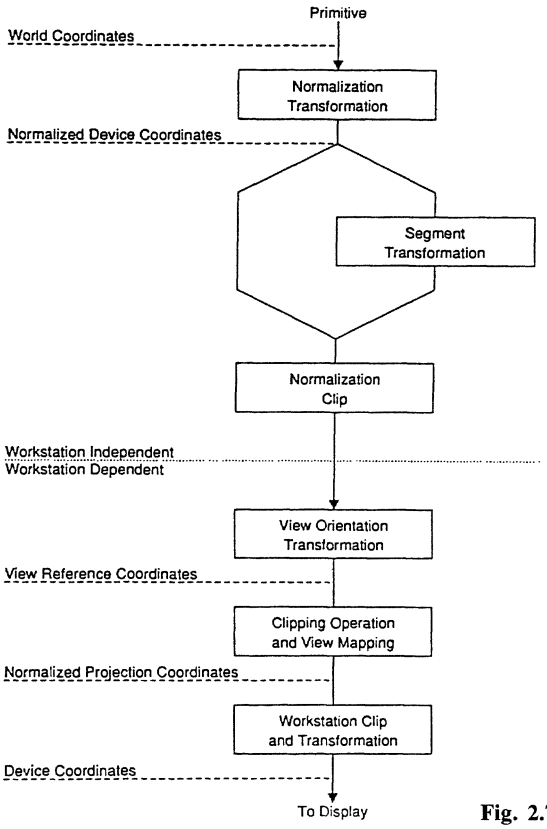


Fig. 2.7. GKS-3D transformation pipeline

GKS-3D also supports the use of workstation-dependent hidden-line/hidden-surface removal which can be defined for each output primitive.

2.3.6 Programmer's Hierarchical Interactive Graphics System (PHIGS)

The proposal of PHIGS, which was brought into ISO in 1985, is also an International Standard, ISO IS 9592.

PHIGS defines that the output primitives and their attributes generated at structure traversal time are identical to those of GKS-3D.

PHIGS addresses different needs using new functionality referring to GKS-3D. The concepts of hierarchical picture definition, editing of the data structures, and modeling of the objects are the major points of extended functionality (Fig. 2.8).

PHIGS supports the storage and manipulation of data in a centralized hierarchical structure store. The fundamental units of data are structure elements and these are grouped together into compounds called structures, which are organized as networks.

Structure elements contain the information for the definition of output primitives.

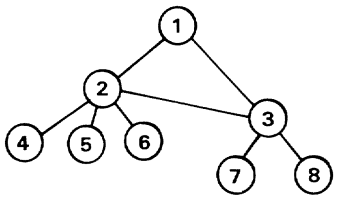


Fig. 2.8. Relationships between structures

Each output primitive has associated with it a name set attribute defined at structure traversal time. This name set attribute defines the eligibility of the primitive for invisibility, highlighting, and detectability. The workstation-independent primitive attributes are specified modally and are bound to a primitive when it is created at structure traversal time. Graphical output on a workstation is produced by traversing a posted (associated) structure on that workstation and interpreting the graphical structure elements (traversal time).

The workstation-independent part of the transformation pipeline contains the modeling transformation (4×4 matrix) and the modeling clipping (set of half-spaces combined with boolean operators defining the acceptance regions). The workstation-dependent part of the transformation pipeline and the input model are identical to GKS-3D.

Besides the metafile interface to CGM, PHIGS defines a second way of saving, transferring, and restoring graphical data: the archive file. The archive file allows random access to structure networks.

2.3.7 Programmer's Hierarchical Interactive Graphical Kernel System (PHI-GKS)

The approach of PHI-GKS describes the concepts which offer the functionality provided in PHIGS, and ensures compatibility to GKS and GKS-3D (Fig. 2.9).

As a slight deviation from GKS-3D the workstation independent segment store (WISS) must always be existent in PHI-GKS. This would compare to an open and active WISS in GKS-3D, but in PHI-GKS the operating state is not influenced through the WISS. All generated segments are stored within the WISS.

In PHI-GKS a segment is defined as a collection of segment elements. Segment elements may be attribute selections, labels, application data, name set specifications, transformation selections, segment references, or elements causing the generation of output primitives at segment traversal time. The segment priority is a workstation-dependent segment attribute which can be set for open or active workstations. The segment attributes are not global values for the whole segment, but initial values at the beginning of the segment, which can be dynamically set without opening the segment.

The workstation-independent attributes of output primitives are stored as separate segment elements within the PHI-GKS WISS. The effective attributes of a primitive are evaluated at traversal time, as in PHIGS.

In addition to PHIGS, an initial set of primitive attribute values can be defined in the segment state list. This initial set is used during editing to copy the current attribute state into the segment, as in GKS/GKS-3D.

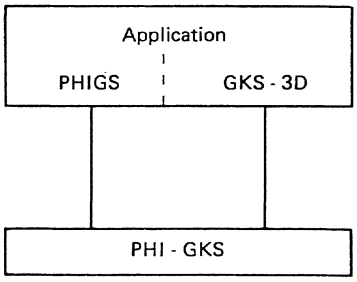


Fig. 2.9. PHI-GKS directly called by GKS-3D and PHIGS applications

All coordinates of output primitives and attributes are transformed with the actual normalization transformation before generating the segment element in the WISS.

With respect to the new higher functionality the compatibility with GKS/GKS-3D cannot be a hundred percent. This leads to minor changes or restrictions in a few GKS/GKS-3D programs.

2.3.8 Language Bindings for Graphics Standards

Language bindings for graphics standards are being generated according to guidelines designed to provide portability. Since current language standards are used as a baseline, portability between hardware configurations is achievable whenever compilers conforming to standards are available. Application program portability is achieved between standard-conformant implementations by giving a single standard binding for each language.

Separate bindings for each programming language have been developed. Language bindings for the new graphics standards are being developed in FORTRAN, Pascal, Ada, and C.

The following bindings exist as DIS, DP, WD, or WI end of 1989:

– GKS	FORTRAN	ISO IS	8651-1
– GKS	Pascal	ISO IS	8651-2
– GKS	Ada	ISO IS	8651-3
– GKS	C	ISO DP	8651-4
– CGI	FORTRAN	ISO WD	
– CGI	C	ISO WD	
– GKS-3D	FORTRAN	ISO DIS	8806-1
– GKS-3D	Pascal	ISO WD	8806-2
– GKS-3D	Ada	ISO DIS	8806-3
– GKS-3D	C	ISO DP	8806-4
– PHIGS	FORTRAN	ISO IS	9593-1
– PHIGS	Pascal	ISO WD	9593-2
– PHIGS	Ada	ISO IS	9593-3
– PHIGS	C	ISO DP	9593-4

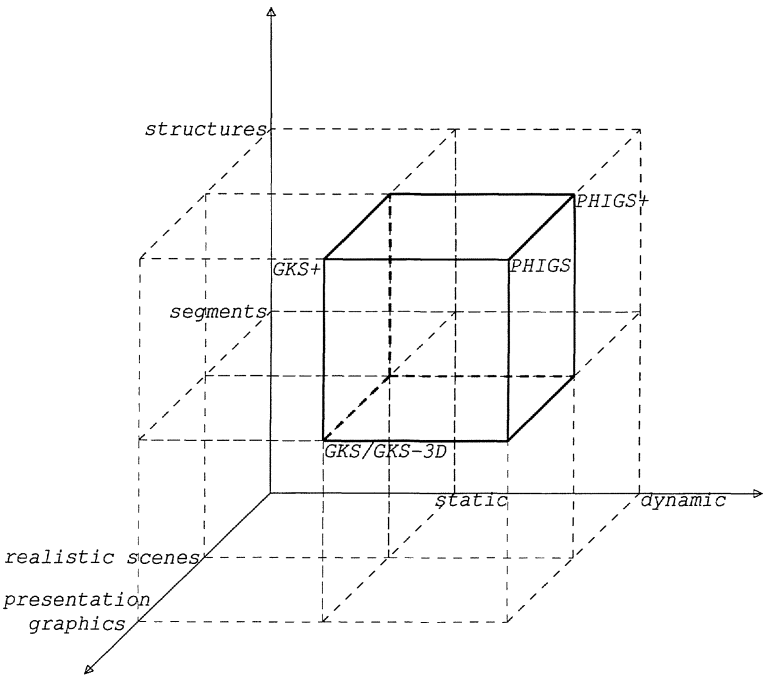


Fig. 2.10. Relations between graphics standards

2.3.9 Future Development

The first main objective for the future is to finish all current projects and bring all Work Items (WI) to Working Drafts (WD), then to Draft Proposals (DP), Draft International Standards (DIS), and finally to International Standards (IS). Such work has to be done in all work areas.

The second main topic is the GKS review started in 1987. A review process for a standard is done every five years. The review process has several goals such as corrections, clarifications, extensions (GKS+), etc.

Another objective is to extend PHIGS for lighting and shading (PHIGS+). Further extensions will then be included in PHIGS++ (Fig. 2.10).

The standardization of a window manager and the clarification of the relation to standards like GKS/GKS-3D and PHIGS are also a very important goal for future development.

2.4 The Graphical Dialogue System

One of the most important basic components of interactive computer graphics in CAD systems is the input and manipulation of complexly structured objects via their

graphical representation. That forms the human-computer interface (also called user interface) playing a central role in the successful use of CAD systems. Although the communication part of a CAD system is one of the most relevant components for acceptance by the user, user interface design is sometimes handled more like an art than like a science.

One of the reasons is that in the past well-organized structures of the user interface, design methodologies, and development tools were often missing. Therefore we will discuss in this chapter a formal model of the user interface, the language model of Foley and van Dam [FOLE84b], structuring human-computer interaction into lexical, syntactical, semantical, and conceptual levels.

We will then discuss some interaction styles like graphics interaction techniques, command languages, menu selection, and multi-windowing.

Beyond GKS as the standard for graphics systems, several tools like window managers, user interface toolkits and User Interface Management Systems have been developed recently to support specific tasks in the design and realization of user interfaces. A short introduction to such tools is given. At the end of this section THESEUS is described which tries to incorporate the tasks of the tools mentioned above within one integrated system.

2.4.1 The Language Model

Foley and van Dam [FOLE84b] tried to model the human-computer dialogue in analogy to languages for interpersonal communication. In a CAD system a dialogue language is the means to describe the user's action for creating and manipulating products to be designed.

This language is structured into four layers:

- conceptual level,
- semantical level,
- syntactical level, and
- lexical level.

The conceptual level incorporates the main concepts of the CAD system as seen by the user. It forms the user's model which is defined by operations, like drawing tangents to circles; objects or classes of objects, like tangents and circles; and the relationships between objects (e.g., topological, geometric, and functional relationships). For this purpose, task analysis methods can be used.

The semantical level describes the real functionality of the CAD system, i.e., the meaning of the operations, the parameters, constraints, semantical errors and their handling, and the feedback which is usually given by changing the displayed images and messages. Formal or semi-formal specification tools can be used for this level.

The syntactical level defines how the user can execute the functionality defined at the semantical level. Each function is decomposed into single user actions called tokens representing minimal interaction units. The functionality defined at the semantical level can then be described as sequences of those input tokens combined with output tokens changing the graphical appearance of the modified objects.

Finally, the mapping of tokens to physical devices is the task of the lexical level. These tokens represent the basic interaction techniques and output primitives. Typical examples are specific menu types, pointing mechanisms, or rubberband line drawing. The input tokens may consist of the logical input devices of GKS [ISO__85] or of basic tokens of another I/O package which manages the mapping to physical devices.

This language model provides a valuable framework for structuring dialogues into more or less independent abstraction levels that can be designed and implemented separately. CAD systems with rich graphics capabilities need some conceptual separation of syntactical tasks (dialogue sequencing and output organization) from lexical tasks of basic input/output control. Yet sometimes no sharp separation from the semantical level can be drawn. Semantical information can be useful in giving syntactical and lexical prompts and feedbacks; while the user is picking objects to be deleted with, e.g., the mouse, those objects that are allowed to be deleted should be highlighted while the mouse cursor is over them. For this kind of lexical prompt semantical information is necessary.

This layer model allows the definition of design criteria tailored to each layer instead of talking about "user-friendliness" in general. For each layer one can define specific evaluation categories so that a system can be improved by local modifications.

The four-level separation has been used successfully in designing many user interfaces and should be used as an organizational framework in specifying human-computer communication.

More recently, several investigators have used an object-oriented approach that allows very flexible definition of the user interface components. In order to develop efficient interfaces using that approach more research work must be done.

2.4.2 Interaction Styles

A multitude of interaction techniques can be used and combined to form the lexical tokens of a CAD user interface. Each interaction technique, like selecting a menu item or rotating an object, is connected to some devices. Selecting a suitable interaction technique based on the devices available, with respect to human factors, is one of the most challenging tasks in implementing CAD systems.

We will now classify the most important interaction styles into the categories of graphics interactions, menu selection techniques, command languages and multi-windowing.

2.4.2.1 Graphics Interaction

Most human-computer interactions in a CAD system are executed using graphical devices like the tablet, mouse, lightpen, joystick, touch pad, trackball, thumb wheels or digitizer. Operating with those devices, the user provides graphical information like positions, lengths, or basic objects as the building blocks for designing products like car surfaces, electronic circuit layout, or architectural constructions. Graphical repre-

sentations of the design products displayed on the screen can then be modified via input of graphical control functions like moving, stretching, or scaling objects.

For the graphics interaction task of providing input data, the logical input devices of GKS are a good characterization. The six types of input devices are:

Input device	Input value
choice	integer
locator	position
stroke	list_of_position
valuator	real within range
pick	object identifier
string	list_of_character

The interaction techniques for those interaction tasks depend on the input devices, visual aids like prompting, echoing, and feedback, and involvement of semantical support during input operation.

Beyond the task of providing data, graphics representations have to be modified interactively. For that purpose Foley, Wallace, Chan [FOLE84a] list, among others, the following controlling tasks:

- Stretch:
  The user picks a point or a part of an object and moves this part to a new position with the rest of the object remaining at its place. This mechanism allows distortion of the shape of a figure; e.g., moving one corner of a rectangle while the opposite one is fixed changes the size of this rectangle. Continuous feedback allows the user to adjust the shape correctly.
- Manipulation:
  An object is moved as a whole without changing shape, but changing its position, scaling factor, or orientation. Changing the reference point of an object by moving it continuously with the pointing device is called dragging. Related objects like connection lines to other objects or related text is updated automatically. Twisting is another manipulation technique used to rotate 2D or 3D objects. The user defines the axis to rotate about. Continuous or discrete feedback is possible. Finally the size of an object can be manipulated by changing the scale factor to make the object appear larger or smaller.
- Shape:
  Smooth curves or surfaces represented, e.g., by bicubic splines or as Bézier curves or patches can be manipulated to change their shape. Selecting and dragging one or more control points is the common interaction technique for this purpose.

Many more graphical interaction techniques are used in CAD systems which are often tailored to requirements of the specific design area.

Graphical representations and graphics interactions of a specific kind of interface which is seen, e.g., in desktop programs or WYSIWYG-editors (What You See Is What You Get) is called "direct manipulation" [SHNE83]. Direct manipulation interfaces provide continuous graphics representation of the objects of interest, physical

actions instead of complex syntax, and rapid reversible operations whose impact on the object is immediately seen [SHNE82]. Around this paradigm much research has been done into "directness" of action, directness of the translation of intention to action, and directness in the feedback and knowledge of the system [HUTC86].

Direct manipulation can influence CAD design by providing "direct" user interfaces close to the real design task domain without an additional abstraction level for human-computer communication. In such interfaces the underlying computer system should be more or less invisible. For further details about direct manipulation see [SHNE87], [HUTC86].

2.4.2.2 Menu Selection

Menu selection is the most frequently used interaction style in CAD systems. The user makes a selection from a predefined set of alternatives. Menu items usually represent a collection of functions and/or data values offered to the user at a specific dialogue step.

Typical selection techniques are: pointing with the lightpen, touching a sensitive panel, controlling the cursor with the mouse, selecting with a tablet, typing a label on the keyboard, hitting a function key, or voice input of the menu item name.

Menus are an excellent means to structure and group system functionality and to avoid errors by restricting user input to those choices that are allowed in a particular situation. Menus also minimize learning tasks and reduce keystrokes, and are therefore suitable for inexperienced users having only a short training phase. However, experienced users familiar with systems functionality may lose time if the physical selection mechanism is not fast enough. In this case, macro definitions mapping menu selection sequences to one item should be offered.

A menu can itself be an element in a menu at the next-higher level of hierarchy. In that way, trees of menu hierarchies can be created. The user navigates through the tree by selecting an item that offers him an additional menu. Several studies concerning depth (number of levels) and breadth (number of items) recommend no more than three to four levels and four to eight items per menu [SHNE87]. Only for purposes of detailed structuring and organizing functionality may more tree levels be useful, but to speed up performance the advantage of breadth over depth has been confirmed, especially in cases of slow display rate.

Rules for grouping of items and ordering them within a menu are hard to state. Clustering related items or sequencing items by alphabetic order, frequency of use or importance are aids for novice users to reduce searching time. In Card's experiments [CARD82] performance was best with alphabetic order. Nevertheless, significance of naming and iconizing is of much more importance than ordering of items.

Permanently displayed menus should be reduced to that functionality which must be always available. Other menus like hierarchical menus or functions that are selectable only in specific cases should be invisible until they are really needed.

Menus can appear at the user's request. Pop-up, pull-down, or drop-down menus become visible in response to pressing down a pointing device. The contents of a pop-up menu depends on the position of the pointing device, while pull-down and drop-down menus appear by clicking at the menu title permanently visible.

2.4.2.3 Command Languages

Menus are a selective interaction style because the system offers the alternatives directly to the user. In contrast, a command language must be typed in explicitly and is therefore called an imperative dialogue technique.

Functions and parameters are typed in on the keyboard. Command languages provide a high degree of user flexibility and initiative (user's locus of control) and are therefore appropriate for experienced users. Macro mechanisms allow the sequencing of several commands for minimizing typing and response time. To speed up typing performance, abbreviations and mnemonics are often used. Requiring intensive user training, this interaction style is most suitable for frequent users who memorize the language.

Command languages require a high degree of abstraction from the user's design task to its description in a language. Moreover, it is unwieldy for the user to operate with graphical input devices and the keyboard in parallel. Therefore, other interaction styles like menu selection and graphical input are preferable presuming they provide the same degree of flexibility and performance.

2.4.2.4 Multi-Windowing

Many CAD systems show several perspective views of models (especially in 3D applications) at the same time, together with additional information, help and error messages, menus, etc. Therefore, the graphics display is a scarce resource. To tailor the screen efficiently, windowing systems were developed. The screen is subdivided into several rectangular areas each called a window. Windows can overlap and differ in size and content. The goal is to present an optimum of information to the user by restricting the amount of visual information to that which is of relevance in a specific situation.

Each window shows a cut-out of a picture. The user can modify the appearance of the windows and their contents at any time. The following window operations are convenient:

- open and close a window
  a window is created or destroyed;
- move a window
  changing the position of the window on the screen with fixed size and content;
- resize a window
  changing width and/or height of a window without scaling the window contents so that more or less information is visible inside the window;
- scale a window
  modifying the scale factor of the window contents without changing the size of the window frame;
- zoom a window
  changing size of the window frame and its contents;
- pan a window
  the visible contents of a window are moved without changing the size of the window frame or the scale factor of the window contents;

- scroll a window
  the visible contents of a window are moved horizontally or vertically using scrollbars or sliders;
- cut and paste
  select an area of one window and copy it to another;
- top a window
  a window which was overlapped by some others becomes totally visible, i.e., it is the topmost in the hierarchy of overlapping windows;
- bottom a window
  it becomes the bottommost window; and
- iconize a window
  a window is shrinked down to an icon that represents it.

These operations are mostly offered by a pop-up window or by boxes and bars in the window border. A window manager is usually responsible for handling these operations. In an overlapping system the window manager controls the hierarchy of windows. A system without overlapping is called a "tiled" window system. In this case all windows are visible, and the window manager splits up the screen into rectangular areas. A comparison of these two types of window management is given in [BLY_86].

Windowing becomes very popular in CAD systems because it enables the user to work in several contexts and views in parallel, while the visible information is tailored to the current context. For further details about window design see [CARD84], [HOPG86].

2.4.3 User Interface Design Tools

One conclusion on dialogue design for CAD systems is that one should use existing user interface design tools that allow one to describe and realize dialogues. Several tools have been developed to support the lexical and syntactical level of dialogue design and implementation. They can be categorized into four classes: graphics systems, window management systems, user interface toolkits and User Interface Management Systems (UIMS).

2.4.3.1 Graphics Systems

Graphics systems are usually used for both graphical output and input. The input component of, e.g., GKS provides logical input devices of six different input classes, each defined by its input value. The input classes locator, stroke, valuator, choice, pick and string are described in detail in Sect. 2.3.2 and Sect. 5.5.1. For each logical input device several predefined prompt/echo types are offered, e.g., for locator a rubber-band, crosshair, and tracking cross. At initialization time of a logical input device one prompt/echo type is selected to be valid. The standard defines a number of prompt/echo types for each input class; a minimum of one must be implemented. The initialized logical input devices form the lexical level of the user interface. Lexical

feedback is given by GKS through the prompt/echo type. The GKS approach frees the user interface designer from defining the lexical level, i.e., defining the input tokens and binding them to physical devices.

The syntactical level of ordering input tokens into a dialogue must be realized on top of GKS. It is often mixed with the semantical application tasks but can be separated by defining a command interpreter or dialogue handler as an intermediate level between GKS and the application.

GKS provides three types of communication mechanisms called input modes: request, sample, and event. Request is used when the application needs a specific input value of an input class from an input device to continue processing; the application waits until a trigger action happens, then the current measured value is sent to the application. In this case, input is strictly sequentialized.

Sample mode is similar to request mode except that no explicit trigger action is necessary to return the input value to the application. The application does not wait for an operator action.

Very flexible user interfaces that enable the user to operate with several input devices in parallel can be realized using GKS event mode. Event mode allows a set of events to be generated simultaneously by a single trigger action. Each input event is written into the event queue time order. The application is able to read the head of the event queue.

With the event input mode GKS provides a sophisticated model which is on one hand very flexible to use, but on the other hand hard to implement. Therefore, the first generation of GKS implementations were often restricted to level 2b, i.e., were without sample and event mode. But with the growing importance of flexible user interfaces more GKS level 2c implementations are becoming available.

For more details about GKS see Sect. 2.3.2 and Sect. 5.5.1. Other graphics systems like PHIGS, PHIGS+, PHI-GKS, or CGI provide more or less the same input model.

2.4.3.2 Window Management Systems

The appearance of bitmap displays has brought on the creation of numerous window management systems allowing the designer to communicate with different contexts in parallel and to represent simultaneously multiple objects and tasks, e.g., several views of the same object. Currently, window management systems form an essential component of advanced user interfaces. Nearly every workstation provides multi-windowing capabilities as a basic service. A general architecture for window management systems was presented at the Workshop on Window Management [HOPG86] by Williams [WILL86] and is shown in Fig. 2.11.

The lowest level of basic input/output transforms device-specific functionality and hardware capabilities to a device-independent level, and vice versa. This component can be realized, e.g., by CGI (Computer Graphics Interface) [ISO_86b], a proposal for standardizing the device interface.

The next layer of Window Management Services is responsible for handling rectangular areas on the screen which may overlap, relating input events to windows, distributing them to clients, and mapping bitmaps into windows.

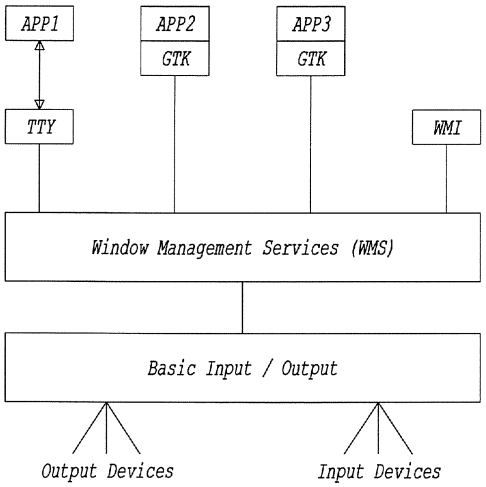


Fig. 2.11. Window management system architectural model

WMI (Window Management Interface) can be seen as a specific application which is responsible for managing all user operations concerning windows like move, size, etc.

GTK (Graphics library and Tol Kit) is a graphics library like GKS but can also be a user interface toolkit. It is a subroutine package linked to the CAD system. In addition, other applications or system services like mailboxes, text editors, or calculators can run in parallel in other windows.

An industrial de-facto standard has been established by X-Windows [SCHE86], a Window Management System that was developed at MIT, sponsored by DEC and IBM. A consortium of hardware companies including DEC, HP, Apollo, SUN, and Data General have announced support for the X-Window environment on their systems.

The American National Standards Institute (ANSI) is working on a standard in Display Management and developed a first proposal to standardize architectures and functionality of Window Management Systems [ANSI86]. Because of the success of X-Windows the ANSI committee is now inclined to switch over to X, and will probably standardize X-Windows version 11.

Most of the commercial window systems do not confine themselves to handling windows and the user operations related to them. In addition they often provide user interface library with low level raster device-oriented output facilities and some interaction techniques like menus and scrollbars. These tools are not well-suited to designing CAD applications because they do not provide graphics interactions like pick.

Only a few offer more powerful graphics systems like GKS for graphics I/O inside windows. In this case they usually are not fully integrated in the window system. A consistent integration of graphics systems and window managers is especially a problem, because systems like GKS were not meant to run within a multi-windowing environment. Therefore some modifications [LUXM88] are necessary to let graphics applications run in some windows of a window manager. The paper of Lux-Mülders et

al. [LUXM88] discusses several integration approaches. Because of market demands we will soon see more GKS implementations on top of window management systems.

2.4.3.3 User Interface Toolkits

The interaction technique libraries usually offered in addition to a window management system are called user interface toolbox or user interface toolkit. They are built on top of window management systems as a collection of subroutines providing a set of interaction techniques like menus, icons, scrollbars, forms, etc. This guarantees a uniform interaction style presented to the user.

The user interface toolkit is responsible for binding hardware devices to interaction units, giving prompt and feedback, and mapping user input to an event of a specific type. The events are pushed into an event queue while the application can read from this queue. Such tools can support the lexical level of user interface management.

Unfortunately, they are often restricted to some specific interaction techniques and lack graphical input mechanisms like pick or locator input because they are unrelated to output capabilities. Therefore some lower level input like button-down and button-up is passed directly to the application which is responsible for more complex graphics interactions.

Nevertheless, systems like the APPLE Macintosh Toolkit, MS Windows and the Presentation Manager from Microsoft, GEM from Digital Research, the X Toolboxes from DEC, HP, and OSF/Motif, SunView, SunNeWS, Open Look and others have become very popular.

2.4.3.4 User Interface Management Systems (UIMS)

A UIMS is a framework for user interface design, specification, and implementation similar to what software engineering environments are to software development. In contrast to toolkits they also support the syntactical level of dialogue sequencing and are not embedded into a programming language.

UIMS have been a research topic for many years [THOM83], [PFAF85], [OLSE87] and are now entering the market. One of the goals of User Interface Management Systems is to specify the user interface with formal means and to generate more or less automatically the real user interface from it.

A UIMS usually provides tools to define:

- lexical tokens and their binding to hardware devices;
- sequences of dialogue steps; and
- the routing of tokens to appropriate semantical processing places, i.e., the interface to the application.

From this specification, which can be coded in a UI description file, the runtime part of a user interface is generated. The user interface runtime part handles all the interaction with the user and is separated from the kernel application. Three types of communication between the user interface runtime component and the application are in current use [ENDE84], [HAYE85]:

(1) Internal control:
Internal control defines the user interface as a collection of I/O services activated by the application program. In this case the user interface is controlled by the application-oriented flow of programming code. The user communication is accomplished by calling corresponding functions.
(2) External control:
Here the application is divided into small packages each processing one dialogue unit. Initiated by predefined input sequences the user interface component calls application functions to give semantic feedback and to execute application-dependent program steps. External control ensures a clear separation of application-dependent functionality and I/O responsibilities. Controlling the context of all possible interactions, the user interface is able to handle any context switch performed by the user.
(3) Mixed control:
Here the user interface package and the application are realized by coroutines or parallel processes. Information is exchanged in both directions. Mixed control provides a very flexible interface between user interface and application because application-specific semantics can be inserted at lower levels of user interaction (e.g., semantic feedback at the lexical level). Yet a well-organized synchronization mechanism is necessary to control the communication flow between user interface and application.

The user interface should be developed by a user interface designer, a specialist in dialogue design, human factors, and psychology who has the knowledge to tailor the user interface to the personal skills of the potential computer-aided design group and to the specific requirements of working place and environment. Many CAD system designers are overtaxed in doing this because they do not have the knowledge of, and education in user interface design. Because the final CAD system should be separated into a user interface component and a real application component developed with different design tools, a personnel division of responsibility is recommended.

Nevertheless, it is nearly impossible for any dialogue designer to develop in a first attempt the optimal user interface. One of the reasons is that in the computer-aided design process a human being is involved whose behaviour is not as predictable as a software component, and whose capabilities are difficult to estimate and to deal with. A top-down specification of the user interface often fails because user skills were misunderstood or unpredictable. A study at the University of California [GING78] compared the properties of the user interface designer, the properties the designer predicted for the end user who will use his system and the characteristics of the real end user. The predicted properties were mostly closer to the designer's characteristics than to the user's.

Consequently, professional user interface design must be an iterative process. A first prototype is developed and tested with real users. The behaviour of the users, their problems and mistakes are inspected and further prototypes are developed. This method of prototyping, evaluation, and acceptance testing is usually more successful than a formal specification of all details at the user interface. A user interface specification should instead be of the form Shneiderman suggested in [SHNE82]:

"after 75 minutes of training 40 typical users should be able to accomplish 80 percent of the benchmark tasks in 35 minutes with fewer than 12 errors". Iterative design presumes a powerful UIMS for fast prototyping, the opportunity for user participation, and fixed acceptance criteria to decide whether a further iteration loop is necessary or not.

A UIMS often provides additional tools for help, error handling, undo, macro definition, and user profiles to customize system characteristics to individual preferences during user's login.

UIMS will spread abroad in the near future; at the moment systems like Apollo's Domain/Dialogue and Open Dialogue, PVI's Enter/Act, Tiger from Team Engineering, Interface Builder from ExperTelligence and Hypercard for the Macintosh, Flair from TRW, and BLOX of Rubel Software are available.

We saw that several types of tools exist to support the development of CAD user interfaces. Most of them concentrate on a specific task in user interface design. At present there is a great demand for integrated systems that incorporate concepts of graphics systems, window management systems, user interface toolkits, and UIMS. Graphics systems have their strong points in picture representation, structuring and support of modeling; window management systems are responsible for managing screen resources at a raster device-oriented level; user interface toolkits concentrate on interaction techniques; and UIMS deal with dialogue specification and dialogue management. It is not a disadvantage that these systems concentrate on what they can do best. But as a minimum there should exist well-defined interfaces between the components. Only a few attempts have been to incorporate dialogue management, graphics, and window management tasks within one integrated system: these are PRODIA [KRÖM88], SIEMCAD [BITT88] and THESEUS [HÜBN87a], [HÜBN87b]. The latter is now described in detail.

2.4.4 THESEUS: An Example of a User Interface Design Tool

The remaining part of Sect. 2.4 describes THESEUS as an example of a system for designing and controlling all the communication tasks between user and computer application. All I/O services are strictly separated from the application with the advantage that the user interface design is affected very little by the remaining CAD system. THESEUS supports:

- multi-windowing for parallel work in various contexts;
- full GKS facilities within windows;
- interaction techniques like menus, icons, object pick, dragging, etc., for user input;
- object-oriented output for hierarchical structuring of pictures;
- event-driven input processing to implement user-controlled dialogue-techniques; and
- mechanisms to specify dialogue flow and sequences.

To minimize the effort of designing and realizing a user interface, the THESEUS programming interface, i.e., the interface to the CAD application, is located at a high level of abstraction.

The following abstractions are provided by THESEUS:

- Physical abstraction frees programmers from device-dependencies. Moreover, this device-independent interface hides terminal size and resolution by maintaining windows for applications.
- Logical abstraction defines I/O-facilities in an application-oriented manner. Logical abstraction allows the use of graphics primitives as well as the creation of complex graphics objects based on predefined primitives. Moreover the application is freed of special interaction styles: e.g., the implementation of a menu selection is hidden from the application. Feedback for user events is standardized and executed by the user interface as long as semantic information stored in the application program is not needed.
- Finally, syntactical abstraction is provided by an approach called incremental dialogue specification. It enables the determination of dialogue sequences explicitly so that dialogue management and execution is separated from the application.

The THESEUS communication architecture between user interface component and CAD application programs is based on a mixed control model. Application-dependent output and control of the dialogue is supported by THESEUS service functions initiated by the application. In the case of input events, however, the corresponding semantical functions are called by THESEUS.

2.4.4.1 System Architecture

The transformation of device-dependent functions to application-oriented tasks and vice versa is realized via three layers. The THESEUS system architecture reflects this fact (Fig. 2.12):

- The basic I/O system converts device-specific functions and characteristics to a device-independent level. Therefore, other levels located above this layer do not have to deal with these special capabilities.
- The task of the second layer, the window manager, is the control and management of perhaps overlapping screen areas (e.g., open, close, resize of windows, etc.). It maps output primitives from window coordinates to a screen coordinate system and correlates physical input with events of some input classes. Moreover, it is responsible for output into visible or hidden parts of windows and for screen repair. It is possible to use existing window management systems, but then only parts of the required functionality can be satisfied.
- The third layer is the communication layer nearest to the application. It is divided into three components:
  - The Presentation Manager is responsible for the display of all information on the screen. It handles all output classes like GKS, object-oriented output, or alphanumeric output and provides functions for window control. This functionality is used by the application program to open windows and to draw and manipulate the contents of a window.
  - The Dialogue Manager maps user input to logical input sets. In the case of actions which have to be handled by the application, an application function is

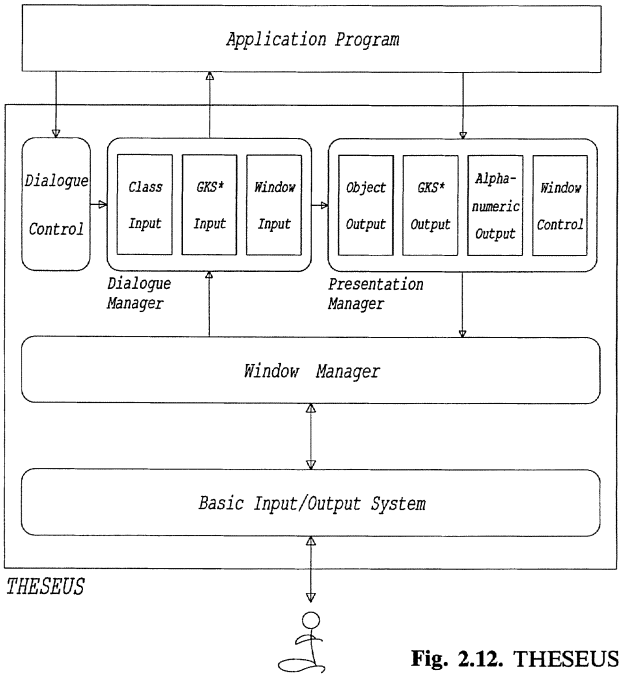


Fig. 2.12. THESEUS system architecture

called. Feedback in response to user input is done via the Presentation Manager. Window-related operations like move or scroll are signalled to the Presentation Manager which contains the data structures of the windows and its contents stored as logical information.

- The Dialogue Control provides capabilities to define dialogue units and dialogue sequences. This is done via the creation of input sets which, e.g., represent menus, icons, etc. Input sets can be created by application programs. In this way, allowable user actions are defined.
  Using the information stored within this component the Dialogue Manager is able to prompt, echo or reject user input, to transform it to application-oriented events and invoke capabilities of the Presentation Manager or the application program itself.

2.4.4.2 Window Management

The window management implemented in THESEUS hides from the CAD program problems normally related to multi-windowing, e.g., to redraw window contents in case of windows moved, resized, or scrolled by the user. THESEUS stores the window contents itself as logical data structures so that all necessary redraw operations can be executed locally without involving the application.

Nevertheless it is possible to inform the application program that such a window-related event had occurred. Three main tasks have to be handled by THESEUS:

- availability and management of a so-called virtual device surface (VDS); the information actually displayed may be only part of the VDS;
- realization of local interactions (move, resize, etc.); and
- correlation of input events – triggered by physical devices – with window contexts and possibly transmission to the CAD program connected with these windows.

THESEUS offers the following functionality for manipulation and control of windows:

- move a window on the screen;
- resize a window;
- scroll the visible part of the VDS; and
- scale a window.

At any time exactly one window visible on the screen is related to user input. This window is called the Listener Window; therefore another function,

- selection of the listener window,

is supplied.

Furthermore, other standardized user functions are available which have to be called by the application program but which are controlled by the user interface:

- open and close a window;
- demand for help information (HELP); and
- roll-back of transactions (UNDO).

The window frame contains interaction regions for these tasks. The window itself consists of a work area where information is displayed and graphics input (positioning, object picking) can be done.

Input and output functions supported by the THESEUS system are applied to different kinds of windows:

- GKS windows for programs written with the graphics standard;
- windows for graphics programs written in an object-oriented fashion; and
- text windows for display and manipulation of alphanumeric information.

The underlying concepts are identical for all kinds of windows: The application program directs its data output to a world coordinate system (WCS) without considering problems associated with terminal screen resolution or window size. Only parts of the WCS are usually visible in windows because of the limited size of the window or the required resolution of the screen. It is possible to shift this visible area – by means of user operations using scrollbars – within limits defined by the application program (pan area).

The application program can inquire about actual window information like size, position, and attributes for a flexible reaction to the new demands. The result is an optimal adaption of the output to the actual window conditions.

2.4.4.3 Output

The output supported by the THESEUS system can be divided into three classes:

- GKS output,
- object-oriented graphics output, and
- alphanumeric output.

Output based on the GKS standard (see Sect. 2.3.2) is mapped into a THESEUS window. Initially, GKS was not intended to support multi-windowing, but with minimal changes at the GKS application interface it is possible to let GKS run in a window.

- One change is that GKS is not allowed to take control over the whole physical I/O device because this is a contradiction to the concept of divided I/O resources found in window systems. Therefore, GKS is connected to virtual devices which are mapped by the window manager to real devices. As a result the GKS display surface is no longer fixed but changes dynamically in respect to the size of its window.
- Moreover, a second change is that user-induced modifications of the visual presentation are only possible by informing the application program responsible for performing updates. For conceptual and performance reasons the underlying window manager is responsible for executing window operations like moving, sizing, or scrolling locally without involving the application by asking GKS to modify the workstation transformation using the segment storage.

This solution keeps changes to the GKS standard minimal but tries to integrate multi-windowing capabilities to GKS in an efficient way. For further details see [LUXM88].

In addition, THESEUS provides an object-oriented approach for graphics output. It offers mechanisms for structuring graphics objects hierarchically. A basic set of graphics object classes can be used to build up larger graphics entities, the complex objects. Windows are then applied to present to the user sections of this "object world". Together with an inheritance principle for dynamical binding of attributes, application-dependent graphics structures are defined, while the visual presentation of those graphics structures is left to the THESEUS system. Therefore, the main application task concerning output is to correlate application-oriented objects with graphics objects and to position these objects into a two-dimensional world coordinate system which can be defined freely by the application.

During its lifetime a graphics object is represented in two ways:

- an entry into the graphics data structure (GDS); and
- a physical (visual) representation on the output device.

The representation of the object at the terminal screen is controlled by data items in the GDS connected to this object. The application can only access the GDS, whereas THESEUS manages the representation autonomously. Both representations have to correspond to each other at all times.

Complex objects are composed of basic objects or other complex objects (hierarchical structuring of objects). This concept allows the definition of any application-

specific structure by means of simple building blocks (the basic objects). The same operations applied to basic objects are also valid for these complex structures.

Once created, these complex objects can be viewed as new elementary units. A complex object therefore consists of basic objects and/or complex objects and can be seen from a logical point of view as a tree with basic objects as leaves.

For the application it is possible to create, manipulate, and delete instances of the different object types.

Visual characteristics of objects like geometry, size, line type, or fill interior style are divided into three groups.

- The first group of characteristics describes the object types, e.g., the shape and orientation of a diamond. These type characteristics are fixed and determine the layout of the object type.
- Other characteristics like size or radius describe how a single occurrence of a specific object is to be drawn on the screen. These attributes are defined before drawing the object.
- The third group of attributes can be changed dynamically, e.g., line type, colour, or visibility. Such dynamic changes to graphics attributes may either be deferred or will be displayed on the screen immediately.

Graphics attributes of complex objects are bound to child objects by an inheritance mechanism. Each child object is annotated as to whether it inherits the attributes of the parent object or not. In the first case, the attribute values of the parent object are used for output, otherwise the child attributes are valid. The mechanism is a recursive one, i.e., it is applied also to child objects which are complex objects.

The alphanumeric output interface handles the output of alphanumeric text consisting of characters of fixed size which are positioned into a matrix of rows and columns. This serves as a basis for intelligent alphanumeric editors usable in a window environment. Text is stored in a buffer and handled autonomously by THESEUS analogous to graphics objects.

2.4.4.4 Input

The THESEUS system architecture allows the implementation of user-driven man-machine interfaces where the user takes the initiative and the system reacts to user actions. Physical user input with input devices like keystrokes or movements of the mouse are captured, collected, and related by THESEUS to logical input events like object dragging. Feedback is given autonomously. In case of appearance of a logical user event THESEUS can call an application function for semantical processing.

In conventional window management systems, physical user input (events) is signalled directly to the application to be processed there [HOPG85]. The THESEUS system frees the application from waiting for input, testing the authorization, and branching dependent on input types. The mapping of physical user actions to logical events related to application functions is done completely internally by the THESEUS system which than triggers the corresponding application functions. The relation user action → application function is not a static one, but can be modified dynamically by the application. The relation is defined within data structures controlled by

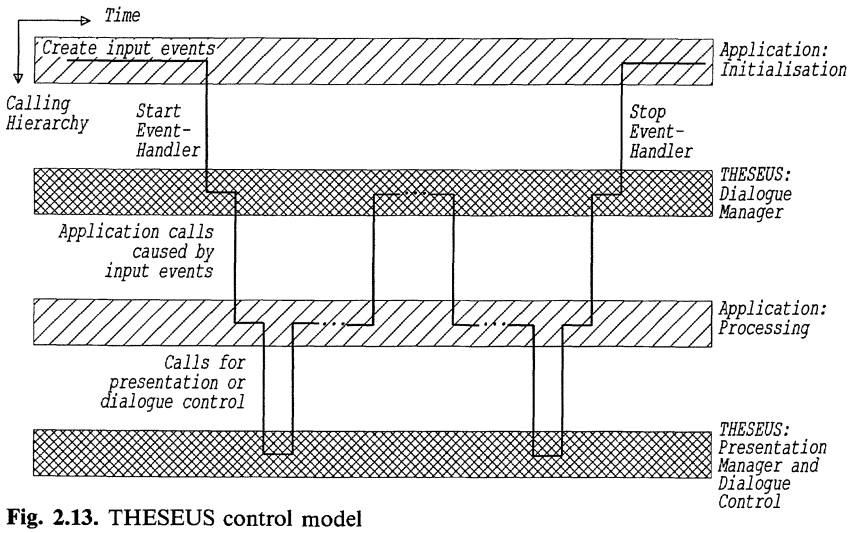


Fig. 2.13. THESEUS control model

THESEUS. Moreover, the application is allowed to control the mapping process by using functions to access the input data structures of the THESEUS system.

The flexibility of information exchange at the interface THESEUS-application requires a well-organized control architecture for communication.

The THESEUS model of alternating control is shown in Fig. 2.13.

At first, THESEUS is instructed by the application in which way it has to react to user actions. Normally this task is part of the main program. After having initialized first input events, the event handler as part of the Dialogue Manager is started. It expects and collects user input, checks its permission, performs lexical, syntactical and in some cases semantic feedback and activates the application if necessary. These applications use THESEUS facilities for output and window control as part of their semantic feedback and dialogue control components for specifying the next dialogue step. After the application function has finished, the event handler is enabled to call the next application function corresponding to user actions that have occurred.

To relate user actions to an application function several steps are executed within the THESEUS system. At first one or a sequence of input events is related to an input class. The relation is governed by fixed rules defined by the input data structures of the THESEUS system. Typical physical input events (e.g., with the mouse) are pressing or releasing one mouse button or moving the mouse. Examples of input classes are menu selection, icon selection, object identification, object dragging, positioning within predefined areas, or keyboard input.

Logical input events are grouped into input sets. Each input set consists of elements of one specific class, e.g., the elements of an input set of class "Menu Selection" represent the menu items themselves. The grouping into sets is done during the initialization phase of the application program according to an application-specific task. Input sets are used to control the dialogue sequences. Input sets can be enabled or disabled to allow or forbid the interaction units within a set. Moreover, each ele-

ment can be enabled or disabled. The input sets and the elements can be modified dynamically. THESEUS provides functions to create and delete input sets, to add and remove these sets to/from the event handler, to add and remove elements to/from sets, to disable or enable sets and elements, and to set or inquire about input sets or element attributes.

Each element of an input set is related to an application function. After having related a physical input event to such an input element, the application function associated with this event is called by THESEUS. Several elements can be connected with the same application function. The application function is not necessarily limited to application-dependent processing. It is also possible to modify THESEUS-specific data structures for the control of subsequent input events. Using this mechanism dialogues can be specified incrementally.

In contrast to most UIMS, the dialogue steps and their order are not specified in a preliminary design phase because many graphical user interactions like picking are related to dynamic elements which cannot be totally predefined. Therefore, the

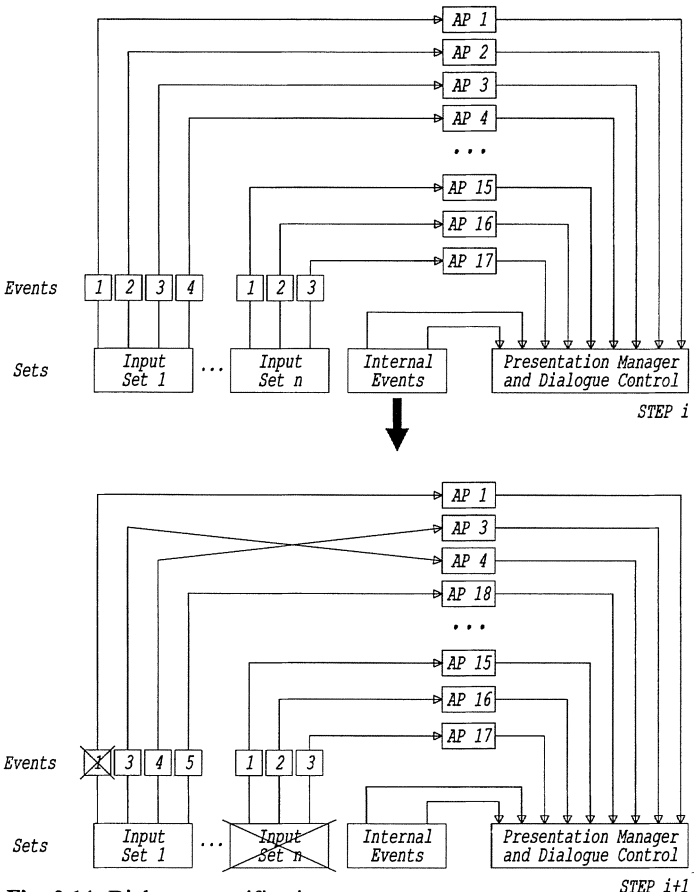


Fig. 2.14. Dialogue specification

dialogue syntax is dynamically defined and modified at run-time. To reduce complexity only the changes compared with the dialogue step before have to be described.

Figure 2.14 shows an example of two successive dialogue steps. The logical events like dragging an object or selecting a menu item are grouped into input sets. Internal events like window operations are processed locally. All other events are connected to application functions. After an event has occurred and an application function is called, a transition to step i+1 is performed. During this transition event 1 of set 1 is disabled so that input related to this event is rejected. Event 2 is removed, the application functions triggered by event 3 and 4 are switched, event 5 is added and the whole input set n is disabled.

THESEUS offers the functionality required to specify the changes during a transition in an incremental way. That method of dialogue specification obtains a high degree of flexibility in describing dynamic graphical user interfaces and reduces complexity problems.

THESEUS is now used in several graphical application areas. For further details see [HÜBN87a], [HÜBN87b]. THESEUS is a research product developed at the Computer Graphics Center, Zentrum für Graphische Datenverarbeitung e.V. (ZGDV) in Darmstadt, carried out within the UNIBASE project, partially sponsored by the Federal Ministry for Research and Technology (BMFT), grant number ITS 8308, with the project contractors ACTIS, ADV/ORGA, IABG, mbp, FZI, GMD, TU Berlin and ZGDV.

2.5 Application Interfaces to Engineering Databases

2.5.1 Introduction

The quality and efficiency of the design process, especially in mechanical engineering and VLSI design, has been improved by the development of sophisticated tools and systems. While developing tools will continue to be a point of research, a new research direction concerns the aspect of the integration of tools by a central database and by standardized interfaces. Today most of the CAD tools work in isolation from each other, using individual input and output files. Each tool requires its special data format; conversion programs have to be written to allow communication between different tools. Aside from this drawback, the file system approach lacks a lot of capabilities which existing database systems for commercial applications offer. Nevertheless, database management systems (DBMS) for administrative applications are not immediately suited for engineering design areas. There exist two principal ways to satisfy the special requirements for CAD database systems. The first solution is to extend existing DBMS with an additional layer. The other way is to develop completely new database systems with all features required for engineering applications.

Engineers in CAD/CAM and graphics-oriented applications on one side, and database theory scientists on the other side have to work together to integrate tools and DBMS into commercial products. The acceptance of such systems by users mostly depends on the database application interface and support of the system in modeling and manipulating the applications data (see also [ENLO 90]).

The intent of this section is to present such engineering database interfaces and to show some examples of how to work with them. We first introduce a new data model called PRODAT, which has been developed for the system engineering environment PROSYT[1].

We use this example to show the main features of object-oriented data models and working with database application interfaces. The second section gives an overview and some examples of the difficult process of defining data structures and data item types for specific applications. The third section shows some mechanisms for handling versions of objects in databases. Finally, we will analyze the procedure of generating data and entering them in the database.

The operations and resources for use will be shown in examples and some deficiencies from an applications point of view will be discussed.

2.5.2 Data Modeling in PRODAT

PRODAT [PROD88] is a database system which supports tools in a system engineering environment (see also [DITT86]). In particular, PRODAT provides basic mechanisms for modeling complex objects, configurations, and versions. These modeling concepts include fundamental operations for:

- construction of complex structured objects from predefined parts or subobjects;
- recognition of the structure of a certain complex object; and
- selection of special parts of complex objects.

Instead of a conventional query language, PRODAT offers a procedural interface to tools, and a graphical interface to human users. Only schema definitions are still done by alphanumerical means.

2.5.2.1 Complex Objects

Complex objects (see also [LORI83]) consist of simple objects connected by standard relationships. Simple objects are described by attributes and may contain text, object code, pixel data, etc., which are called the *contents* of the object. The contents are a variable-length byte string which is not interpreted by PRODAT. *Attributes* are similar to those of the relational model, where each attribute has a domain and is unstructured. Figure 2.15 explains the two parts "contents" and "attributes". The example object contains a program source code and the attributes describe the generation date of the program, the author, the programming language and an internal code number.

The structure of complex objects is modeled through *relationships*, which are defined in the SUCCESSORS part of the schema definition (Fig. 2.16). It is possible either to give an expression that determines the set of admissible SUCCESSORS or to use one of various standard relationships (e.g., 'has_component', 'has_alternative'). Relationships group simple objects in a hierarchical manner; there is no limit,

[1] PROSYT R&D project was sponsored by the German government (BMFT) under reference number ITS-8306A7.

Attribute	Domain
11.11.87	DATE
heiner	NAME
pascal	CHAR 15
160559	INTEGER
Contents	
program gen_pic; *begin* ... *end.*	

Fig. 2.15. Attributes and contents of an object

```
TYPE logical_object
      STRUCT
      name CHAR (20)                          /* attribute definition */
      date DATE                               /* attribute definition */
      CONTENTS                                /* contents definition */
      SUCCESSORS                              /* subobject definition */
        1 text_object OR 1..2 picture_object  /* non-exclusive or! */
;
```

Fig. 2.16. Type definition of an object

however, to the number of predecessors of an object. Completion of complex objects is a necessary condition for closing a phase of the computer-aided design process. The structuring of objects might be a long process during which objects are not guaranteed to stay consistent in each state of development. But tools and applications need some predefinitions and restrictions when using objects. It should be possible to fix certain conditions for the structure of complex objects in the schema definition of objects. This is the reason why PRODAT supports definitions about the completeness of complex objects. A complex object is in the state *complete* if the set of its current subobjects matches the expression in the SUCCESSORS part of the object's definition both in type and cardinality. The completeness status of an object is evaluated by the system each time a subobject is moved or otherwise manipulated.

Figure 2.16 shows a schema definition. It determines that for each object of type "logical object" one subobject of type "text object", one or two subobjects of type "picture object", or a combination of both cases must be present for the object to be complete. Incidentally, this defines the 'has_alternative' standard relationship. Figure 2.17 gives an example of a complex object with subobjects. For certain kinds of combinations of types in the subobject definition part, PRODAT defines standard relationships which are typical structures of objects in systems engineering environments. The precise definition of these standard relationships is not necessary in the context of this paper. The example in Fig. 2.17, however, provides some basic ideas.

The object "Flip-Flop" has two subobjects, "phys. structure" and "logical descript.", which are connected to it by the relationship 'has_representations' (R). In this case "Flip-Flop" is complete if the so-called representations both exist and if each of them is of the appropriate type. Object "logical descript." has two subobjects "text" and "picture", which are connected to "logical descript." by a relationship

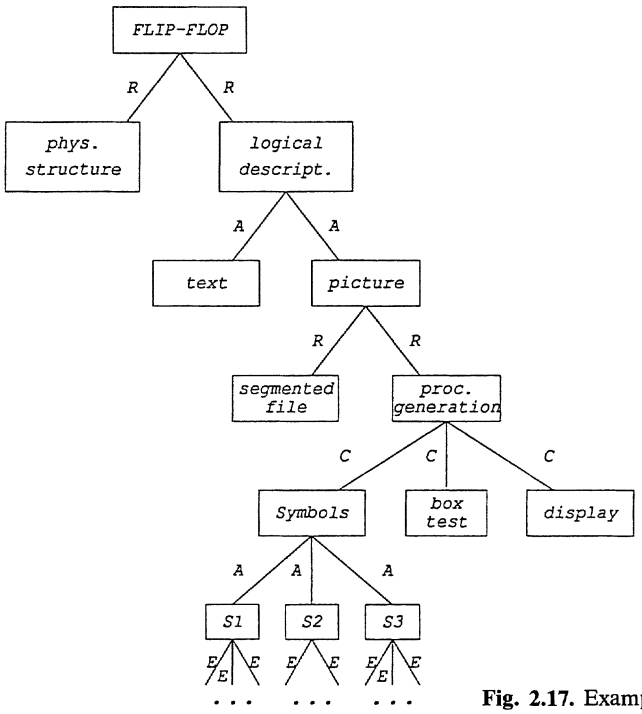


Fig. 2.17. Example of a PRODAT object

'has_alternative' (A). The 'has_alternative' relationship defines that the superobject "logical descript." is complete if any subobject of a type specified exists. Object "proc. generation" has three subobjects, which are connected to it by the 'has_component' relationship (C). The object "proc. generation" is complete if exactly the specified number of subobjects for each defined type exists.

2.5.3 Database Schema Design

Before user data may be inserted or manipulated with manipulation operations of a data manipulation language (DML), a database schema has to be defined for the database by an authorized person (e.g., the project administrator). This section gives an overview and some examples of the difficult process of defining data structures and data item types for specific applications. We will discuss in this section only the development of a database schema and not the general modeling aspects of CAD. More information on developing a schema for CAD systems and for CAD applications is given in Sect. 3.3.1.

Database schema modeling features like data definition languages (DDL) and support tools help users in modeling their miniworld semantics. A future trend in this direction is the use of graphical tools or interfaces for definition of a database schema. Systems support the aspect that people think in pictures and therefore want to use diagrams or similar techniques.

We first determine and formulate some general requirements for schema design in technical applications.

Although the process of modeling design information with logical database structures may be intuitive and individual, we can find many common advances to map the applications meta data (logical data and structures for holding user data). This is indeed the reason why database designers can find data models. We can not give a recipe for schema definition for engineers, but we can describe the demands and some advances in engineering design. From this starting point we can derive some proposals for data description and give some suggestions in working with a data model, which is described in Sect. 2.5.2. Working with DML operations on user data is described later in a further section.

2.5.3.1 Principles of Using Data Model Features

In each application, database schema definition is an examination of the features of the data model, so it is the first test of the formal framework.

Data declarations give an answer to a basic question, namely the question 'is it possible to map the elements of a particular application to the components of a particular model?'. If it is, the model would be useful and the corresponding database system can find acceptance. Schema definition is the most important part of the integration of tools/applications and database system, because the schema is the kernel of the interface between the two components.

Data modeling is a method or formal description for structuring user data by defining:

- object types;
- object structures;
- relationships; and
- consistency rules.

In the following we concentrate on determining object or entity types, object structures, and relationships as main components of the data structuring process. The engineer has to observe some principles. If he describes his design world by the schema, he has to keep in mind the definition level. There is a strong distinction between the *data type definition level* and the *instance level*, where insertion and manipulation of object instances by entering and changing of values are done. In discussing object semantics, we often forget this distinction, which is important when we discuss types, objects, configurations, versions, and static and dynamic links in the structure part of structured objects (see also Fig. 2.18).

We can compare these two parts with computer programming languages: the data definition is the declaration part and the second part are the programming commands. But in engineering applications the analogy is not strong, because the schema definition should not be as static as the type declaration in programming languages. Design is an *evolutionary process*, object types and structures evolve during the design phases. In the course of declaring types and structures we have to notice that they must be extendable.

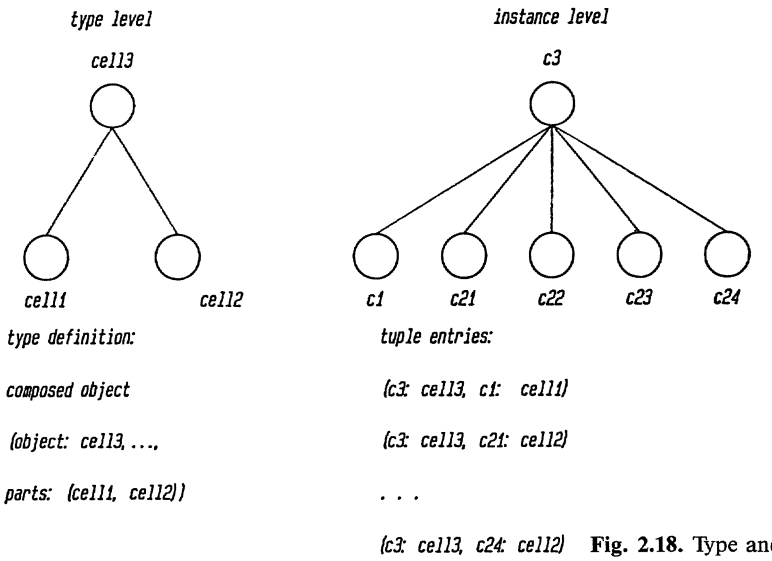


Fig. 2.18. Type and instance levels

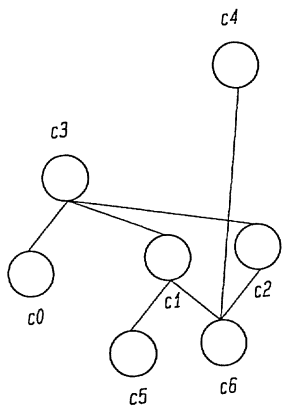


Fig. 2.19. Shared subobjects

By modeling fixed *relationships* between object types like the 'is_part_of' link in the composition of a structured object, we have to take care about the *resulting framework*: it must be useful later for designers! (See Fig. 2.19.)

The careful use of link types leads us to the problem of *redundancy*. Sometimes structuring of data is manageable, but there is a lot of redundancy in the information because the features of data models are not efficiently used (think of the algebra in the relational model). By structuring the data one can already avoid some redundancy, if a global overview of the overall schema and the whole set of design objects exists.

Last but not least, the user's intention is to have not only a good logical representation of the data, but also efficient access. *Performance* is, of course, an important factor for acceptance of the database system. So, we should not forget this point in the schema declaration either. For example, if the model supports the destinction between simple and compound objects (complex structure), we should only declare a simple object type for unstructured simple data values. Otherwise, the overhead on

the instance level, e.g., for retrieval is too high. The conclusion is to be careful in using complex features of the model for simple solutions, because every service has a price.

2.5.3.2 Semantic and Logical Organization

Basic data structures which are given by the data model (e.g., tables and columns in the relational model) are the logical organization of data. To define a sub-schema (specific schema for a certain application) the designer has to group and arrange his types of data in a way which allows the application or the user manageable insertion and manipulation of data. Furthermore, the designer should take advantage of the whole complexity of the particular data model. Only like this is an optimal use of the database and its DML operations possible. Let us first consider the kind of applications and data which use an engineering database with features like the ones described in this book.

The global task is to map the engineering design semantics onto features of the data model. What we mean by semantics are the contents and the meaning of the objects, and the interrelationships between them, which together describe the design area.

Engineering design description incurs the very difficult task of fixing as little semantics as possible, but no less than necessary. If less are fixed, the later user has more freedom in writing his own application programs. If a great deal of semantics are fixed by the model, the user has a well-defined environment and it is guaranteed that special tools will work in the right way.

An example of this principle is the information about cell ports in a VLSI design. In the schema description one can fix the types of ports, or this information can be variable and not fixed. Later in the design phase, the application has more possibilities if the type is not fixed (see example in Fig. 2.20). But it is now the task of applications to fix and interpret the semantics of a particular port. In this case, the engineering database system can not support control of cell ports.

Therefore, in an engineering design environment, where user-written DML programs are not the primary application, but tools working on data are, the administrator should fix as much semantics as possible. Of course there are some counter examples, but in an integrated system, which should support and control very different tools, we take it as a rule to fix semantics strongly.

This task enforces an information analysis of the existing application-specific tool environment. Which information flows between which tools, i.e., what kind of data are the input and what kind of data are the output of a certain tool or application?

After getting this information and structuring the data thus determined, one can begin with the data description in the database. The administrator has to model the

```
less fixing:     relation cell (name: string, . . ., port: string)
good fixing:     relation cell (name: string, . . ., port: (in, out, inout))
strong fixing:   relation cell (name: string, . . ., inport: in, outport: out)
```

Fig. 2.20. Semantics fixing

representational and structural details of a design and the design process. We first describe this process and the data produced.

Engineers work with interactive graphics methods and tools at a workstation. After changing the design into a certain state, they start with some tools to check the design, or simulate and test it. Therefore, the whole design exists in different representations or views on different abstraction levels, with additional interrelationships across the representations. Moreover, it must be possible to partition the design among subsystem designers. The database must provide structures for organizing the design description within and across representations, design versions and interface specifications.

2.5.3.3 Objects and Interdependencies

Here is something more about the objects, which must be described in the database. For the term 'object' we use in this section identically also the terms 'entity' or 'collection of data items'.

There are many different characteristics that together determine the object's semantics, which can be divided into three classes:

- object representations,
- object interdependencies, and
- library or object overlapping information.

In the following subchapters we consider these three main characteristics in more detail. The above mentioned points of design objects are very important for the description of the designer's miniworld. From each of the following points, we can derive some requirements on the data structures and relationships needed for the design description in the database.

A) Object Representations

Representations in the context of database objects mean the description of an object at different abstract levels. A complex object exists in more than one representation (see also [KATZ85]). Some kinds of representations are not application-dependent and we can find some common points and examples. Objects can be described by:

- internal and external information;
- interface information;
- the behaviour;
- the documentation; and
- graphical information.

The schema definition must include representations of *internal and external information* belonging to an object. At certain levels, objects may be handled as black boxes with a special input and output. If the design wants, for a cell component, differentiation between, e.g., the name, colour, and dimensions as external, and the number of primitives, geometry of primitives, etc., as internal information, he has to define two representations of the object. The distinction is significant: if a user wants

to place this object in a certain drawing environment, then he needs only the external information.

Another example is a program module: its name, and input and output parameters may be defined as an external part, while the internally defined subroutines or other details are hidden, externally inaccessible information.

Some objects have interfaces to other objects in the database, e.g., if an object is used in another one. Therefore, the database has to manage *interface information* of an object as well. Consider once again the program module example. Interface information may be the program's function, the programming language, and a reference to its external information.

Design objects need a *behavioral description*, if they are functional parts of a system configuration. There are many different forms for the behavioral description of design objects. It may be state diagrams and tables, informal text, logical truth tables or other mathematical descriptions, etc. The formal descriptions are mostly at a high logical level, and so are the features of a data model. So, we can find a mapping between the two forms.

Objects are often very complex and not self-descriptive. Therefore, they need a *documentation*, mostly in text form. Textual information like documentation should be placed into long fields of the data base, so it need not be partitioned or cut in an impractical way.

Engineering design often produces objects which have a *graphical representation.* Graphical information like geometric parameters of an object must be structured and kept with data model features. Here there is overlap with the first kind of representation: some graphical information can also be partitioned into internal and external parts.

B) Object Interdependencies

As an example for an object interdependency, suppose that if object A is modified, this event induces an action (e.g., that object B must be generated anew).

Such complex consistency constraints can not be enforced by the system in every case. Sometimes consistency is best maintained semi-automatically by sending messages to the user, or by prohibiting some operations on such data. Another possibility is a strict control over the execution of tools in a well-defined order (e.g., to run a logic test before the physical layout of a circuit is done). However, there are some general interdependencies the engineering database system can support like:

- the composition;
- the hierarchy;
- connectivity information; and
- the evolution.

The designer has to give some semantics of the *composition* of complex objects. He has to generate structures that express how complex objects are composed from primitives (basic objects and structures). We can describe a compound object as a set of independently manipulated primitives or complex objects. Between these objects exists a characteristic relationship (like, e.g., a hierarchy or a network). An example of a compound complex object is a VLSI cell, shown in a graphical representation in Fig. 2.21.

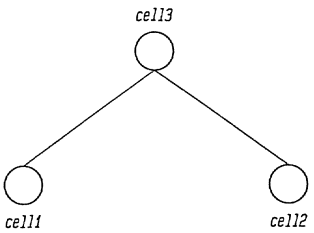


Fig. 2.21. Compound object in VLSI

Hierarchical construction of the design is a result of the natural top-down partitioning in the design process. Thereby the *hierarchy* is a particular relationship between objects and the construction a particular method. In many systems in our daily lives, we can find hierarchically arranged subsystems as compound parts. Some of these are the 'component__of', 'child__parent', and 'is__a' relationships (partition/composition, specialization/generalization).

Some objects may be connected in, e.g., a netlist structure. In VLSI design there is information about ports and junctions between parts of a netlist.

To model the evolution of design objects and design interdependencies between them, the user should classify his objects as versions and use systems version management. A version mechanism is described in Sect. 2.5.4. From a more abstract point of view, this issue will be revisited in Sect. 3.3.5.1.

C) Library and Object Overlapping Information

Many design objects belong to projects. There is global information for each project to manage the different design parts. Library objects or relations of objects to libraries are also important for design environments. We have to consider object-overlapping information like:

- project dependency;
- simulation information;
- technology information; and
- standard library.

An important information for partitioning a collection of designs is the dependency between *projects* and objects. For each project the administrator has to define a new database for organizational reasons. Between the different databases it should be possible to interchange project-overlapping information or technical details for further design constructions.

Simulation tools manipulate objects and include further ones or complementary data. To simulate, e.g., the electrical characteristics of a VLSI cell represented as a complex object in the database, the designer has first to map the features and semantics of some hardware description language into the data structures of the basis data model.

A design object in a certain representation may need some *technology* parameters to include the object in a final system configuration. Technology information is mostly not private, individual data; instead, it is, e.g., dependent on a workstation or dependent on a certain material. So the designer should define this information in a

general library, accessable to all users who work with the same design in the related project. To use the technology data, one has to reference the related objects or make entries in a table or entity describing technology.

Design data and structured objects are often reused in other designs. After completion of object preparation, some objects will be released from private use to the global environment. Released objects will be compared with or included in the current development. Another point is inclusion of standard solutions, e.g., VLSI cell design with standard cells is a common technique. In these cases we need *standard libraries* in a global database area.

2.5.3.4 Example

After informal description of problems in the definition of database schema for engineering applications, we introduce a specific example of data classification and schema definition. Step by step, a schema for the example given in Sect. 2.5.2.1 will be defined by using the data model features of PRODAT (see Sect. 2.5.2). On the basis of object types and relationship types described in this section, we show the creation and manipulation of some objects in the following Sections.

The design of structured object 'Flip-Flop' (FF) is done topdown; that is we first define the object type for FF, then the types for its successors, and so on. But the types exist independently; there are no subtypes or other hierarchical concepts used in the schema definition.

Type Definitions of the Structured Object FLIP-FLOP

```
/*** Predefined Types for Attributes ***/
TYPE NAME                    STRUCT[ name CHAR (20) ];
TYPE ADDRESS                 STRUCT[
                               name NAME,
                               prename NAME,
                               street CHAR(40),
                               city CHAR(40)
                             ];
/*** Object Type Definitions ***/
TYPE FLIP__FLOP__OBJ         STRUCT[
                               shortname NAME,
                               author ADDRESS,
                               generation DATE,
                               CONTENTS,
                               SUCCESSORS
                               1 PHYS__OBJ AND 1 LOGICAL__OBJ
                             ];
TYPE PHYS__OBJ               STRUCT[
                               level INTEGER,
                               generation DATE,
                               simulated BOOLEAN,
                               CONTENTS
                             ];
```

```
TYPE LOGICAL__OBJ          STRUCT[
                             shortname NAME,
                             author ADDRESS,
                             generation DATE,
                             kind CHAR [10],
                             CONTENTS,
                             SUCCESSORS
                             1 TEXT__OBJ OR 1..2 PICTURE__OBJ
                           ];

TYPE TEXT__OBJ             STRUCT[
                             author ADDRESS,
                             generation DATE,
                             system NAME,
                             editor NAME,
                             formatter NAME,
                             CONTENTS
                           ];

TYPE PICTURE__OBJ          STRUCT[
                             author ADDRESS,
                             generation DATE,
                             graphics NAME,
                             bitmap BOOLEAN,
                             device NAME,
                             CONTENTS,
                             SUCCESSORS
                             1 SEGMENT AND 1 PROGRAM
                           ];

TYPE SEGMENT               STRUCT[
                             metafile CHAR [10],
                             generation DATE,
                             CONTENTS
                           ];

TYPE PROGRAM               STRUCT[
                             language NAME,
                             author ADDRESS,
                             generation DATE,
                             execute BOOLEAN,
                             CONTENTS
                             SUCCESSORS
                             1 INCLUDE AND 2 MODULE
                           ];
```

```
TYPE INCLUDE                STRUCT[
                              language NAME,
                              generation DATE,
                              CONTENTS,
                              SUCCESSORS,
                              1..10 SYMBOL__OBJ OR 1..100
                              PRIMITIVE
                            ];

TYPE SYMBOL__OBJ            STRUCT[
                              longname NAME,
                              language NAME,
                              generation DATE,
                              CONTENTS,
                              SUCCESSORS
                              PRIMITIVE
                            ];

TYPE PRIMITIVE              STRUCT]
                              longname NAME,
                              language NAME,
                              generation DATE,
                              CONTENTS
                            ];
```

The expressions in the SUCCESSORS clauses define the following 'completeness rules' for objects of the named types:

A AND B: There must be objects of both types, type 'A' and type 'B'.
A OR B: There must be either objects of type 'A' or objects of type 'B', or objects of both types (non-exclusive).
A XOR B: There must be either objects of type 'A' or objects of type 'B' (exclusive).

If 'k' is a cardinality, we can write the following abbreviations for standard relationships. The database system will interpret them as described above. If there is no cardinality given, we can generate as many objects of this type as we want (see type SYMBOL__OBJ).

For k_1A_1 AND...AND k_nA_n
we write K (k_1A_1, ..., k_nA_n)
to define the 'has__component' relationship.

For $1...k_1A_1$ OR...OR $1...k_nA_n$
we write A (k_1A_1, ..., k_nA_n)
to define the 'has__alternative' relationship.

For 1 A_1 AND...AND 1 A_n
we write R (A_1, ..., A_n)
to define the 'has__representation' relationship.

For A_1 OR...OR A_n
we write E (A_1, ..., A_n)
to define the 'has_element' relationship, which is the most common one.

2.5.4 Version Management

Generally speaking, all technical design processes produce versions of the objects to be designed. It is impossible to give a formal specification of a version; versions may represent provisional results, erroneous developments, improved designs, updates, or final results. In particular, versions reflect special interrelations resulting from the design process. An idea of version generation is illustrated in Fig. 2.22.

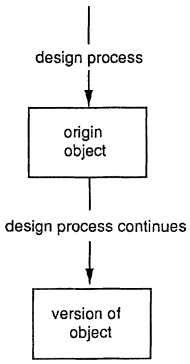


Fig. 2.22. Version generation process

The lack of a formal specification makes it impossible to determine the semantics of versions by a database system. Semantics of versions depend on the design process. Therefore the support of version management by an engineering database system is restricted to some basic versioning mechanisms. A model of version interrelations using these mechanisms must include the design process.

2.5.4.1 Version Generation in the Course of a Design Process

Each design process can be subdivided into different design phases. The sum of these design phases and their interdependencies sets up a so-called design life cycle. Remember our example of a VLSI design. In the case of a VLSI design, we can find at least four design phases:

- requirement engineering;
- conceptual design;
- logical design; and
- layout or physical design.

The first step of a VLSI design is to establish a list of design goals. The conceptual design results in a decomposition of the circuit to be designed into subsystems, and a specification of the behaviour and interactions of these subsystems. This way, the functional dimension of the circuit is given, e.g., by a register transfer description or block diagrams. The logic design leads to a logic description of the subsystems, usual-

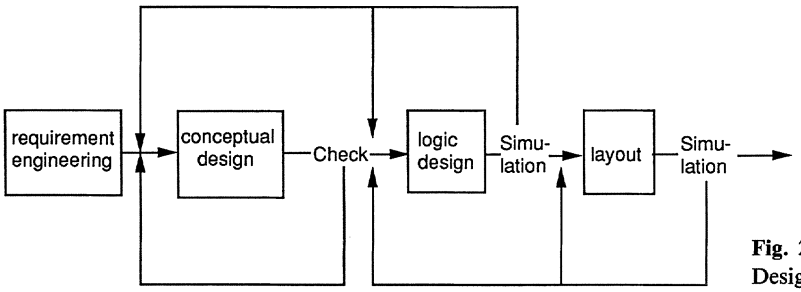


Fig. 2.23. Design life cycle

ly presented as logic gate schematics. The last step in the course of VLSI design is to work out the layout. The layout describes precisely how to place and interconnect the design parts. Each design phase leads to a complete description of the circuit, but reflects a different level of abstraction – so-called representations of an object.

Of course a design process is not as linear as implied by the above explanation. As illustrated in Fig. 2.23, a lot of iterations may occur during a design process.

Each design phase is followed by a check for correctness. In a VLSI design this can often be done automatically, e.g., the layout is checked by design rule checkers, and the logic design by logic simulators, switch level simulators, or circuit simulators.

It stands to reason that detection of errors leads to iterations, i.e., the corresponding design phase has to be passed through again. Iterations are not only caused by error detection, but also by an optimization of a certain design step. The design process consists of an inner loop of synthesis and analysis parts.

Even if the results of a certain design phase are correct and optimized, redesign may be necessary. For example, the logic design does not take into account geometrical aspects. When designing the layout, there are geometrical restrictions we have to take into consideration. If it is not possible to design an adequate layout of our logic design because of such restrictions, we have to look for alternative solutions. From these considerations results another loop of the design process. (See Sect. 3.1 for more details on CAD design processes.)

As shown above, many things can force iteration cycles, each one leading to a version of an object. This is not only a characteristic of VLSI design, it is typical of all technical design processes. We can find similar design phases more or less in all technical design processes. Looking carefully at the design process we can now distinguish between two classes of version – variants and revisions. Variants of an object have similar functionality but may have different performance characteristics. For example, two layouts of a circuit may be designed, one consuming less area and another one consuming less power. Variants reflect design decisions while revisions of an object reflect improvements. The generation of a revision is always done based upon its temporary predecessor, while variants may be generated at any time.

Furthermore, we have to distinguish between released and in-progress versions. Once a version is released, changes have to be prohibited. Bear in mind that variants as well as revisions may be released. Released versions set up a kind of working base, i.e., not only the designer of the version itself but also other designers have access to released version and may use them for ongoing work.

Let us consider VLSI design again. Even if the layout of our circuit is not optimized, a prototype based on it may be produced. In such a case we have to guarantee

that the version of this layout will neither be changed nor deleted. In contrast with released versions, in-progress versions are the ones still under development. Hence a designer should be able to change, improve, or delete them.

To summarize, we should bear in mind that each design object is described at different levels of abstraction (representations). Usually various versions of the designed objects exist for each of these representations (see [NEUM83]). These are two classes of versions to be managed during the design process:

- revisions; and
- variants.

Of course, versions may have other meanings, but in our context we shall focus on the classes given above. Other semantics of versions are not yet supported by engineering database systems.

2.5.4.2 Modeling of Version Interrelations

Version interrelations do not only exist within a certain representation, but also across different representations. Modeling of version interrelations across different representations is a very difficult task. Therefore we must focus on the modeling of version scheme interrelations within a certain representation.

To manage version interrelations, we need a modeling scheme. A modeling scheme depends on what interrelations a designer is interested in. Thus we have to work out different schemes depending on the interrelations to be managed.

Often version management (see also [DADA84], [DITT85], [KATZ84], [TICH85]) is needed to manage history information only. In this case a linear modeling scheme is sufficient, even if it implies sequential version generation as illustrated by the following example (Fig. 2.24).

In our example given above, we assume that V_0 is the generic version of the object to be designed and the numbering of versions reflects their creation times. Then our scheme implies that version V_1 is a revision of version V_0, version V_2 a revision of version V_1 etc., i.e., each version is based on its temporal predecessor. Although the sequence given above is correct from a time-dependent view, it need not reflect the real design interrelations. For example, version V_2 may be a redesign of version V_0 because version V_1 turned out to be a dead end. Hence, a linear modeling scheme does not allow distinction between variants and revisions.

To distinguish between variants and revisions, a tree-like modeling scheme is useful (see [RIED86]). (Figure 2.25 gives an example of a tree-like scheme of modeling variants and revisions.)

Again we assume that V_0 is the generic version of the object to be designed. Versions V_1, V_2, and V_4 are variants, i.e., they are alternative redesigns of version V_0.

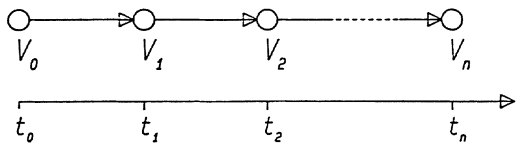


Fig. 2.24. Sequential version generation

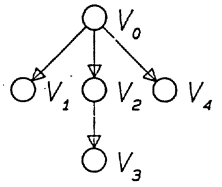


Fig. 2.25. Tree-like version generation

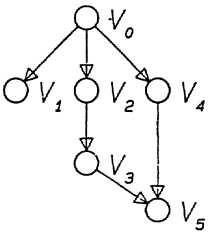


Fig. 2.26. Directed acyclic graph version generation

Even if we assume that version V_1 is a dead end, in the course of a design process it has to be interpreted as a variant. Version V_3 is a revision of version V_2. Generally speaking, we can interpret a tree-like modeling scheme like this: a certain path in the tree reflects revisions of an object, while different paths reflect alternative solutions. If the numbering of versions implies their creation times, it is possible to manage history information by a tree-like scheme as well.

Tree-like schemes are useful for distinguishing between variants and revisions. But how does one manage merging of versions by a tree-like scheme? In the course of a design process, often the results of different versions are merged to a new version. This leads us to a directed acyclic graph as a modeling scheme, as given in Fig. 2.26.

The graph given above may be interpreted as follows. Version V_0 is the generic version. Versions V_1, V_2, and V_4 are variants, while version V_3 is a revision of version V_2. Version V_5 – the final result – is a merge of versions V_3 and V_4. Hence, it is a revision of version V_3 as well as a revision of version V_4. As with a tree as modeling scheme, history information is managed by a directed acyclic graph if the numbering of versions implies their creation times.

To sum up, we have to bear in mind three modeling schemes:

- a linear modeling scheme;
- a tree-like modeling scheme; and
- a directed acyclic graph.

A directed acyclic graph is the most powerful scheme. Using such a scheme, we are able to distinguish between revisions and variants. Merging of versions as well as history information is supported. This does not mean using such an acyclic directed graph is always suitable. The modeling scheme to be used depends on the version interrelations that a designer wants to manage.

A very rough rule is that the modeling scheme to use depends on the design phases. It seems useful to have a tree-like modeling scheme during the conceptual design, a directed acyclic graph during the logical design, and a linear modeling scheme for the physical design.

2.5.4.3 Version and Configuration Management based on PRODAT

Versions in PRODAT are objects which are derived from one another. This directed dependency is maintained in a 'is_derived_from' relationship. It is possible not

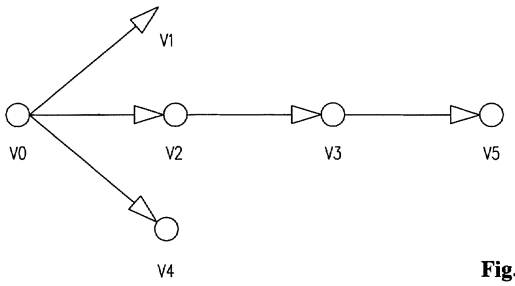


Fig. 2.27. Tree-like version interdependencies

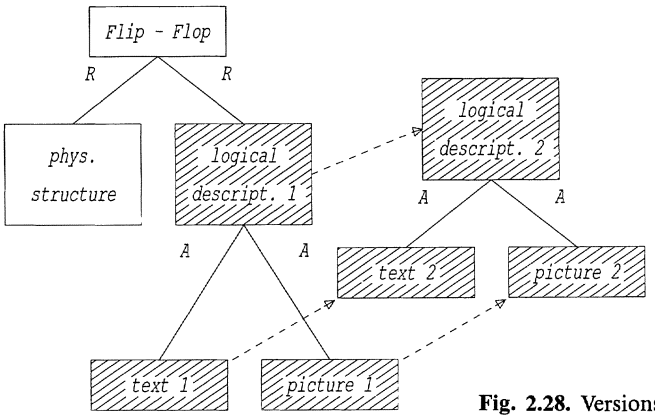


Fig. 2.28. Versions of a complex object

only to have a linear sequence of versions, but also to build version trees by deriving variants from one version which in turn may spread out arbitrarily. The resulting version tree is shown in Fig. 2.27. To create a new version of an object, the original object has to be released, which means this object cannot be changed further. If it is changed, dependent versions can no longer be derived from it. Only complete objects may be released.

The user can generate versions (by the create_version operation) from both simple and complex objects. If a version is created, a new object is generated automatically and the system connects the original and the new object by the version relationship described above. Both the original and the new object acquire version status; nevertheless, from the user's point of view they are treated like normal objects. The new version is not released and, therefore, can be arbitrarily manipulated.

Figure 2.28 shows a version of a complex object, here named "logical descript. 1". If the version "logical descript. 2" is created as a complex object, the user must specify that the unit for making a version is a complex object (instead of a simple one). Version relationships between the originals and the new versions are generated automatically. Each new object with version number 2 is now embedded in a different version graph. After this generation process, the same standard relationships as in the original object exist between the object and the subobject (in the new version). This is because versions of an object have the same object type, and version generation of

```
CONFIGURATION FF__conf
  ...                                   /* components of the configuration */
  ...                                   /* definition for each type part */
  LOGICAL__OBJ
  STRUCT
  customer CHAR (20),                   /* additional...*/
  price INTEGER,                        /* ...attributes */

  SELECTION                             /* selection rule */
    1 text__object XOR 1 picture__object,
  ...
;
```

Fig. 2.29. Configuration type definition

a complex object does not change the structure of the origin. Starting from this state, it would be possible to change the structure of the new version object or to manipulate anything else (e.g., the contents).

To save memory space, versions are compacted automatically by storing only differences between them [DITT85], [TICH85], so-called *deltas*, whenever it is possible. A delta can refer to either a previous or a subsequent version. See [BATZ87] and [RIED86] for a detailed discussion of the version concept.

Configurations can be seen as a logical grouping of objects; but it is usually necessary to provide some additional information describing the configuration. For example, a configuration could consist of a customized software package which is taken from a complex object containing all potential parts of a package. Configuration-specific data could then be the name of the customer, the date of delivery, or the price.

PRODAT configurations therefore contain an "inventory" of objects together with user-defined attributes for each object. Configurations are described in a special configuration type definition which contains selection rules similar to the completeness rules mentioned above (Fig. 2.29). The selection of objects must happen in accordance with the relationship involved. For instance, *all* children of a choosen object must be picked up (if using the 'has__component' rule) or *excactly one* child must be picked up (if using the 'has__alternative' rule). As a result, the user can influence the configuration process by fixing a certain relationship at type-definition-time.

Creating a configuration is done by indicating the configuration type desired and the simple objects involved. A conformance test with the configuration type definition is performed automatically. If any rule is violated, the action will be rejected; the same is true if any of the objects involved is not released.

Figure 2.30 gives an example of a configuration; it originates from the complex object shown earlier.

The main advantages of a separate configuration concept are as follows: configuration structure is predefined to a certain degree by type definition; configurations can be treated like "normal" objects; and they can be supplied with extra individual attributes. For a more detailed discussion see [BATZ87] and [BAUM88], which also deals with archiving of versions and configurations.

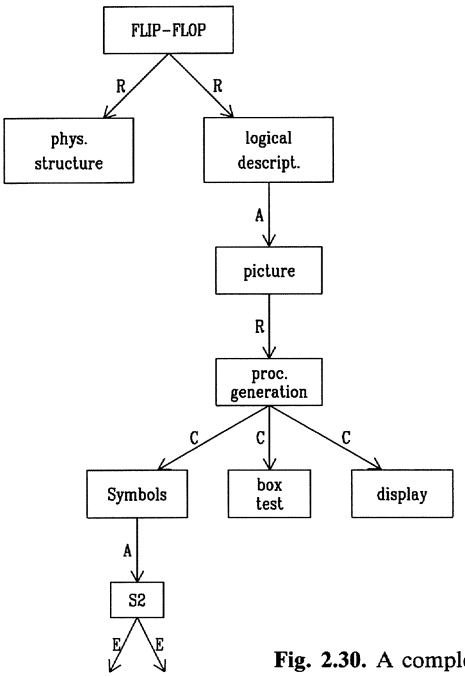


Fig. 2.30. A complex object

2.5.5 Generating and Entering Data

In this section we will analyze in detail the procedure of entering data in the database and what must be done first to generate these data. The first subsection deals with ways of generating and entering data in different applications.

Generally, engineering design systems handle data at different points. Data arise between the user and the user interface management, between the user interface management and the application programs, or between application programs and the database system, for example. Generating or entering in this context concerns the data at the database interface. Therefore we will examine primarily data sent to the database by the other components of the design system. But we have also to take into consideration the entering and generating of data at other interfaces in our design environment, because of their influence on the behavior at database interface.

The importance of the whole system of which the database management system is an integrated component, is outlined first. Next, the criteria for characterizing the generation and entering data are described. By means of these criteria we treat the subject in commercial applications, and present the differences in engineering applications. We will thereby make apparent the interfaces important for generating and entering data, and will analyze the handling of data by tools.

2.5.5.1 System Environment

An engineering database system is an integrated part of a comprehensive engineering application system. The data handled in such an environment are dependent on the

individual parts that process and produce data. Although our main interest is the data produced at the database interface, all components involved in processing data must be taken into consideration. Thus, when we want to discuss generating and entering of data we have to look at the architecture of systems and distinguish these particular components. Starting with components which communicate directly with the database, all other components involved must be analyzed with respect to their types of data, and the way these data are generated.

First, we outline some criteria as a basis for examination. Then we apply these criteria to a typical commercial application to give a better understanding of generating and entering data, and to show the contrasting requirements in engineering applications.

2.5.5.2 Criteria for Characterizing the Generation Process

When analyzing and comparing the generation and entering of data for different applications, we need common criteria. The following points are used for later discussion of data handling by applications both in commercial and engineering environments:

- *process* of generating data;
- *type* of data;
- *quantity* of data.

The *process* of generating data describes the route data takes from its origin to the database interface. Points of interest are the application (dialogue or batch process, or complexity of the data processing) and the influence of users (degree of interaction, use of input devices).

For the *type* of data it is important to know whether the application deals with simple attribute values or with structured objects. An example of the first kind are flat tuples in a relational database. Data of the second kind can be objects which are composed of other subobjects or relationships. The type of data used can be changed by the application (a graphic structure used by an application may go on to be transformed into longfields for entry in the database).

To describe the *quantity* being entered into a database a distinction must be made between size and number of units. Whereas the size of units specifies whether attributes, tuples, longfields, or complex objects are being entered, the number of units is a measure of the amount of data produced in a design step.

By means of these criteria ways of generating and entering of data in typical applications can be analyzed.

2.5.5.3 Commercial Applications

Generally, applications using the database in an commercial environment hardly differ. The components in a commercial system and their use are shown in Fig. 2.31.

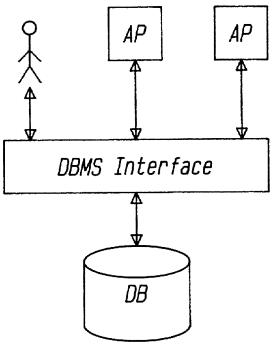


Fig. 2.31. Commercial applications

The following kinds of different applications are usually found in such systems:

- stand-alone DML users;
- simple interactive applications;
- batch applications; and
- conversion programs.

In *stand-alone* DML applications the *process* of generating data is very simple. The user refers to data structures and data types, i.e., entity types with attribute types and relationship types, defined by the schema, and enters the data to be stored directly into the database. The *type* of data is the same as defined by the schema, i.e., in commercial applications usually simple attributes. Also the *quantity* of data entered at one time (DML session) is very small, both in unit size and number of units.

Applications with *simple* interactions, e.g., flight booking or accounting in bank applications use specific actions and data like filling out a screen mask. The *process* of such programs is retrieving data from the database, making some calculations and updating the database. The *type* and *quantity* of data in interactive commercial applications are comparable to stand-alone applications.

By *batch applications* we mean monthly salary, statistics, settlement, or similar applications. In such a batch *process*, many data can be retrieved from database, calculations done, reports produced, and also many updates or store operations can be done. As in the other cases we have only a simple *type* of data. The *quantity* (only number of units) entered in the database is going to be large, but that is no disadvantage in batch.

In every database environment there exist some data *conversion* programs, which transform already existing information in machine readable form into the structure needed for the defined database schema. Such conversion *processes* can run in batch. The *type* of data generated in these programs is simple, because data types are basic and structures are predefined. The *quantity* of data is generally very large; an example is the number of units in staff data.

Let us now summarize the facts for generating and entering data in commercial applications. In most cases it is a simple *process* to generate the output data from certain input. Sometimes data is generated interactively by user. But there are also simple *types* and small *quantities* of data handled. Only in batch processes is the *quantity* of data large. In all cases the size of units is small.

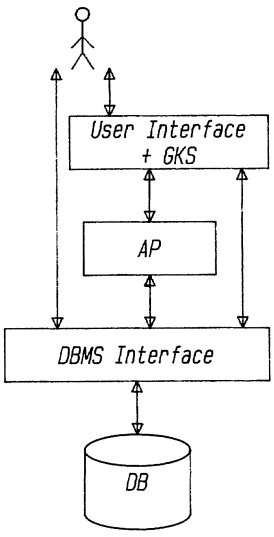


Fig. 2.32. Engineering applications

2.5.5.4 Engineering Applications

Commercial applications were introduced to emphasize what entering and generating means and how it is done. Now we describe a fundamentally new situation in engineering design. As in the previous section we first present the components of an engineering design system. An engineering design is a complex structure composed of many components. Figure 2.32 shows the rough architecture of an integrated engineering design system. We can discern three main parts:

- user interface management and device independent graphics system;
- database management system; and
- application modules.

We briefly describe these parts and then consider the different data flows between the components. The user interface management system realizes the dialogue between user and application modules. The *device-independent graphics system* manages the representation of graphical output produced by the application or directly extracted from the database. It is also responsible for graphical input in cooperation with user interface management. The *database management system* is the central part in the design environment. The *application modules* comprise all tools required for the design. This architecture may not be seen as a rigid structure. When we describe user interface management and the graphical kernel system as one component it does not mean that this cannot be divided into two parts. But we do not discuss in detail whether the user interface management is above GKS. In this section, only the existence of such components and their data communication is important. All the data flows discussed in the following do not have to be present in every design environment, because their presence depends on the configuration and the systems integration. Interfaces exist between:

- user and database management system (DBMS);
- user and user interface management/GKS (UIM/GKS);
- UIM/GKS and application modules;
- UIM/GKS and DBMS; and
- application modules and DBMS.

The interface between *user and DBMS* serves mainly direct queries to the DBMS by the designer. The user's most frequent interactions are performed by the *interface to UIM/GKS*. The interface between *UIM/GKS and application modules* is necessary to control the user's applications and graphic representations of application results. The last two interfaces, between *UIM/GKS and DBMS*, and between *application modules and DBMS*, can be seen as one single interface, if we take the UIM/GKS as an application from the database point of view.

In the context of engineering systems the analysis of the tools communicating with the database must be done. An engineering system consists of three classes of tools (see also [KATZ85]):

- synthesis tools;
- analysis tools; and
- tools for information management.

In the first class we find *synthesis* aids for design capture, which transform the design description to a machine processable form. The simplest way is digitizing a pencil and paper specification or using a language-based text editor for creating some program code. In VLSI design we find integrated circuit (IC) geometry editors for specifying the processing masks for circuit fabrication or schematic editors for creating a description of a hardware system at the logic gate level.

The *process* is marked by high human interaction, often using graphic input devices. The application programs transform the human input (graphic input) to the internal representation, which can be – depending on the database schema – complex objects or longfields, which determine the *type* of data for the database.

Geometric descriptions of designs produce a large data *quantity*. Both the units to be entered into the database (complex objects or longfields) and the number of units (e.g., many transistors) are large.

Mapping tools map the high level description of an object into its physical implementation. Examples for this type are programming language compilers or module generators in software engineering, and PLA (Programmable Logic Array) generators in VLSI design. Placement and routing tools, which place subobjects in a two- or three-dimensional space and make the interconnection between them, are also part of the synthesis class. All these tools generate data like machine language instructions, fabrication mask geometries, wrap lists or printed circuit board (PCB) artwork, or ship or building blueprints.

A designer makes fewer interactions in these kinds of *processes*. We can detect a decreasing human interaction (Fig. 2.33).

The *type* of data is comparable with that of the last class. Furthermore, one can notice an increasing *quantity* (e.g., stick diagrams are expanded to physical layouts).

Another important set of tools is available for *analyzing* a design. A simulator models the behavior of a system by using an abstract description of it. In most cases the model can be formulated mathematically. With topological analysis tools the cor-

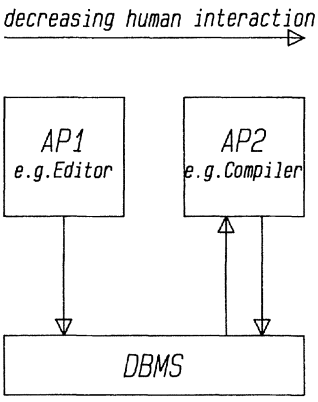


Fig. 2.33. Human interaction

rectness of the layout and interconnections of subsystems can be checked. An example is a geometric design rule checker in VLSI design. A third class of analyzing aids are timing analyzers for finding critical timing paths in VLSI designs or profiling the run times of a program in software engineering.

When examining the data handled by this class of applications, we have to distinguish between dynamic and static analysis. Tools for static analysis like topological analyzers or timing analyzers take a particular representation of design as input and produce a report describing the errors encountered, a list of critical paths, etc., as output. The analysis is independent of designers' input; it uses representational data produced in another, earlier step of the design. Dynamic analyzing tools, which comprise all kinds of simulators, produce outputs which are critically dependent on the input set. Therefore systems must manage large quantities of data that are separated from representational data like input or test data cases and the corresponding output. The resulting output must also be stored in the database for later use, e.g., in the test phase or for redesign.

Static analyzers *process* data produced by an earlier design step whereas dynamic analyzers need some additional data like test data or stimuli. These data can be given by human interaction or by database. Applications without human interaction can run in batch. Analyzing tools produce large data *quantities* which must be stored in a database for later use.

Finally, we consider tools for *information management.* An engineering design system must provide facilities to query information about general design management data like state of design parts, versions, releases, or designers involved in a design. Furthermore the design management has to guarantee the consistency of designs by starting complex consistency check applications.

The *process* of design management is highly interactive. It is marked by input from humans, who normally enter data of simple *type* and *quantity.*

2.5.5.5 Example

When generating the structured object 'Flip-Flop' as shown in Fig. 2.17 of Sect. 2.5.2, one must either use an interactive object editor or the procedural interface, which is

named PQL (PRODAT Query Language). The latter case is shown here. A structured object must be generated by successively calling the create__object operation for each node of the graph. After this generation process described below, one can use functions to:

- update and read attributes, or read or write the contents;
- create or delete relationships;
- traverse the object structure; and
- archive designed objects.

The syntax for the basic create functions is as follows: (If a parent key value is null, there is no parent.)

```
create__object:
crob    (in: db identifier, object type, object name, parent object key, out: object key)

create__relationship:
cred    (in: parent object key, child object key)

/* setting a variable for the database identifier */
SET     $DB,        CELLBIB;

/* creating the first level or root object */
CROB    $DB,        FLIP__FLOP,         flip-flop,          0,          $parent1;

/* creating the second level or all next direct subobjects */
CROB    $DB,        PHYS__OBJ,          phys.structure,     $parent1,   ;
CROB    $DB,        LOGICAL,            logical__descr,     $parent1,   $parent2;

/* creating the next level */
CROB    $DB,        TEXT,               text,               $parent2,   ;
CROB    $DB,        PICTURE,            picture,            $parent2,   $parent3;

/* creating the next level */
CROB    $DB,        SEGMENT,            segment__file,      $parent3,   ;
CROB    $DB,        PROGRAM,            proc.generation,    $parent3,   $parent4;

/* creating the next level */
CROB    $DB,        INCLUDE,            symbols,            $parent4,   $parent5;
CROB    $DB,        MODULE,             boxtest,            $parent4,   ;
CROB    $DB,        MODULE,             display,            $parent4,   ;
```

2.6 Integrated Systems and Methods Bases

2.6.1 The Concept of Integrated Systems

The software environment in which CAD systems are developed is usually characterized by:

- FORTRAN, or – more precisely – the FORTRAN dialect of the available computer;
- several subroutine packages for access to non-standard features of the computer installation such as graphic devices, data bases, etc.;

- the job control language of the particular computer; and
- the implicit assumption of the availability of certain hardware resources (a certain amount of memory capacity, for instance).

Transfer of a CAD system from one installation to another – assuming that the two environments are not identical – requires:

- modification of those FORTRAN statements that are not common to both dialects;
- adaptation of those parts in the system that have been influenced by the computer architecture (such as the word length);
- adaptation of the package calls to other packages (at least some of them);
- adaptation of the job control language;
- adjustment to more stringent hardware resources.

In the 1970s, a number of so-called "integrated systems" were developed [SCHL74]. An integrated system consists of a set of subsystems for solving problems from various engineering disciplines, and of a "system nucleus" whose facilities are shared by all the subsystems. Its primary purpose is:

- to provide a user-oriented environment for formulating and solving application problems from various disciplines in specialized subsystems;
- to support the development of such subsystems (in particular CAD systems), and to minimize the work necessary for adapting a given system to a new computer environment.

This goal is achieved by localizing all the environment dependency within the system nucleus.

In addition, the system nuclei support the development of application systems by providing higher-than-FORTRAN-level capabilities that are often needed. Examples of such capabilities are: data structuring, language processing, program and data management, documentation management, and user guidance.

The idea of concentrating basic software for all applications in a nucleus has also been adopted in other systems, e.g., in IPAD [FULT81].

CAD system nuclei envelop the basic computer (hardware and manufacturer-supplied software) and hide it behind a software machine, which provides facilities for constructing higher-level software machines:

- facilities of the *nucleus* may be used by the CAD system developer for providing new CAD capabilities (such as three-dimensional modeling). The term "subsystem" is used for these new CAD capabilities. If the subsystem is useful for solving problems related to a large class of objects (a finite element or a line-drawing subsystem, for example), it is called a *problem-oriented subsystem*;
- the capabilities of one or more problem-oriented subsystems may be combined by an application programmer to formulate the design tasks for specific objects of design (such as welding machines). Due to the fact that all subsystems belong to the same family (being based on the same nucleus), they may be combined freely without the danger of conflicts. Subsystems on this level are generally *product-oriented*;

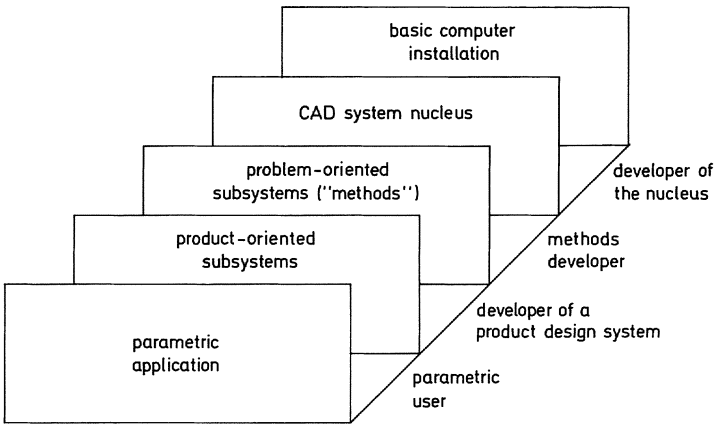


Fig. 2.34. The level concept in an integrated system

- a product-oriented subsystem may still leave some parameters open. The *parametric user* may specify such parameters and perform variations on a basically fixed design.

Figure 2.34 illustrates the various levels of the nucleus concept.

The existence of distinct levels of applications has also been recognized in the area of CAD turn-key systems [GRAB79], [GRAB81]. Pioneering work on system nuclei was done at MIT starting in the mid-1960s. The "Integrated Civil Engineering System" or ICES [ROSS76] is still the most widely known CAD system nucleus. (The term "Integrated System" is generally used synonymously to indicate what we prefer to call the "nucleus".) Other systems followed the same philosophy, placing emphasis on different aspects (portability, efficiency, ease of subsystem development, interactivity).

The systems DINAS [BEIE76], [BEIE78], GENESYS [ALCO71], IST [PAHL78], and REGENT [SCHL76], [SCHL81b] belong to this class of CAD system nuclei. We will discuss REGENT in more detail as a representative of this class.

Every implementation of the REGENT system consists of

- the nucleus;
- a number of subsystems for general-purpose use; and
- any number of application-oriented subsystems.

The nucleus itself consists of software machines for user support at various levels (see Fig. 2.34):

- subsystem development;
- definition of a schema for the subsystem data structure;
- generation of modules;
- definition of a subsystem language (POL = "problem-oriented language" or "product-oriented language");
- subsystem execution; and
- subsystem documentation.

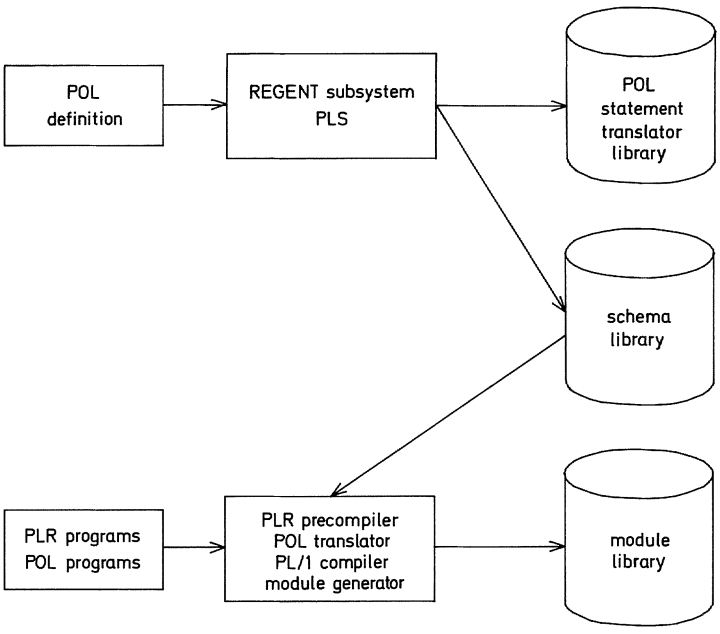


Fig. 2.35. The generation of a REGENT subsystem

The architecture of a typical system of this type (REGENT) [SCHL81b] is shown in Fig. 2.35 and Fig. 2.36. The components of a subsystem (subsystem language, schema for the data structure, the modules, and the messages) are stored in three libraries: a schema library, a library for the POL statement drivers, and a module library. Subsystem data are stored in a data base and/or files (Fig. 2.37).

When these systems were developed, computer power was not yet so readily available as it is today. The "integrated systems", though oriented towards interactive data processing, found their dominant application in batch processing. Furthermore, they were developed and operated mostly in a batch environment, and with typical batch subsystems such as the finite-element ICES subsystem STRUDL II [NELS72] or the REGENT subsystem GIPSY [ENDE80], [ENDE81] for geometrical and graphical applications.

In this environment, much emphasis was placed on providing an easy-to-remember problem-oriented language to the user. Sample Listing 2.1 shows a short program in the GIPSY language with the result presented in Fig. 2.38. Since the principal man-machine communication technique has shifted towards interactive communication (menus, mouse, and the like) the "integrated system" approach is no longer being pursued. The basic techniques developed are still found in current systems:

- tools for adapting the man-machine interface to user wishes;
- dynamic management and invocation of modules;
- management of complex data structures and their dynamic behavior;
- management of systems of files;
- data base management for archiving purposes; and
- tools for extending the capabilities of an existing system.

Sample Listing 2.1. A program in the solid-modeling language GIPSY. This program will produce Fig. 2.38.

```
ENTER GIPSY:
 DECLARE SURFACE(6) PLANE;
/* SPACE(n) means: a SPACE element with up to n surfaces                    */
/* SPACE elements are convex                                                */
 DECLARE INNER__SPHERE SPACE(1);
 DECLARE CUBE SPACE(6);
/* BODY means: it is a general, not necessarily convex body                 */
 DECLARE OBJECT BODY;
/* length units are CM, unless specified otherwise                          */
  CHANGE UNITS LENGTH CM;
/* a PLANE is defined by a point in the plane and by the vector pointing to the material
   side                                                                     */
 SET SURFACE(1) = PLANE(POINT( 0,  0,  0),POINT(-3,  0,  0));
 SET SURFACE(2) = PLANE(POINT(-3,  0,  0),POINT( 0,  0,  0));
 SET SURFACE(3) = PLANE(POINT( 0,  0,  0),POINT( 0,  0,  3));
 SET SURFACE(4) = PLANE(POINT( 0,  0, -3),POINT( 0,  0,  0));
 SET SURFACE(5) = PLANE(POINT( 0,  0,  0),POINT( 0,  3,  0));
 SET SURFACE(6) = PLANE(POINT( 0,  3,  0),POINT( 0,  0,  0));
/* Both cube and sphere are centered at (-1.5, 1.5, 1.5)                    */
 SET CUBE = SPACE(SURFACE(1)+SURFACE(2)+SURFACE(3)
                  +SURFACE(4)+SURFACE(5)+SURFACE(6));
 SET INNER__SPHERE = SPACE(BALL(POINT(-1.5,1.5,1.5),1.8));
/* we now subtract the inner sphere from the cube                           */
 SET OBJECT = BODY(SPACE(CUBE-INNER__SPHERE));
/* we specify the direction of projection,
          the projection plane by its normal vector,
          and the location of the projected origin of the 3D-coordinated system in
          the projection plane                                              */
 CHANGE PROJECTION PARALLEL   (-100.,-95. ,+130.),
        PROJECTION PI__NORMAL(-100.,-95. ,+130.),
        PROJECTION ORIGIN      (   60. MM , 40. MM );
/* we specify the representation of lines, pen 4 is thick                   */
 CHANGE STANDARD PEN(4),
        INVISIBLE LINETYPE DASHED;
 OPEN PLOT DINA(6) BROAD;
 PLOT(OBJECT);
/* we now wish to plot the shifted object, but suppress hidden lines        */
 CHANGE INVISIBLE LINETYPE OMITTED;
 PLOT(SHIFT(OBJECT, 5. CM, -5. CM , 0. ));
END GIPSY;
```

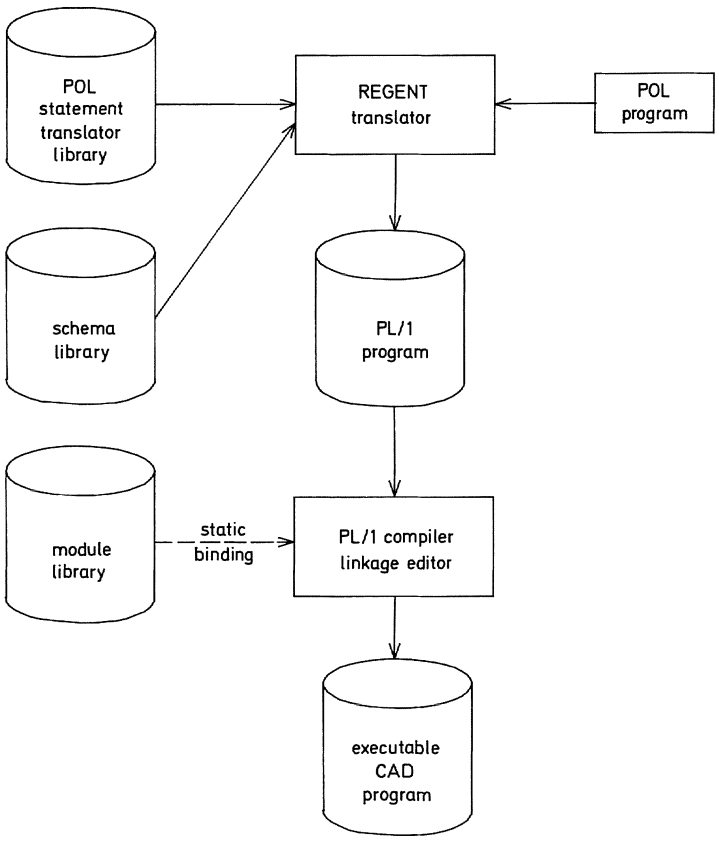


Fig. 2.36. The generation of an application program with REGENT

2.6.2 Methods Bases

Similar system architectures have become better known as "methods bases" or "banks of methods" [DITT79]. In a sense, the "integrated systems" idea is a predecessor of the methods base concept [SCHL82]. However, for methods bases, documentation support and interactive user guidance have been included in the concept right from the beginning. See, for instance, [BART80], [EGGE81], [NOLT76], [SCHI77], [SCHL81b]. The experienced user would employ a methods base in much the same way as he would use an integrated system: he would state and solve his problem by formulating it in a problem-oriented language. The inexperienced user, however, will be guided from a first and perhaps imprecise statement of his problem (maybe even in natural language) to the formalism required for the application of the appropriate method. The methods base will try to recognize in the initial problem statement patterns that are characteristic for the range of methods that it can propose; it will ask for further data about the problem until the appropriate method has been identified; it will then continue to ask for additional information as is needed for ap-

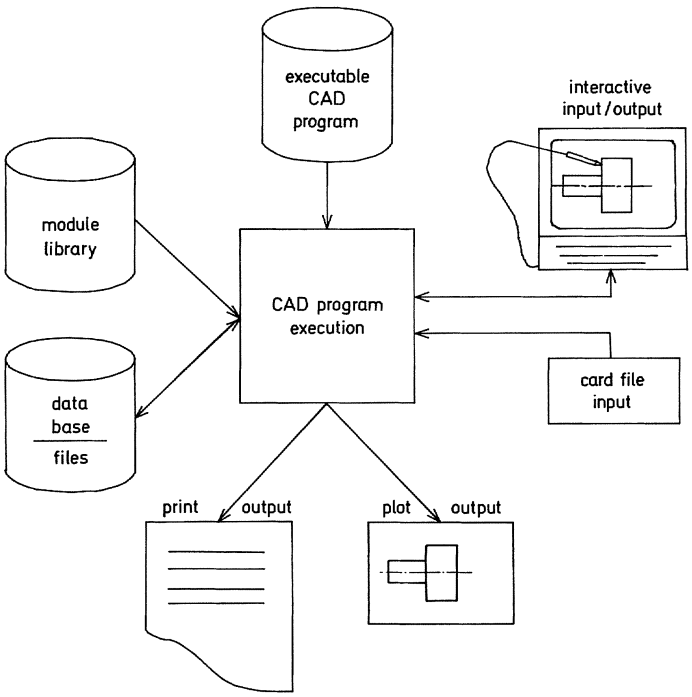


Fig. 2.37. The execution of an application program under REGENT control

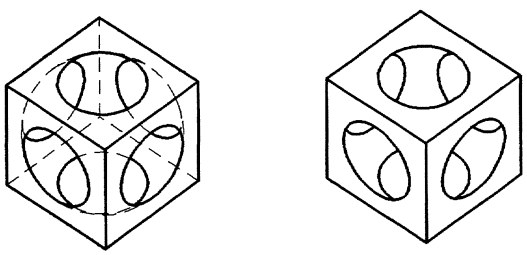


Fig. 2.38. The result of a sample program written in the problem-oriented language GIPSY for geometrical applications

plying the method. Keywords and/or a hierarchical structure will be used to navigate in the problem domain.

Whether or not a significant amount of design knowledge can be preprocessed into the formalism and structure required for this approach has yet to be seen. It is perhaps not a question of whether this approach is principally feasible but simply a matter of time and man-power required to create this formalization, which may cause the failure of this concept if attempted on an overly broad scale. In limited areas, however, methods bases will not only help users to obtain solutions for isolated problems, but will also provide a means for computer-aided education in the corresponding problem domain.

2.7 Configuring, Evaluating, and Choosing CAD Systems

Computer-aided Design has developed rapidly in the last 20 years. The hardware and software has varied greatly and is now accessible to many potential users. Drafting tasks are solved today from PC to network environments.

Some reasons for the user's difficulties in deciding when and how he should enter the CAD field are:

- increasing hardware and software offerings;
- uncertainties regarding standards;
- high initial costs;
- lack of measurable justification;
- lack of qualified personnel; and
- time span required for planning, selecting and introducing a CAD system.

The approach given here proposes how this decision problem can be handled by showing a global model of the decision process and dedicated implemented prototypes. Such a decision system, in spite of the usual consistency and updating problems, is able to save considerable time and manpower. Still, its major advantage is that it is the beginning of a methodology for maintaining and developing knowledge in this field, and better tools to deal with it. Prototyping helps to classify, to force the judgment of necessities, to analyze the knowledge development and to approach a methodology. The choice, introduction, and expansion of CAD systems may be properly and effectively carried out by analyzing the firm environment, the technological support and the estimated costs and savings. These form the domains of the model and its elements, which are organizational, technical and economic facts, and rules.

2.7.1 The CAD Evaluation Model

The model (Fig. 2.39) contains the following aspects:

Management

This aspect represents the analysis and planning of costs and benefits. The methods and precision in which the involved parameters are considered supports the local economic correctness of the decision.

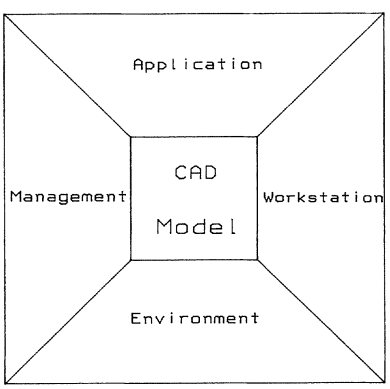


Fig. 2.39. CAD evaluation model

Environment

The environment which directly influences the choice, introduction, and operation of a CAD system is described in terms of departments, processes, functions, personnel, and tools.

Application

The application processes, functions, restrictions, and conditions are described, forming challenges, objectives, and requirements.

Workstation

A hardware/software classification including individual components, attributes, configuration dependencies, and market analysis is described.

The implementation approaches deal with these four aspects. The formal model nevertheless does not partition the theme into these four aspects but into three, which are: *organizational, technological,* and *economic.* The application aspects are included in the organizational description. From these domains, decision processes, parameters, facts, and rules are designed and implemented.

The following formal definition is expressed in first order predicate calculus and its model theory. This formalization originated through discussions and a proposal made by W. Kasprzak [ENCA85]. One positive aspect is the conceptually simple transition to an implementation in the form of an expert system.

For the CAD model we build a language Ω_c of the class Ω

* $\text{sorts}/\Omega_c/ = S_{OP} \cup S_{TP} \cup S_{EP}$,

  S_{OP} = The sort name for organizational parameters
  S_{TP} = The sort name for technological parameters
  S_{EP} = The sort name for economic (cost and saving) parameters

* The correspondent set of variables $X_c = OP \cup TP \cup EP$

* $\langle \text{CAD model} \rangle = \langle\langle M_c, \psi \rangle, Y, \Phi \rangle$

M_c = an implementation of the language Ω_c

$\psi = T \cup I_1 \cup I_2$

T = The axioms of the language Ω_c
I_1 = Inference of the type: $formula(OP) \rightarrow formula(TP)$
(They describe the transition from a description in organizational parameters to a description in technological parameters)
I_2 = Inference of the type: $formula(TP) \rightarrow formula(EP)$
(The transition to an economic description in cost and saving parameters)

$Y = R_1 \cup R_2 \cup \delta$ = the analysis methods

R_1 = so-called metarule in inference form
$formula_1(OP) \rightarrow formula_1(OP) \cap formula_2(OP)$
(They describe the possibility of expansion of the given formula to a formula with additional organizational parameters)

R_2 = so-called metarule, which contains statements upon the choice of the better variants

$$formula_1(TP) \cup formula_2(TP) \rightarrow formula_1(TP)$$
$$formula_1(EP) \cup formula_2(EP) \rightarrow formula_1(EP)$$

$\delta: t_1/x_1, \ldots, t_n/x_n$ = Substitution rules applied to inferences I_1, I_2, R_1, R_2.

$\Phi = (\phi_{x^*}/x^* \in X_c^*$ = a family of interpretations, which are distinct only in the values for the variables of X_c.

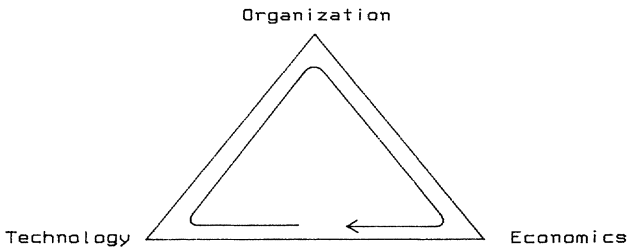


Fig. 2.40. Mutual influence of domains

Since this model is accompanied by implementation approaches, real problems are regularly tested and reformulations of the structure are attained. The general domain structure is represented in Fig. 2.40.

Table 2.1. Phases of CAD planning and introduction

Initial investigation:	Evaluation of the state-of-the art Concept development for CAD introduction
System analysis:	Analysis of the company's current situation Analysis of CAD introduction strategies Catalog of functional and technical requirements
Choice:	Market analysis and preliminary choice of alternatives Test the alternative CAD systems Analysis for CAD system expansion Evaluation of alternatives and decision
Preparation:	Room preparation Personnel planning Personnel training and information supplying Computer operation planning
Introduction:	Pilot installation Generation of existing data and drawings
Operation:	User orientation and control CAD system maintenance, expansion, and improvements

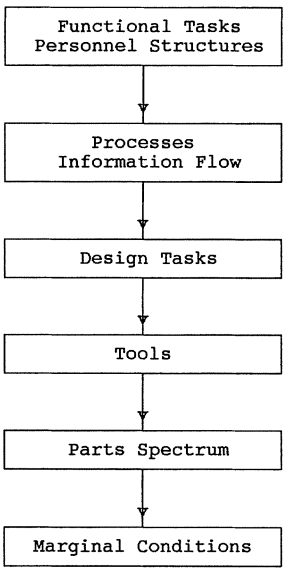


Fig. 2.41. Procedure to obtain the current state

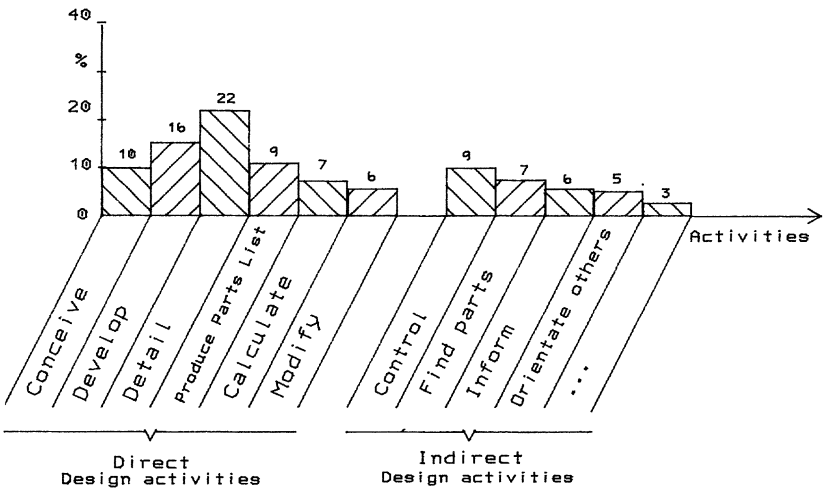


Fig. 2.42. Activities distribution in a design department

2.7.2 Phases of CAD System Choice and Introduction

It is just a matter of time until all branches of industry use CA technology. Table 2.1 gives a short description of activities which should be followed in order to guarantee a controlled CAD planning and introduction.

Spur suggests in [SPUR84] the following steps in a procedure (Fig. 2.41) to obtain the current state of the environment and shows in Fig. 2.42 an example of activities distribution in a design department.

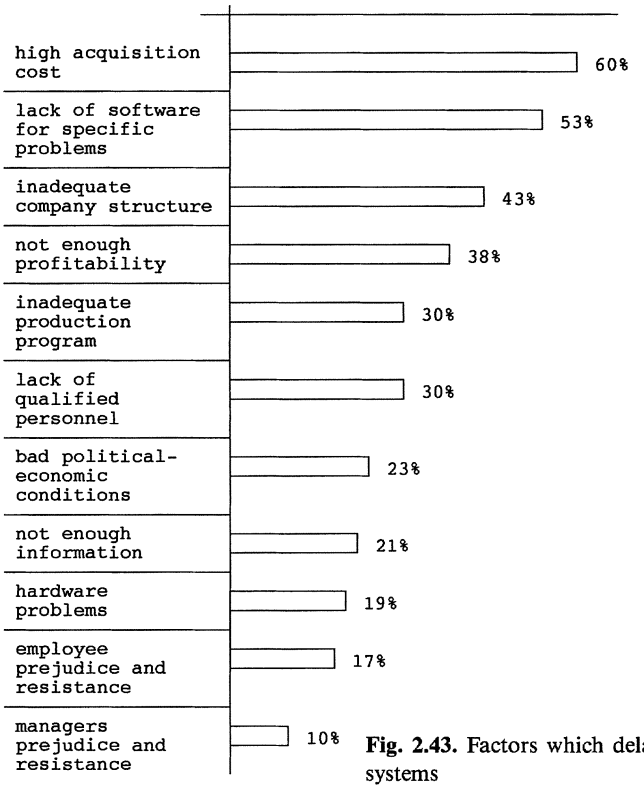


Fig. 2.43. Factors which delay the introduction of CAD systems

2.7.3 Restriction Factors Versus Advantages

The factors described in Fig. 2.43 delay the introduction of CAD systems [POTH86]. They correspond to a questionnaire answered by 450 firms.

The possible benefits of introducing CAD systems are also well documented in [WILD86], [WEDE86]:

- increases the potential for competition;
- reduces design and production costs;
- increases drawing productivity;
- increases company flexibility;
- improves product quality;
- increases automation of the design process;
- improves the workstation;
- enables dynamic development of a CAD/CAM technology; and
- enables the synergic effect of information flow.

Hatvany, Newman, and Sabin also give other advantages [HATV77]:

- more efficient utilization of design skills;
- homogenization of technology;
- more precise design documentation; and
- easier updating of documents.

2.7.4 Organizational Parameters

These parameters describe the organization in order to support an analysis of the environment in which a CAD system will be introduced. The organization's functional behavior and the operation of the application processes which correlate to design activities are described by sets of parameters, facts, and rules.

The upper-level parameters are:

- Application field
  A hierarchical structure under an application field such as, e.g., mechanical, determines the general events and requirements of the field. The following logical sentence shows an example:

  Mechanic $\cap$ machine-tool design $\rightarrow$ FEM $\cap$ NC...Graphic Workstation
  So that: Application $\rightarrow$ Type of CAD-Software $\cap$ Type of Workstation...

- Personnel and qualification
  Determining the available manpower provides better grounds for predicting requirements according to objectives and tasks to be fulfilled. This supports task distribution planning and calculation of the number and type of workstations.
  The VDI (Verein Deutscher Ingenieure – Association of German Engineers) suggests, in guideline 2216 [VDI_87], the following functional distribution in respect to a CAD operation team: designer (the german terms according to acquired skills are: detailer, drafter, constructor) with or without CAD experience, CAD application programmer, systems analyst, application analyst, and CAD instructor.

- Integration with existing computer network
  If integration to an existing computer network (or a simple computer resource) is to be achieved, this existing system must be technically specified. According to the implementation approaches discussed later in this context, a dialogue interface for technical parameters must accept configurations as input. This adds some restrictions to the choice of new hardware/software components.

- Standards and documents used
  According to the application field, standards and catalogs of parts support the designer's work. These, and the way in which they are available, are important tools. They are: standards drawings, lists, tapes, disks, microfilms, databases, knowledge bases... .

- Project coordination
  Tasks can be formulated in details or phases, each of which could have attached to it the personnel capacity, the necessary tools, and the computer support. Additionally, both man and machine are responsible for specific tasks. This analysis can be accomplished by an adaptation of the smallest time method yielding the smallest activity method.

- Supplier and client dependence
  The type of communication (as in standards and documents) and the hardware/software technical restrictions support the description of this dependence. Observe that integration restrictions are always possible at the technical level.

- Product and production philosophy
  The spectrum of products and partial products, of design activities, of drawing and of production processes, their connections, and a technical description of the tools used in the whole chain form a basis for the comprehension of this philosophy. Needless to say, we do not claim to give a complete solution to this description. Our aim is to provide the problem with some simple (e.g., tree, graph) structures and analyze the evolution of the problem description and its successes and/or failures, suggesting structural and application-dependent changes.
- Flexibility of the organization
  Supposing there is an approximated description of the design and production process as to what concerns its technological application-dependent structure, it seems not unreasonable to estimate the flexibility of the organization for a variant structure.
- CAD system allocation
  All or part of the machines can be in one or more buildings, different sites, etc. This gives measures for cables, work procedures, and hardware/software configurations.
- Clearance of the task to be supported by CAD systems
  Through the firm structure and application fields, if the tasks are not yet determined, they can be deduced from the attributes and functions of the CAD systems on the market by giving the highest priority to the better covered fields (avoiding risks!).

These parameters and their subsequent levels are not claimed to be complete; they are a first approach to be refined. A more detailed discussion of these parameters with examples and the way of quantifying them is found in [ENCA84].

2.7.5 Technological Parameters

Targeting the basic purpose of choosing an adequate CAD system configuration, a detailed list of hardware and software components, attributes, and values constitute the elements of the classification. A combination of these elements following technical and expert-derived facts and rules yields a correct configuration. [MESS88a] gives a hardware/software classification of CAD systems which helps in building up an individual company-specific catalog of requirements.

Some characteristics may be pointed out as first tests:

- Software expandability;
- Portability;
- Documentation completeness;
- User interface friendliness;
- Software/hardware maintenance;
- List of users; and
- Market place for the products.

2.7.5.1 The Industrial Design Process

The palette of CAD activities serves an interesting and fruitful example of a field that involves a variety of domains. Their industrial application given in [HATV84]: product structure definition, concept sketching, design parameter definition, design logic rules definition, dynamic dimensioning and tolerancing, 3D model building, analysis and optimization, viewing and verification, layout and production drawing and lists, process planning, operation planning, NC-programming, and quality control.

Although each field (area of application, e.g., mechanical, electronic) has its individual themes and particular way of treatment, common data and operations are available; e.g., *numerical*, *geometrical*, and *nominal*, identifying and creating objects for sequential or parallel processing.

Today, a general software configuration for CAD systems [MESS88a] may be said to consist of a geometric kernel involving geometric objects and operations, a database system manager to enable manipulation of various objects, usually a library of available, attachable programs for use in certain applications and/or for connection to other phases of product development and manufacturing, and dialogue processes which enable the man-machine communication. Figure 2.44 shows this composition graphically.

The design activity begins with product planning; it then proceeds from conceptual to preliminary and finally to detailed design. After these activities, product information is used for manufacturing planning (Fig. 2.45).

In the product planning phase the initial product is at least roughly defined in what concerns its general functions and attributes. The conceptual design uses this specification and shapes the product by attaching its initial geometric definition, and by testing its functions by simulating and measuring the results. The preliminary design is the first attempt to complete the product definition by testing the more complex functions, and also by testing for durability, for a large number of different cases, and exhaustively for some which appear to present oscillatory behavior. In the detailed design the product receives its present final documentation.

It is unavoidable that all phases may end up in "product returns", when the product should be redesigned from certain phases, determined by a history of the design process. The lack of centralized information on products, activities and processes

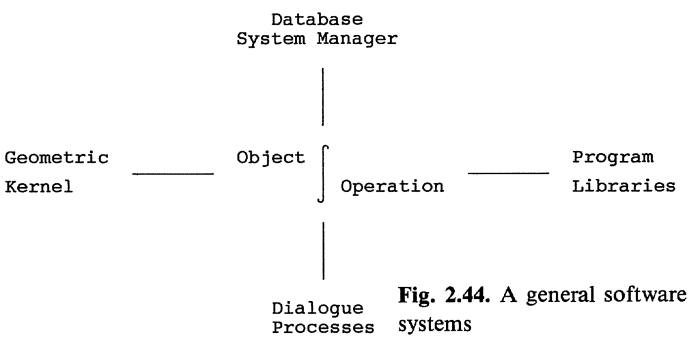


Fig. 2.44. A general software configuration for CAD systems

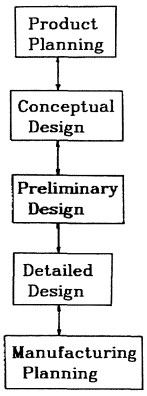


Fig. 2.45. Phases of the design process

causes these product returns. The manufacturing planning phase prepares and adds manufacturing-dependent information.

A) Product Planning

As stated in many reports, perhaps more than 90% of industrial design activity is based on variant design; this means that there is already a well-formalized description of an object (perhaps a product model) possibly with functions similar to the so-called new variant object. The task here is to find the object with the closest functions, and manipulate or improve these functions, according to the ones desired, by changing or adding data and attributes to the existing formalized object. Figure 2.42 shows a distribution of activities which do not include variant design; but on the other hand it presents an activity 'Find parts' which is actually basic for CAD design and which will support automatic search for similar parts.

For example, in planning the design of a new camera, the designer may search for an old type, and based on its functions, attributes and on his or catalogued knowledge (rules and facts), he may add new, desired functions, by varying some chosen parameters.

The remaining 10%, i.e., new design, may use very little of already existing objects and knowledge about them. The planning of such products assumes a much more refined knowledge base, also containing trial and error data and methods based on similar research.

In a broad view, it is clear that areas such as raw and synthetic material research, economics, market analysis, manufacturing, and specific application-dependent knowledge should also be considered. However, the establishment of such a structure demands an understanding which at the moment cannot be formalized.

The planning of a product is in fact a task which is done by a large number of experts and is usually based on a number of risks. The degree of complexity of the task rises even more if the dynamic changes in time to the planning of a product and to the product itself are considered. This suggests a continuous evaluation of the processes just executed against the current state of knowledge.

B) Conceptual Design

It is assumed that at the beginning of this phase the initial product specification is already set; this means the product has, although not complete, an initial definition of the functions and attributes. This provides the first data with which to continue the design in a stronger technical/scientific way.

The designer uses the tools he has at hand to develop his ideas and his knowledge in a systematic way to produce an idealized object. The usual representation schemes used in industry are engineering drawings, containing all sorts of information besides geometric. In the present configuration of CAD systems on the market, the designer does not have at hand sufficient support in order to take into consideration real engineering or manufacturing constraints. Structural analysis methods and a few mathematical simulations are still the only available support.

As mentioned earlier, as long as much of this work consists of manipulating existing parts and changing these parts, the designer should be able to get some amount of the task already formalized ready for use. On the other hand, for this formalization to exist, it is necessary that designers work with some sort of system capable of keeping track of their sequence of operations [MESS88b] in a variety of ways, which is also able to discard irrelevant sequences. By this variety of ways, it is meant that the surrounding areas of knowledge must be integrated to enable these topics to be taken into account.

So, in this phase the topology and the geometry of the object is fully treated and some simulation tests are executed.

C) Preliminary Design

This phase is considered as the testing stage for the designed product. The product is subjected to all conceivable sorts of tests, possibly by simulation. Tests involve geometric constraints, material resources, product lifetime, range of operations, manufacturing constraints, etc.

It is obvious that in order to be able to use such diverse knowledge, the formalization of domain-dependent procedures plays a very important role. Once more the combined use of mathematics and geometry appears, with descriptive observed laws (heuristics), which cannot be mathematically described but can be easily formalized, e.g., in predicate calculus, in the form of facts and rules.

The output of this phase consists, then, of an exhaustively tested product, whose data is delivered to the next phase with all the information grained in these tests plus the ones already achieved at conceptual design.

D) Detailed Design

The final product drawings will be elaborated in this phase. It is assumed that drafting standards and manufacturing regulations will be the major tools here.

Still, as in all other phases, a product may for some specific reason (e.g., impossible to produce according to a local manufacturing regulation), not pass this phase and therefore must be "sent back" to some previous stage. What previous stage should be selected is decided by the type of change necessary on the product, which is not always easy to determine. Decision routes, according to the type of failure, must be determined, based on an analysis of the stored operation sequence track [KIMU87].

This problem would be reduced if all possible information were available at all stages. But this is still unrealistic.

On the other hand, if a product succeeds through this phase, its documentation contains all the available information necessary for production preparation or manufacturing planning.

E) Manufacturing Planning

The specification of the available data and tools seems, as in all other areas of research, development, and production, to be the key point for detailed and programmed planning of the operations and processes to occur. This means, very basically, the specification of machines, paths, and work operations on a product in the production cycle.

Actually, the degree of detail contained in the specification, plus the way in which it is stored and will be used, determines the a priori optimization in the production and design processes. This manufacturing specification supports, therefore, not only process planning, machining, and assembling, but also product planning, where it serves as a filter to the planning of ready-to-make products, avoiding misunderstandings and asynchronous or manual information transfer.

On one side there is information about the machines and paths; on the other side, the object to be manufactured contains detailed information about its geometric shape, its technological data and its surrounding engineering knowledge. Based on this information the sequence of processing operations must be determined or matched to an existing sequence.

Although the inclusion of many features listed here has not yet succeeded in real commercial systems, prototypes based on a small set of parameters are being developed [MESS86], [FUJI85], [TOMI86], [KIMU87], [WARM87].

F) Interaction and Integration

The construction of CAD software systems demands, therefore, a well integrated structure to support product models, which include geometric models, basic engineering knowledge about the product, and dialogue processes.

CAD systems on the market, because of the structures of the implementation languages and state of the art of hardware/software environments, offer very few possibilities, if any, for the development of appropriate dialogues and knowledge structures to represent the situations at each phase of product development properly.

The task of a designer in a CAD system is to devise, on availability of some objects and operations on objects, a new object which fulfills desired functions. He needs a set of objects, operations, and confirmations, some of which are very dependent on his application. These objects have, in most cases, geometric properties and should therefore be presented to him as such. Graphic interactive interfaces represent not only a suitable tool for the designer or draftsman, but also for all system users, as the development of graphic dialogues evolves, presenting them with easier and more comprehensible symbols for operation.

Figure 2.46 facilitates the comprehension of the integration of activities involving the design, drafting, and manufacturing of products with respect to a product model.

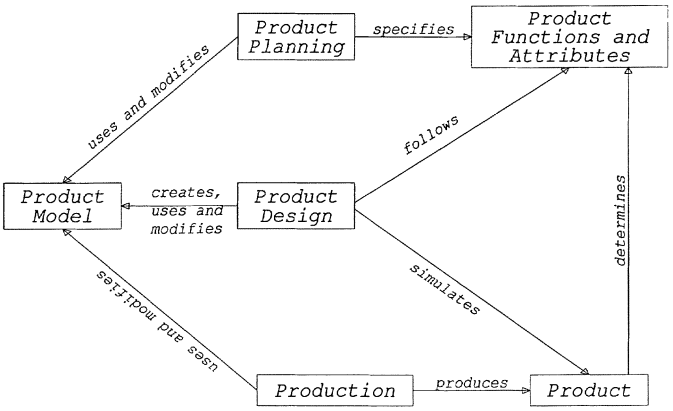


Fig. 2.46. Design, drafting, and manufacturing of products

2.7.6 The Economics of CAD Systems

This section deals with the balance of cost and benefit of CAD systems. Quantification methods and examples are shown, giving an introduction and indicating advanced and specific literature in this field. But the main purpose is to provide a practical approach, which is at least partially applied in the implementation approaches for evaluation.

2.7.6.1 The Initial and Annual Costs

If the technical parameters of a configuration are already set and there are components on the market which build up an equivalent configuration, the initial and annual costs for this configuration may be roughly estimated, based on real value calculation and ideal experimental approximations.

The initial costs are:

- hardware and software investment;
- room preparation;
- cables and connections;
- company current state analysis;
- CAD introduction planning;
- CAD systems choice;
- introduction preparation;
- training;
- installation and integration;
- dead period; and
- data input.

The annual costs are:

- personnel;
- training;

- data integrity and backups;
- material and power consumption;
- hardware and software maintenance;
- insurance;
- rate of interest;
- rent; and
- depreciation.

The list of costs in this first level of the classification specifies the elementary costs. Some mathematical formulas may be used for cost calculations, as for estimating the costs with training. Otherwise, experimental rules, e.g., of percentage estimates based on the investment or on other quantities are applied.

The VDI guideline 2216 [VDI__87] suggests for the training period in a 2D system 2 to 4 weeks and in a 3D system 6 to 12 weeks. Now, if a formula to calculate the costs with training is to be found, the following may be used:

N_i = Number of employees of the class i
TW (2D, 3D) = Training time in weeks
CW (*in, out*) = Costs per week
DC = Drop out costs per week

$$Training\ Costs = \sum_{i=Drafts}^{i=Manager} N_i \cdot TW_i(2D,\ 3D,\ \text{etc.}) \cdot (CW\ (in,\ out)_i + DC_i)$$

2.7.6.2 The Benefits

Benefits are often classified as direct and indirect. Most of them are very difficult to quantify, if at all. Figure 2.47 gives a first classification [ENCA84], where three benefit components are derived from a technical support. They are: productivity, quality, and flexibility.

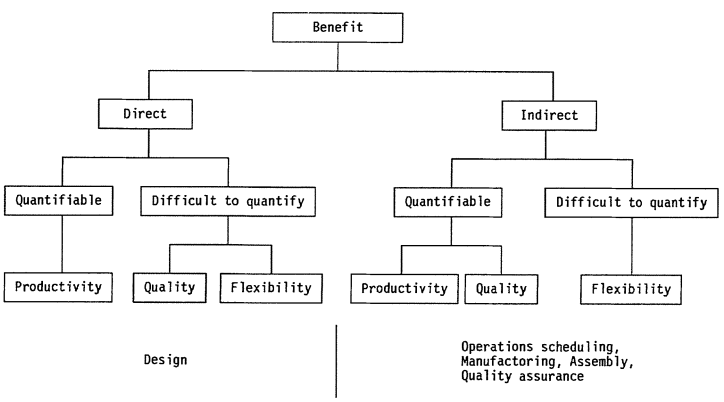


Fig. 2.47. Technical benefit components

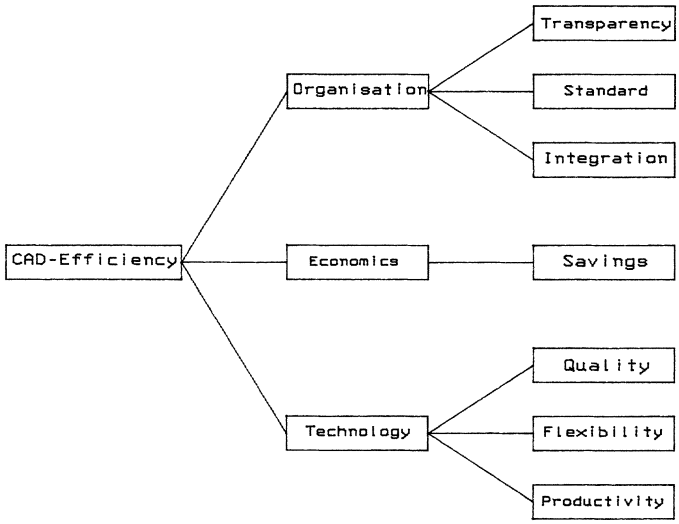


Fig. 2.48. Technological, economic, and organizational benefit components

Figure 2.48 helps in understanding the expansion of benefit components, which also influence industrial processes and improve the industrial knowledge considering explicitly economic and organizational components.

Methods for Quantifying the Benefits

The main tasks to be solved are:

- the understanding of the types of potential benefits;
- the tools or factors which influence these benefits; and
- the measurement of how much a certain tool can influence a type of benefit.

The methods considered here apply to drawing tasks although more elaborate design activities such as the development of a product based on a similar one may in the near future also be considered, which in some sense will not displace these methods as long as they apply to a task division in its measurable elements. This is exactly the basic topic of the first method.

1) Smallest time method

A typical engineering drawing is classified into its elements (material specification, rotational parts, cuts dimensioning, etc.) to which acceleration factors dependent on the available hardware/software configuration are designated. The total acceleration factor (gain of productivity) is then given by the formula:

$$TAF = \frac{\sum_{i=1}^{i=n} AF_i \cdot NE_i}{\sum_{i=1}^{i=n} NE_i}$$

Modification of a drawing		(2.95)	
Test, quality and delivery regulations		( )	
Manufacturing regulations		( )	E
Material specification		( )	
Raw material specification		( )	N
Surface texture symbols	6	(6.00)	
Additional specifications		( )	D
Contour elements	34	(1.80)	
Dimensionings	25	(6.20)	
Tolerance dimensioning	10	(5.00)	
Geometrical and positional tolerance		(5.60)	
Hatch-areas	23	(3.40)	
Scale specifications		( )	
Dimensioning arrows	24	(3.50)	U
Auxiliary lines	37	(4.00)	P
Lines		(1.30)	D
Special symbols		(3.00)	A
Macros		(4.00)	T
Texts	31	(2.57)	E
Unwinding and/or revoluted surfaces		(7.50)	
Views	1	(4.00)	
Straight sections		(1.75)	
Penetrations		( )	
Separate details		( )	
Enlarged details	2	(3.00)	
Symmetric elements	1	(5.97)	T L
Rotated and/or translated elements		(6.00)	O E
Copied elements		( )	T R
Standard grade (1-3)		( )	A .
Variants		(5.10)	L F
Datatransfer to: - Methods of calculation		(4.00)	A
- Production resources		( )	A C
• Parts list preparation		( )	C T
• Production plan preparation		( )	C O
- NC-machine		(10.00)	E R
- different types of drawing		( )	
		( )	3.65

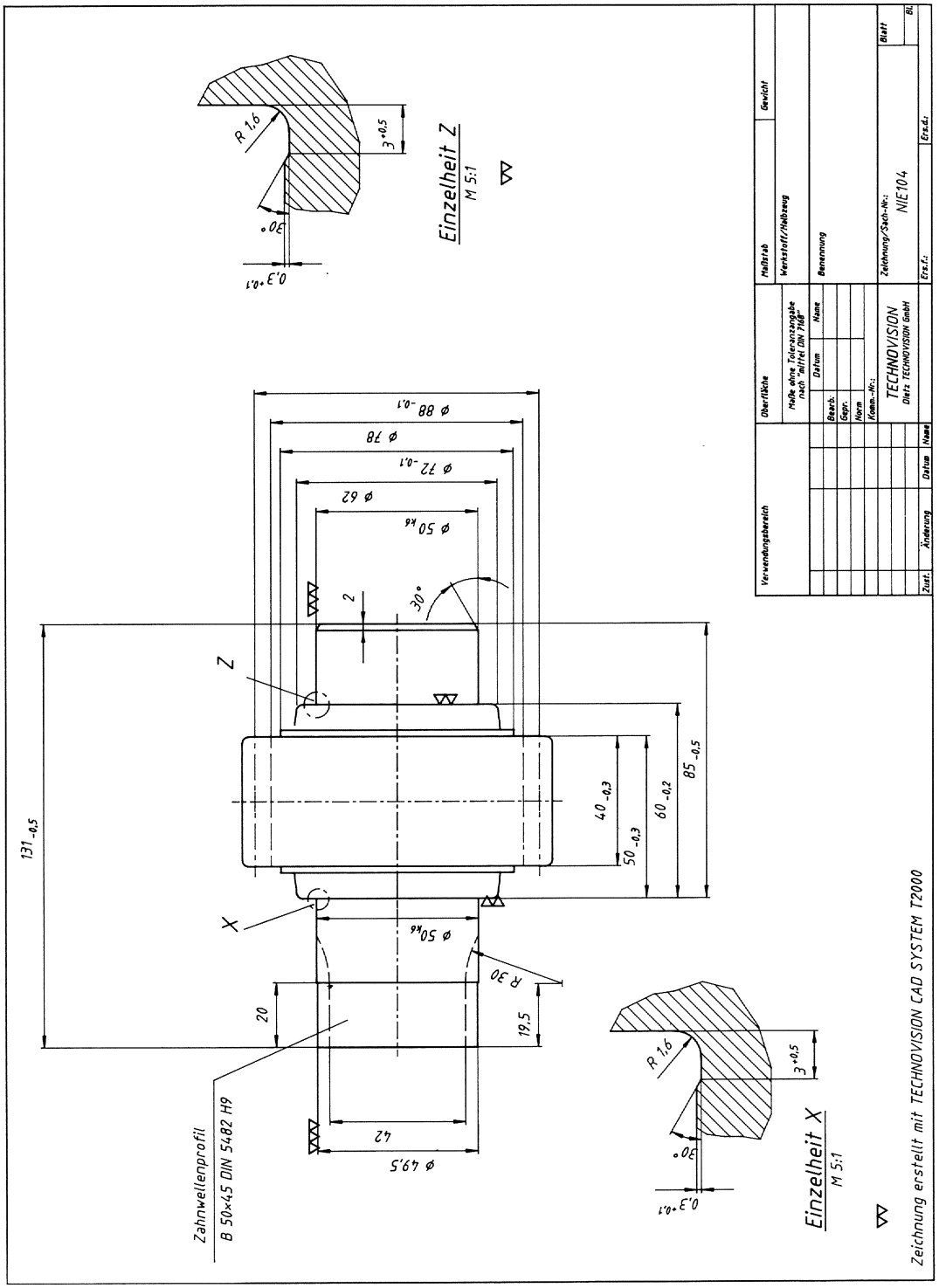
Fig. 2.49. Example of a drawing classification with acceleration factors and number of elements

The result is the mean acceleration factor for a drawing with the given specification supported by a CAD configuration. The acceleration factors AF_i for individual elements are based on measured values according to experimental studies. NE_i is the number of elements 'i' of a drawing with corresponding acceleration factor AF_i.

The difficulties encountered here are 1) the composition of the set of elements, 2) the corresponding hardware/software support for one element, 3) the acceleration factor interval of an element, and 4) the individual acceleration factor designated based on hardware/software support. Experimental values are used, based on industrial practice. Figures 2.49 and 2.50 show an example.

This method can be extended to treat acceleration factors for design activities in general, by splitting the design process into known tasks supported by hardware and software tools.

Fig. 2.50. Example of a typical drawing


Einzelheit Z
M 5:1
Einzelheit X
M 5:1
Zahnwellenprofil
B 50×45 DIN 5482 H9
Zeichnung erstellt mit TECHNOVISION CAD SYSTEM T2000
TECHNOVISION
Dietz TECHNOVISION GmbH
Zeichnung/Sach-Nr.:
NIE104
Benennung
Werkstoff/Halbzeug
Maßstab
Gewicht
Oberfläche
Maße ohne Toleranzangabe nach "mittel DIN 7168"
Verwendungsbereich
Blatt
Bl.
Ers.f.:
Ers.d.:
Änderung
Datum
Name
Norm
Gepr.
Bearb.
Komm.-Nr.:
131 -0.5
85 -0.5
60 -0.2
50 -0.3
40 -0.3
20
19.5
42
R 30
2
30°
R 1.6
3 +0.5
0.3 +0.1

In general, if the time required for the manual production of a drawing is given, and this task is accelerated through hardware/software tools by a given acceleration factor, the cost reduction can be calculated.

2) Global comparison

Two drawing tasks are compared. In the easiest case two drawing elements are compared. If it is assumed that for a given drawing task the time is known, the time for the second task is dependent on the new environment and on the comparison of both environments. To find similar tasks it is usually necessary to know the history of the drawing [REIN85], [VDI_87], [KIMU87].

3) Efficiency measures

The idea is to classify the CAD activities in drawing, calculation, modification, and parts lists production. According to an integration with manufacturing activities, NC programming and manufacturing scheduling are also considered [HETE85].

The following formulas give an idea of some average reference quantities to apply as (1) measures of productivity, (2) flexibility, and (3) information quality:

$$\textit{Drawing Time} = \frac{\textit{Total time for CAD Drawings}}{\textit{Number of CAD Drawings}} \tag{1}$$

$$\textit{Number of Drawings per Designer} = \frac{\textit{Number of CAD Drawings}}{\textit{Number of Designers for CAD Drawing Activities}} \tag{1}$$

$$\textit{Turnaround Time for Drawing Tasks} = \frac{\textit{Total Sum of Turnaround Times of CAD Drawings}}{\textit{Number of CAD Drawings}} \tag{2}$$

$$\textit{Number of Mistakes per Drawing} = \frac{\textit{Number of Noticed Mistakes in CAD Drawings}}{\textit{Number of CAD Drawings}} \tag{3}$$

$$\textit{Number of Mistakes per NC Program} = \frac{\textit{Number of Noticed Mistakes in CAD NC Programs}}{\textit{Number of CAD NC Programs}} \tag{3}$$

4) Cost-saving calculation

This method is based on a time division for the execution of a drawing task, which is given by CPU-Time+I/O-Time+Drawing-Time, and on comparisons between attribute values of two CAD hardware/software systems, which in turn modify the time portions yielding a different number of drawing tasks per year. The cost saving per

annum is then calculated. The task is assumed to have been executed with an initial CAD system which will give the initial time partition [ENCA84].

According to the time partition, the hardware and software attributes are considered as follows:

Drawing Time (D_t)

Two attributes are considered at the moment:

- (a) Hardware dynamics at the workstation

  The type of hardware support such as rotation, translation, panning, zooming, clipping, fill area, frame buffer, z-buffer, etc. are taken into account.

- (b) Software drawing (designing) methods

  This describes the software functionality. Some examples are: 2D drawing in planes, cuts, and sweeping yielding 3D object generation, uniform dialogue interfaces through drawing processes, calculation and NC programs, available drawing library, variant elements and consistency rules, etc.

CPU-Time (CPU_t)

- (c) MIPS, MFLOPS
- (d) Main Memory

  Although the sizes of the main memory of two computers may be directly compared, memory management should be compared as well, especially if virtual memory management is done.

- (e) Software Techniques

  Some examples are: analytic versus approximated volume description, search algorithms, database management, computational methods, expert systems. These are only guidelines to be used and very often depend on the application.

I/O-Time (I/O_t)

- (f) I/O-processors

  The I/O-time may be extremely reduced if I/O-Processors, a fast bus and fast parallel interfaces are present.

- (g) Multiprocessors

  The use of a multiprocessor architecture may reduce queueing problems. I/O-processors, multiprocessors and architectures should have their attributes described in order to enable comparisons.

Increasing the comprehension of hardware and software components and attributes, combinations of components, attributes and values will provide better means for classifying them properly and for estimating their comparisons.

The number of CAD hours per year is $CAD\ hours = N_d \cdot N_t \cdot D_d \cdot A$.

N_d = Number of days a year (= 250)
N_t = Number of terminals
D_d = Drawing time a day (= 8 h)
A = Terminal load level (~ 70%).

The total time for the drawing task is $T_t = CPU_t + I/O_t + D_t$.

Assuming that a certain drawing task has been done with a CAD system (S_1), that the partial time was measured, and that the values of the attributes of the CAD system S_1 are known, the number of drawing tasks per year is:

$$N_{tasks}\ (S_1) = \frac{CAD\ hours}{T_t(S_1)}$$

Taking another CAD system (S_2) with known attribute values, one can estimate $T_t(S_2)$ and therefore $N_{tasks}(S_2)$ based on the comparison S_2 to S_1.

$$T_t(S_2) = D_t(S_1) \cdot \frac{1}{(a_2/a_1) \cdot (b_2/b_1)} + CPU_t\ (S_1) \cdot \frac{1}{(c_2/c_1) \cdot (d_2/d_1) \cdot (e_2/e_1)}$$

$$+ I/O_t\ (S_1) \cdot \frac{1}{(f_2/f_1) \cdot (g_2/g_1)}$$

$$N_{tasks}(S_2) = \frac{CAD\ hours}{T_t(S_2)}$$

The relation between the systems is given by $R_{2,1} = \frac{N_{tasks}(S_2)}{N_{tasks}(S_1)}$.

It is equivalent to say that the number of CAD hours regarding both systems will be changed by this relation, so that:

$$CAD(S_{2,1}) = CAD\ hours \cdot R_{2,1}$$

The CAD saving hours is therefore $CAD\ (Saving_h) = CAD(S_{2,1}) - CAD\ hours$.
The personnel saving is given by $CAD\ (Saving_p) = CAD\ (Saving_h) \cdot manhour\ rate$.
The overall saving per year is then given, with the total costs per year subtracted:

$$CAD\ saving_{year} = CAD(Saving_p) - Total\ costs\ per\ years$$

Where the total costs per year may be given as:

$$C_y = Depreciation + Interest + (Maintenance + Personnel) + Room + Power + Consumables$$

Table 2.2. Cost saving percentage caused by CAD introduction

	Operations scheduling	Manufacturing	Assembly assurance	Quality	
Reduction of mistakes	5%	5%	5%	10%	defective goods, increased consumption (personnel, material, machine)
Personnel optimization in production	20%	10%	10%	10%	Personnel cost in production
Optimization of machine		10%	10%	10%	machine cost

5) Cost reduction based on percentage of savings

This method is based on a triple (department, advantages, cost) which depicts possible advantages gained by CAD introduction specifically estimating the percentage of saving.

The following example shows the intent of this method. As was shown before by the advantages and possible benefits, introducing a CAD system not only means a change in the structure of the design department, but it influences gradually integrated or possibly to-be-integrated processes.

The purpose of this method is 1) to recognize the departments, 2) to recognize the advantages that may occur, and 3) to estimate the percentage of cost saving implied by the advantage.

Take for example a CAD system which has two other modules besides drafting: FEM and NC programming. The introduction of this CAD system will gradually result in cost savings in the departments under the advantages and cost classes shown in Table 2.2.

6) Benefit Value Analysis

This method consists basically of:

a) Construction of a tree structure representing the object (CAD system configuration) to be analyzed, where child nodes are partial criteria, and leaves are the attributes which will present a set of possible values. CAD systems on the market, which serve as alternatives, will be analyzed based on this structure and on the values of the attributes.
b) The criteria are weighted according to their importance for the user, so that each level of criteria belonging to an upper criterium sums up to 100%.
c) The possible values of an attribute are classified in a table, as intervals, or as a function, assigning a grade (N, usually between 1 and 5) to a value.

For each alternative configuration a cost-benefit value is calculated through the formula:

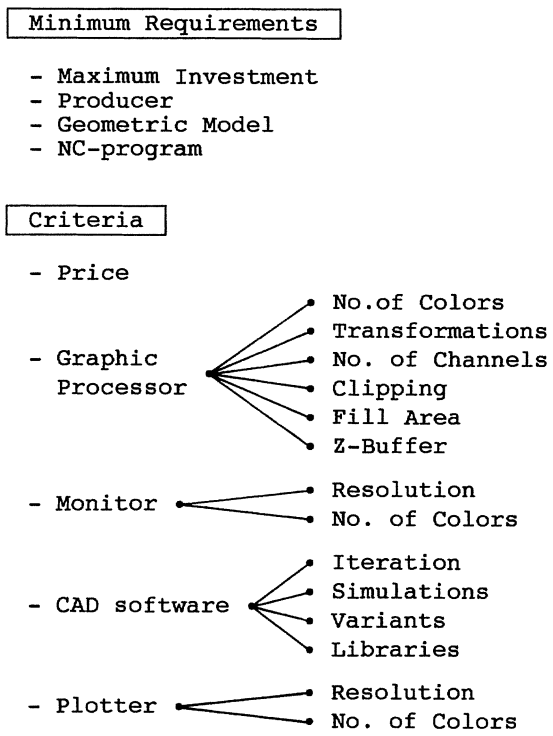


Fig. 2.51. Example of objectives for analyzing a CAD system

$$BV = \sum_{i=1}^{i=n} N_i \cdot W_i \ .$$

It calculates the sum of the partial benefits by multiplying the corresponding grade N_i given to the value of the attribute 'i' by the accumulated weight W_i given by the product of the weights in the path.

The number of alternative systems analyzed by the method is naturally reduced based on the fact that at the beginning some 'must have' attribute values (minimum requirements) are usually desired. This means that only alternatives providing these values will be analyzed. Figure 2.51 shows an example.

2.7.6.3 Methods for the Analysis of the Economics of CAD Systems

The methods in this section are not concerned with the quantification of benefits and cost-saving estimation based on a hardware/software configuration. Some concepts, formulas, and one example for the analysis of CAD economics are given.

Method 1: Sellmer and Schmidt consider in [SELL84] the productivity ratio P_r as the relation between the costs after and before CAD introduction. "It costs P_r more to produce a drawing by using a CAD system. The break-even point will be reached when drawing production occurs P_r times faster."

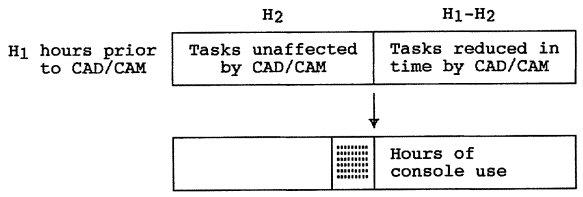


Fig. 2.52. Drawing time reduction

C_p = *average personnel cost* (*per hour*)
C_m = *console, machine or workstation rate* (*per hour*)

$$P_r = \frac{C_m + C_p}{C_p}$$

$$C_m = \frac{\textit{Total CAD system cost per year}}{\textit{CAD hours per year}} \; [\textit{cost/hour}] \; .$$

Method 2: Chasen in [BLAU80] begins with the following question "How can cost benefits be realistically measured in terms of parameters familiar to everyone?" He also uses the term productivity ratio, but defined in terms of manhours.

H_1 = *manhours prior the introduction of CAD*
H_2 = *manhours unaffected by CAD*
H_3 = *manhours at the console*
K = *estimated indirect benefits* (*worst case* $K = 0$)

$$P_r = \frac{H_1 - H_2}{H_3}$$

$$\textit{Cost Reduction} = K + (H_1 - H_2) C_p - H_3 (C_p + C_m) \; .$$

Chasen given in [BLAU80] some interpolative tables, estimates and examples to help the analysis. Figure 2.52 shows the time reduction through CAD introduction. One can also expect a certain additional influence on the unaffected hours.

Method 3: Scott in [SCOT81] points out that CAD should not only improve the design but should also make it more profitable! He considers two measures for the productivity ratio:

$$\textit{Total Productivity Factor} = \frac{H_1}{H_2 + H_3}$$

$$\textit{Computer Productivity Factor} = \frac{H_1 - H_2}{H_3} \; .$$

Method 4: Grabowski [EIGN80] observes that since 1900 the productivity in manufacturing has increased 1000%, whereas in design only 20%. The investment per workstation in manufacturing has reached according to the level of automation between US$ 25000 and 250000, whereas for design on average only US$ 1500 was invested. The disproportion of investment explains the difference in productivity gain.

Grabowski considers the productivity ratio also as a measure of the smallest time method. This means that for a typical drawing, values of productivity ratio may be based on measured values for equivalent tasks, assuming the environment is the same.

Because the costs with CAD C_{mach} must still be smaller than the conventional costs without it C_{conv}, the following expression holds, $C_{mach} \leq C_{conv}$.

N = Number of drawings per year
G = Average time for the production of a drawing
B = Operational costs per hour.

$$N_{mach} \cdot G_{mach} \cdot (C_p + C_M + B_{mach}) \leq N_{conv} \cdot G_{conv} \cdot (C_p + B_{conv})$$

$$P_r = \frac{G_{conv}}{G_{mach}} \geq \frac{(C_p + C_m + B_{mach}) \cdot N_{mach}}{(C_p + B_{conv}) \cdot N_{conv}} .$$

Method 5: Warman gives in [WARM78] examples for system costs and system justification. A sensitivity analysis is also given based on the following parameters: number of workstations, working hours per day, productivity ratio and manhour rate. The justification is:

S = System cost
I = Installation cost
OC = Operating costs (maintenance, supervision, power, consumables)
OB = Operating benefit
D = Depreciation

OB = number of workstations · shifts · number of designers
· draftsman saveable cost p.a.

The efficiency of the first year reaches 75% on average. After the second year it may be calculated as 100%. The saving is then given by:

Cash benefit = OB · 75% − OC
Accounting benefit = OB · 75% − OC − D.

The savings here are based on the saveable costs of a draftsman.

Method 6: The economics in [KRAU84] is also based on the productivity ratio. First the current costs of the design tasks are calculated:

Current cost = Cp · actual hours US$/year
= US$ 30/h · 30000 h/year = US$900000/year

The initial costs are:

$$\begin{aligned} Ci &= \text{Hardware} + \text{Software} + \text{Adaptation} + \text{Training} + \text{Room preparation} \\ &= \text{US\$ } 1\,030\,000 + 100\,000 + 25\,000 + 25\,000 + 25\,000 \\ &= \text{US\$ } 1\,205\,000 \end{aligned}$$

The annual costs that represent the CAD costs are:

$$\begin{aligned} Cy &= \text{Depreciation} + \text{Interest} + (\text{Maintenance} + \text{Personnel}) + \text{Room} + \text{Power} \\ &\quad + \text{Consumables} \\ &= \text{US\$ } 241\,000 + 120\,500 + 170\,500 + 9000 + 5000 + 5000 \\ &= \text{US\$ } 551\,000/\text{year} \end{aligned}$$

The estimated costs determine the maximum personnel costs for CAD support.

$$\begin{aligned} \text{Estimated costs} &= \text{Current cost} - \text{CAD costs} \\ &= \text{US\$ } 900\,000 - 551\,000 \\ &= \text{US\$ } 349\,000/\text{year} \end{aligned}$$

The number of design hours per year with the CAD-System are:

$$\text{Design hours} = \frac{\text{Estimated costs}}{\text{manhour rate}} = \frac{349\,000}{30} = 11\,663 \text{ h/year}$$

The console rate or the workstation cost per hour is:

$$C_m = \frac{\text{CAD cost}}{\text{Design hours}} = \frac{551\,000}{11\,663} = \text{US\$ } 47.24/\text{h}$$

The break-even-point is therefore only formally reached when the productivity ratio achieves the following value:

$$P_r = \frac{C_p + C_m}{C_p} = \frac{30 + 47.24}{30} = 2.57$$

2.7.7 A Decision Support for Configuring, Evaluating, and Choosing CAD Systems

As we have seen, the planning, choice, and introduction of CAD systems demand qualified personnel and time for analysis. The proposal here is to build a system which is able to support this task. The domains to be covered are complex and for implementation purposes should be seen as an example of the task.

Table 2.3 describes briefly the most important domains, which were explained in previous sections.

Table 2.3. Domains supporting the decisions

Concepts	Modeling/Parameterizing
Environment	Reduced description of the design environment
H/S configuration	Classification of hardware/software tools Description of CAD systems on the market
CAD economics and benefits	Cost classification Estimation of benefits and savings

2.7.7.1 Implementation Approaches

The ideal system should be constructed in such a way that the organizational (OP), technological (TP), and economic parameters (EP) mutually influence one another. This is represented by Fig. 2.40.

Some general questions arise:

1) Based on an organizational description, which technological support is necessary and sufficient?
2) How can this derivation be explained and justified?
3) How complete must the organizational description be in order to achieve an acceptable technical specification?
4) What is the basic justification for the economic values and where are their limits?
5) What kind of data structures and manipulation could handle the combination of these domains better?
6) When should data and structures be modified?

Many other questions could be posed as well. It is also clear that no general algorithm exists. These questions are usually left for a certain strategic decision, which takes into account experience and knowledge in the fields. Structures for the manipulation of such complex matters are the subject of current research and do not yet represent theories, which can be explained, justified, and applied to practical problems.

A practical solution to the configuration, choice, and evaluation problems consists of a sequential approach which starts with an organizational description, derives alternative technical solutions, and applies economic analysis to these. This represents, in fact, a procedure which also enables returns and partial specifications.

A simple example shows this approach. Let us suppose an organizational description is based on three questions:

- Type of application;
- Drawing complexity; and
- Number of designers.

These questions imply alternative technical equipment of the following types:

- CAD software;
- Computer;
- Terminal; and
- Plotter.

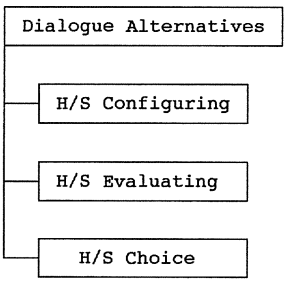


Fig. 2.53. General structure for decision support

Based on the values for both domains, alternative combinations arise which generate correct, complete configurations. These may then be economically tested following some procedures:

- Initial and current cost calculation;
- Benefit analysis through drawing capability; and
- Cost-saving calculation.

Figure 2.53 shows the general structure of a system for this type of analysis. The following paragraphs will be devoted to the clarification of requirements for the construction of such a system, which has been at least partially solved at the THD (Technical University of Darmstadt).

Having programs to support these tasks brings many advantages; but many restrictions on the construction of the first prototypes are also imposed, which make the system rather inflexible. These impositions are clear on what concerns the necessary data and structures.

Let us take the case of having to store information on the CAD systems currently on the market, in order to be able to search them under specific conditions. What attributes of a system should be stored? That is, what logical scheme will be implemented to store information about CAD systems on the market? Classifications of up to 5000 or more items (not yet values) are known [BRAN83], [IKO__83], [KRAU84], [ENCA84], [MESS88a]. But the use of such classifications is tiresome and diverts attention from really important attributes for a specific application. Furthermore, most attributes do not yield a precise advantage that could be directly taken into account for the economic and benefit analysis.

There is unfortunately no general subclass which could be taken for all cases. A practical decision is to consider the tools at hand and based on them implement partial solutions to cover some parts of the problem. One solution which was tackled, although it does not demand completeness, is the sequential general solution which can be represented by OP→TP→EP. To demonstrate this solution one can not make use of many parameters, but the ones used influence one another. The system implemented for this case uses a database system (CORAS) and GKS for the graphic interactive dialogue interface. The steps of the system may be described as follows:

User: Describes the organizational conditions
System: Suggests alternative CAD systems (H/S)

User: Chooses one
System: Possible CAD costs
System: Possible benefits and cost savings.

The description of the organizational conditions limits itself to the specification of the application field, the number of designers, the maximum investment, and whether an NC package is necessary. A configuration (H/S) is then built up from the database of CAD systems on the market. The CAD configurations are described by the following attributes: class of computer (mainframe, mini, micro, pc), main memory, price, MIPS, number of processors, number of I/O processors, geometry model, workstation hardware, and dynamics of interaction. Most of these attributes are considered with normalized scaled values in order to be able to compare two systems. The costs are listed according to initial and current costs. Some estimates are made based on experienced values to enable a first guess. These costs may be interactively changed and they may be printed as a report. Three methods of quantification (1, 4, and 5) are used to estimate the benefits and the cost saving. There is no doubt that a more detailed specification is necessary, but it serves to demonstrate the proposal.

If the analysis is restricted to a search of CAD systems on the market through a technical specification of the desired attributes of a system, a database application based on a refined data scheme of hardware/software attributes of a CAD system is sufficient. In this case the UDS (Universelles Datenbanksystem) running on a Siemens 7570 was used [DEDE86], [BECK87]. The interaction was implemented using a mask concept from UDS.

To handle the choice and the configuration of CAD hardware and software a specific data management system was also implemented [SÄNG86]. This system has the advantage that the scheme may be interactively modified. This saves an enormous amount of time and dynamizes the dialogue session. The graphic interaction is GKS-based.

One of the most used methods for the evaluation of computer systems, because it enables the evaluation of numerical and non-numerical attributes giving them grades according to a scale, and weights according to their importance for the user, is cost-benefit analysis, which was described in the section for quantification of benefits. The formulation of the sequential analysis OP→TP→EP was also implemented (Fig. 2.54). With a certain combination of OP values, combinations of TP

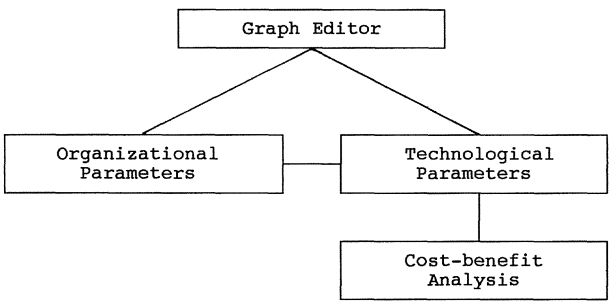


Fig. 2.54. Cost-benefit analysis: implementation structure

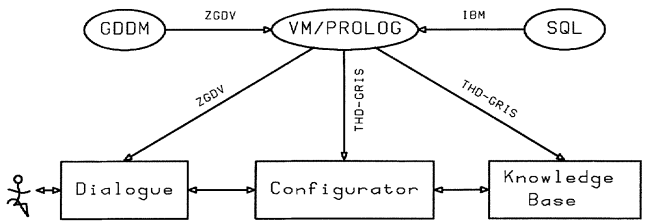


Fig. 2.55. Structure of an expert system prototype

values are associated (determined by experts), through which alternative CAD systems may be matched. These alternatives may thereafter be evaluated by the method. A direct specification of necessary technical attributes is also available. This implementation offers an interactive way of describing the organizational and the technical parameters through a graph editor. The graphs created have nodes that, in this case, are a hierarchy of attributes to describe OP and TP. The alternative CAD systems are generated choosing values for the attributes of the hierarchy. This approach was implemented in IfProlog and Turbo Prolog [SCHR86], [LEMP85], [THOM88].

If one considers the problems of maintaining an administration of configurations, it turns out that the amount of information about the available hardware and software plus the interdependencies among them increases very rapidly. The possible, the practical, the profitable, and the efficient forms of configuring are usually not methodically analyzed. They follow some local and arbitrary analysis which do not correspond to a scientific approach. These do not yet mention the problems of misunderstandings and mistakes. In addition to these two points, it must be said that graphic interactive dialogue interfaces do help users to check on-line, semantically intricate errors, eliminating many steps in the trial process.

The problem of configuring has also gained interest since DEC internally uses the system XCON to configure DEC computers. A similar approach was implemented at the THD (Technical University of Darmstadt) aiming at the configuration of hardware and software, using the data of IBM products and under a study contract with IBM Education and Research. This prototype of an expert system was written in VM/PROLOG and uses the graphic functionality of GDDM (Graphical Data Display Manager). A complete interface to the GDDM-Package was written for this purpose [SÄNG87]. The structure of the system is shown in Fig. 2.55.

The aim of the system is to help vendors configuring CAD systems for their clients. The system has a graphic interactive dialogue interface, through which a vendor may develop a hardware/software configuration. The client starts by choosing the areas of application he intends to cover and by giving the number of designers. The configurator (system) searches in the knowledge base according to the application, and based on the number of designers and configures a corresponding number of workstations and the appropriate software. As long as additional alternatives are possible, the system is able to show these to the client. These alternatives are ordered by increasing price.

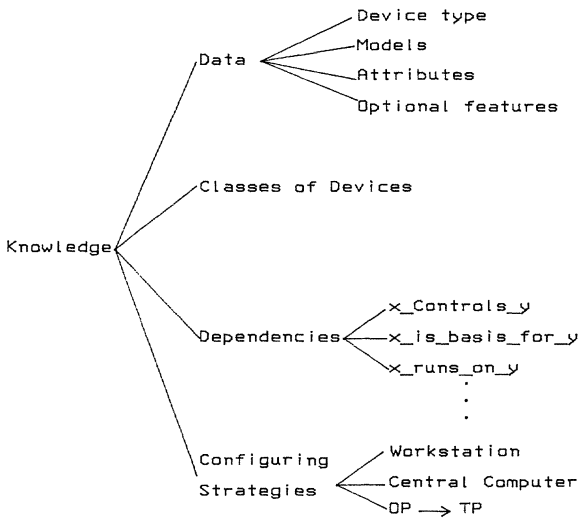


Fig. 2.56. Interpretation for configuring hardware and software

Modifications and expansions of an existing configuration are allowed. The system also enables a direct specification of hardware and software components of the configuration. In this case the configurator searches for restrictions in the knowledge base in order to show to the client only the expansions which are possible depending on the actual components of the workstation, and on the entire actual configuration.

Figure 2.56 shows the knowledge interpretation concerning the configuring problem.

The knowledge base contains:

- hardware and software IBM product models, standard attributes, and optional features;
- the technical and heuristic facts and rules representing the dependencies and the possible connections between IBM products; and
- the rules and facts connecting the application, the environment, and the components of a configuration.

The configurator enables:

- the interpretation of facts and rules of the knowledge base passing them to the graphic dialogue interface;
- the choice of strictly possible products at each step of the configuration process;
- the state of dialogue evolution;
- the configuration of hardware and software for the workstations;
- the derivation and configuration of a computer center based on the workstations; and
- the derivation and configuration of necessary products, which were not specified, always providing a correct configuration.

CAD Department Configuration	
Device classes at the workstation:	PC_1
local_processors	
display_processors	
display_terminals	configuration
printer_plotters	
graphical_input_devices	
mass_storage	
	help
	quit
Please select	

Fig. 2.57. Workstation hardware

The user interface enables:

- graphic interaction through menus and icons;
- interaction through unified layouts based on areas of information, processing, controlling, and messages (Fig. 2.57); and
- the graphic visualization and manipulation of the H/S configuration (Fig. 2.58).

One of the advantages of the Prolog graphic predicates built to support the graphic dialogue interface is their general purpose. This means that the layout, pick actions, help functions, and messages may be interactively created using an easy syntax. The whole dialogue interface was created using these predicates. In other words, by using these predicates, dialogues may be interactively generated and modified.

2.7.8 Conclusion

Many reasons have been given in the literature supporting the benefits of CA technology. The economics

$$\text{Total Design Cost}_{\text{mach}} \leq \text{Total Design Cost}_{\text{conv}}$$

is rather difficult to prove. Not only a static economic analysis, but a timely dynamic one involving environment parameters and change give an understanding of the development that has already occurred. The high investments are considered as rationalization of work.

The choice of a CAD system or of any other system demands a considerable amount of time and qualified personnel. The experience and the knowledge obtained in this phase is usually kept only by the responsible personnel and not enough

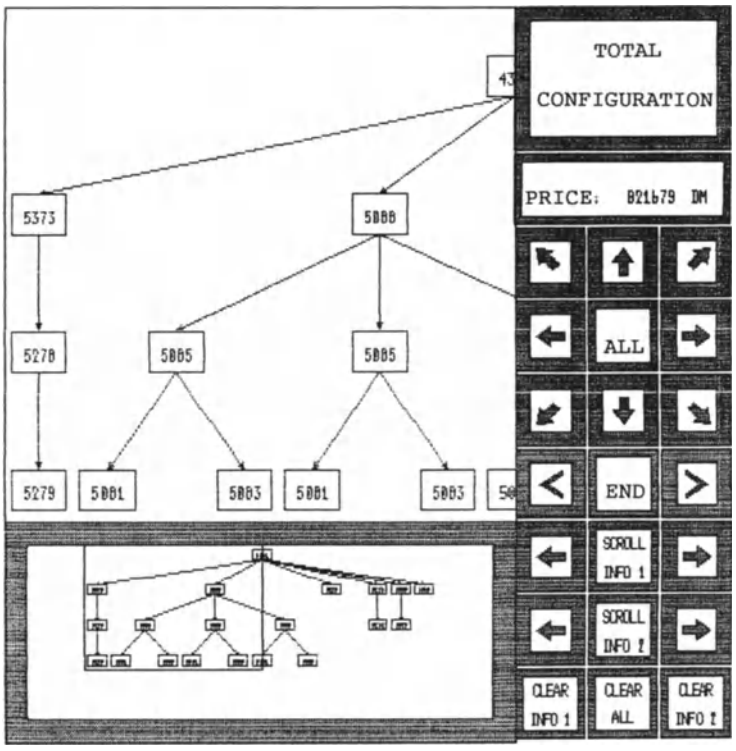


Fig. 2.58. Visualization and manipulation of a H/S configuration

methodology is applied to their maintenance and update. One could estimate that 1 man/year is necessary for this task, which in turn yields costs around US$ 100000. The implementation of systems to support this and correlated tasks is therefore in the long run profitable. It generates a methodology for knowledge maintenance and development.

2.8 Interdisciplinary Aspects of CAD

A basic characteristic of larger CAD systems is the diversity of the computer science and engineering science methods used in their implementation:

- design methodology;
- computational methods (for design, analysis, and optimization);
- data sorting and searching;
- interactive graphics;
- information handling and retrieval; and
- numerous application-oriented algorithms.

Furthermore, we have to consider at least three kinds of qualifications for various people concerned with the design, implementation, and use of CAD systems:

- computer science specialists (both hardware and software):
  qualified to develop the fundamental methods, tools, and equipment of CAD systems;
- application programmers:
  highly qualified in the design methodology and algorithms of an application area and capable of composing problem-oriented or product-oriented CAD systems from the basic components; and
- designers:
  highly qualified in their design work, and sufficiently well trained in utilizing the CAD systems' capabilities.

One of the most important aims for the coming years in this area will be to find widely accepted methods and concepts, and a common terminology. Then the effective education and training of CAD specialists will be possible in all three of these domains.

2.9 Summary

CAD was defined as the creation, analysis, and documentation of physical components, structures, or facilities. The concept and the precise form of the title CAD is due to the work of Coons early in 1958. CAD is firmly and profitably established in automobile industry, aerospace engineering, ship design, chemical engineering, nuclear engineering, and electronics; in mechanical engineering, profitable applications have been mainly in the area of analysis, with a very rapid increase in the applications of Computer Aided Drafting.

The high percentage values for drawing and information gathering activities in design work indicate not only that there is a great need for interactive computer graphics in design, but also that CAD systems should be designed as information systems with adequate support for information retrieval. On the other hand, using computer-based methods for the computational part of the design process (which accounts only for a small percentage of the work) is more likely to increase than to decrease the cost of design, but with the potential for a respectable increase in overall production profits.

2.10 Bibliography

[ALCO71] Alcock, Shearing, and Partners: GENESYS Reference Manual. Loughborough, The GENESYS Centre (1971).

[ALLA73] J. J. Allan III: Foundations of the Many Manifestations of Computer Augmented Design. In: J. Vlietstra, R. F. Wielinga (eds.): Computer Aided Design. Amsterdam, North-Holland Publ. Co. (1973).

[ANSI86] ANSI Display Management Preliminary First Draft. Technical Report ANSI X3H3.6/86-44 (1986).

[BART80] H. Barth: Grundlegende Konzepte von Methoden- und Modellbanksystemen. Angewandte Informatik 8 (1980), pp. 301–309.

[BATZ87] T. Batz: Versionsverwaltung im Datenhaltungssystem PRODAT des Systementwicklungssystems PROSYT. In: Proc. GI-Workshop „Datenbanken für Software Engineering", Dortmund (1987).

[BAUM88] P. Baumann, D. Köhler: Archiving of Versions and Configurations in a Database System for System Engineering Environments. To be published in Proc. of Int. Workshop of Software Version and Configuration Control in Grassau, Germany January (1988).

[BECK87] K. Beck: Untersuchung und Weiterentwicklung des Informationssystems CAD-System Auswahl DBCAD. TH Darmstadt, Diplomarbeit, FB Informatik, FG Graphisch-Interaktive Systeme (1987).

[BEIE76] K.P. Beier: Systemsoftware für ein integriertes schiffbautechnisches Programmsystem. Dissertation. Berlin, Techn. Univ. (1976).

[BEIE78] K.P. Beier, W. Jonas: DINAS – A Transportable Executive System for Interactive Computer Aided Design. Proc. Int. Conf. Interactive Techniques in Computer Aided Design. Bologna (1978), pp. 393–403.

[BENE79] I.D. Benest: A Review of Computer Graphics Publications. Computers & Graphics 4 (1979), pp. 95–136.

[BITT88] H. Bittner, M.J. Cote, F. Eser, D. Frantz: A User Interface Management System for Integrating Electronical and Medical CAD. 3rd IFAC Conference on Man-Machine Systems, Oulu, Finland (1988).

[BLAU80] R.E. Blauth (ed.), C. Machover: The CAD/CAM Handbook, Computervision Corporation, Bedford, Massachusetts (1980).

[BLY_86] S.A. Bly, J.K. Rosenberg: A Comparison of Tiled and Overlapping Windows. Proceedings of CHI'86 Human Factors in Computing Systems (1986), pp. 101–106.

[BRAN83] H. Brand, H. Felzmann, R. Glatz, R. Grabowski, H.H. Rubensdörffer: Qualitätsbeurteilung von CAD/CAM-Systemen, Testhandbuch Band 1, Technische Nutzwert-Analyse, SCS Scientific Control Systems GmbH (1983).

[BROW63] S. Brown, C. Drayton, B. Mittman: A Description of the APT-Language. CACM 6, 11 (1963), pp. 649–658.

[CARD82] S.K. Card: User Perceptual Mechanisms in the Search of Computer Command Menus. Proc. Human Factors in Computer Systems (1982), pp. 190–196.

[CARD84] S.K. Card, M. Pavel, J.E. Farrell: Window-based Computer Dialogues. In: B. Shackel, Human-Computer Interaction-INTERACT'84 (1984), pp. 355–359.

[CCRE87] Marktübersicht: CAD-Systeme ab 100000 DM, 1. Teil Mechanik. CAD-CAM-Report Nr. 8 (1987).

[CHAS73] S.H. Chasen: Economic Principles for Interactive Graphic Applications. AFIPS 44 (1973), pp. 613–620.

[CHAS81] S.H. Chasen: Historical Highlights of Interactive Computer Graphics. Mechanical Engineering 103 (1981) 11, pp. 32–41.

[CHIY88] H. Chiyokura: Solid Modelling with Designbase. Addison-Wesley Publ. Co (1988).

[CIM_88] Projektgruppe CIM im DIN, AK1: CAD-Schnittstellen. Bericht zur Voruntersuchung (1988).

[CLAU71] U. Claussen: Konstruieren mit Rechnern. Konstruktionsbücher, Band 29, Springer-Verlag (1971).

[COON63] S. Coons: An Outline of the Requirements for a Computer Aided Design System. AFIPS, SJCC 23 (1963), pp. 299–304.

[DADA84] P. Dadam, V. Lum, H.-D. Werner: Integration of Time Versions into a Relational Database System. Proc. 10th Int. Conf. on Very Large Databases, Mexico City (1984), pp. 509–522.

[DEDE86] A. Dede: Implementierung einer Datenbank auf dem Datenbanksystem UDS zur Auswahl von CAD-Software/Hardware aufgrund einer CAD-Klassifizierung. Diplomarbeit, FB Informatik, FG GRIS, TH Darmstadt (1986).

[DFN_86] Deutsches Forschungsnetz – DFN: Status review and future plans for graphics, modelling, and document services in DFN. DFN-Bericht Nr. 46, Berlin (1986).

[DICK78] P. Dickinson: Versatile Low-Cost Graphics Terminal 13 Designed for Ease of Use. Hewlett-Packard Journal (1978), pp. 2–6.

[DIEB76] Diebold Deutschland GmbH: Rechnerunterstütztes Entwickeln und Konstruieren in den USA. Report KFK-CAD 7, Kernforschungszentrum Karlsruhe (1976).

[DITT79] K.R. Dittrich, R. Hüber, P.C. Lockemann: Methodenbanksysteme: Ein Werkzeug zum Maßschneidern von Anwendersystemen. Informatik-Spektrum 2 (1979), pp. 194–203.

[DITT85] K.R. Dittrich, R.A. Lorie: Version Support for Engineering Database Systems. Research Report FJ4769, IBM Research Laboratory San Jose, California (1985).

[DITT86] K.R. Dittrich, W. Gotthard, P.C. Lockemann: DAMOKLES – Database System for Software Engineering Environments. Proc. IFIP Workshop on Advanced Programming Environments, Trondheim, Springer-Verlag (1986).

[EGGE81] R. Eggensberger: Design eines interaktiven didaktisch orientierten Methodenbanksystems. Angewandte Informatik 9 (1981), pp. 394–399.

[EIGN80] M. Eigner, H. Grabowski, H.-D. Habn: Auswahl und Einführung von schlüsselfertigen CAD-Systemen. FB/IE 29, Heft 3 (1980).

[ENCA72] J.L. Encarnação, W. Giloi: PRADIS – An Advanced Programming System for 3D-Display. AFIPS, SJCC 40 (1972), pp. 985–998.

[ENCA78] J.L. Encarnação, E.G. Schlechtendahl: Konzepte, Probleme und Möglichkeiten von CAD-Systemen in der industriellen Praxis. Informatik Fachberichte 16, Springer-Verlag (1978), pp. 308–325.

[ENCA79] J.L. Encarnação: Interactive Computer Graphics – The Rich and Dynamic Man-Machine Environment. In: P.A. Samet (ed.), Proc. EURO IFIP 79, London, September 1979. Amsterdam, North-Holland Publ. Co. (1979), pp. 521–529.

[ENCA81] J.L. Encarnação, O. Torres, E.A. Warman (eds.): CAD/CAM as a Basis for the Development of Technology in Developing Nations. Amsterdam, North-Holland Publ. Co. (1981).

[ENCA84] J.L. Encarnação, H.-E. Hellwig, E. Hettesheimer, W.F. Klos, S. Lewandowski, L.A. Messina, W. Poths, K. Rohmer, H. Wenz: GI/CAD-Handbuch Auswahl und Einführung von CAD-Systemen, Springer-Verlag (1984).

[ENCA85] J.L. Encarnação, Z. Markov, L.A. Messina: Models and methods for decision support systems for evaluating and choosing CAD-Systems. Proc. of IFIP W.G.5.2 Working Conference on Design Theory for CAD, October 1985, Tokyo Japan. Edited by Yoshikawa H. and Warman E.A., North-Holland (1987).

[ENDE80] G. Enderle, K.H. Bechler, F. Katz, K. Leinemann, W. Olbrich, E.G. Schlechtendahl, K. Stölting: GIPSY-Handbuch. Report KfK 2878, Kernforschungszentrum Karlsruhe (1980).

[ENDE81] G. Enderle, K.-H. Bechler, H. Grimme, W. Hieber, F. Katz: GIPSY-Handbuch Band II. Report KfK 3216, Kernforschungszentrum Karlsruhe (1981).

[ENDE84] G. Enderle: The Interface of the UIMS to the Application. Computer Graphics Forum 3 (1984), pp. 175–179.

[ENLO90] J. L. Encarnação, P. C. Lockemann (eds.): Engineering Databases, Springer-Verlag (1990).

[FOLE84a] J. D. Foley, V. L. Wallace, P. Chan: The Human factors of computer graphics interaction techniques. IEEE Computer Graphics & Applications 4, 11 (1984), pp. 13–48.

[FOLE84b] J. D. Foley, A. van Dam: Fundamentals of Interactive Computer Graphics. Addison-Wesley, Reading, Massachusetts (1984).

[FREE67] H. Freeman: An Algorithm for the Solution of the Two-Dimensional "Hidden-Line" Problem. IEEE Trans. Electr. Comput. EC 16, 6 (1967), pp. 748–790.

[FREN70] L. French, A. Teger: GOLD – A Graphical On-Line Design System. AFIPS, SJCC 40 (1970), pp. 461–470.

[FUJI85] T. Fujita, F. Kimura, T. Sata, H. Suzuki: Designing machine assembly structure using geometric constraints in product modelling. Annals of the CIRP 34, 1 (1985), pp. 169–172.

[FULT81] R. E. Fulton: Using CAD/CAM to Improve Productivity. Mechanical Engineering 103, 11 (1981), pp. 64–69.

[GING78] McLean Gingras: A study of users and designers of information systems. Center for Information Studies, Working Paper 2–79, UCLA (1978).

[GRAB79] H. Grabowski: Veränderte Arbeitsstrukturen durch CAD-Systeme. ZwF 74, 6 (1979), pp. 294–300.

[GRAB81] H. Grabowski, H. Maier: Der Konstrukteur als Programmierer. NC-Report 1–81 (1981), pp. 125–130.

[HATV77] J. Hatvany, W. M. Newman, M. A. Sabin: World survey of Computer Aided Design. Computer Aided Design 9, 2 (1977).

[HATV84] J. Hatvany: CAD-state of the art and a tentative forecast. Robotics & Computer-Integrated Manufacturing 1 (1984).

[HAYE85] P. J. Hayes, P. A. Szekely, R. A. Lerner: Design Alternatives for User Interface Management Systems Based on Experience with COUSIN. Proc. of CHI'85 Human Factors in Computing Systems (1985), pp. 169–175.

[HETE85] E. Hetesheimer: Planung von CAD-Einführungsstrategien unter Berücksichtigung organisatorischer Inhalte und Regelung der Effizienz des CAD-Einsatzes. Dissertation, Univ. Karlsruhe, Fortschritt-Berichte VDI, Reihe 1: Konstruktion/Maschinenelemente, Nr. 127, VDI Verlag (1985).

[HOPG85] F. R. A. Hopgood, D. A. Duce, E. V. C. Fielding, K. Robinson, A. S. Williams (eds.): Methodology of Window Management. Springer-Verlag (1985).

[HOPG86] F. R. A. Hopgood, et al. (eds.): Methodology of Window Management. Springer-Verlag (1986).

[HÜBN87a] W. Hübner, G. Lux-Mülders, M. Muth: THESEUS – Die Benutzungsoberfläche der UNIBASE-Softwareentwicklungsumgebung. Springer-Verlag (1987).

[HÜBN87b] W. Hübner, G. Lux-Mülders, M. Muth: Designing a System to provide Graphical User Interfaces – the THESEUS approach. In: G. Marechal (ed.) Proc. of EUROGRAPHICS'87. North-Holland, Amsterdam (1987), pp. 309–322.

[HUTC86] E. L. Hutchins, J. D. Hollan, D. A. Norman: Direct Manipulation Interfaces. In: D. A. Norman, S. W. Draper (eds.): User Centered System Design. Lawrence Erlbaum Ass. (1986), pp. 87–124.

[IKO__83] IKO Software Service GmbH (Hsg.): CAD-Kriterienkatalog. Stuttgart (1983).

[INMA78] B. Inman: The Boeing Electronic Computer Aided Design System. AFIPS 47 (1978), pp. 353–355.

[ISIS86] ISIS Engineering Report, ISSN 0175-601 X, Nomina GmbH, München (1986).

[ISO__85] International Standard Organisation (ISO): Information Processing Systems – Computer Graphics – Interfacing Techniques Graphical Kernel System (GKS) – Functional Description, ISO IS 7942. New York, ISO (1985).

[ISO__86a] International Standard Organisation (ISO): Information Processing Systems – Computer Graphics – Graphical Kernel System for Three Dimensions (GKS-3D), Functional Description, ISO/TC97/SC21 DIS 8805. New York, ISO (1986).

[ISO__86b] International Standard Organisation (ISO): Information Processing Systems – Computer Graphics – Interfacing Techniques for Dialogues with Graphical Devices (Computer Graphics Virtual Device Interface) – Functional Specification, ISO DP 9636. New York, ISO (1986).

[ISO__87a] International Standard Organisation (ISO): Information Processing Systems – Computer Graphics – A GKS shell for PHIGS-Implementations, ISO/TC97/SC21/WG2/PHIGS/66, preliminary draft. New York, ISO (1987).

[ISO__87b] International Standard Organisation (ISO): Information Processing Systems – Computer Graphics – Programmer's Hierarchical Interactive Graphics System (PHIGS), ISO/TC97/SC21 DIS 8805. New York, ISO (1987).

[ISO__87c] International Standard Organisation (ISO): Information Processing Systems – Computer Graphics – Programmer's Hierarchical Interactive Graphics System (PHIGS), ISO/TC97/SC21 DIS 9592. New York, ISO (1987).

[JACK64] E. Jacks: A Laboratory for the Study of Man-Machine Communication. AFIPS, FJCC 26, Part 1 (1964), pp. 343–350.

[JOHN86] R. J. Johnson: Solid modelling: a state of the art report. North Holland, second edition, Amsterdam (1986).

[KATZ84] R. H. Katz, T. J. Lehmann: Database Support for Versions and Alternatives of Large Design Files. IEEE Transactions on Software Engineering 10 (1984), pp. 191–200.

[KATZ85] R. H. Katz: Information Management for Engineering Design. Springer-Verlag (1985).

[KIMU87] F. Kimura, Y. Yamaguchi, P. ten Hagen: Interaction Management in CAD Systems with History Mechanisms. In: G. Marechal (ed.) Proc. of EUROGRAPHICS'87, North Holland (1987).

[KLEM88] K. Klement, H. Nowacki: Exchange of model presentation information between CAD systems, Computers & Graphics 12, 2 (1988).

[KOFO66] J. Koford, P. Strickland, G. Sporzynski, E. Hubacher: Using Graphic Data Processing Systems to Design Artwork for Manufacturing Hybrid Integrated Circuits. AFIPS, FJCC 29 (1966), pp. 229–246.

[KRAU84] F.-L. Krause, G. Spur: CAD-Technik, Lehr- und Arbeitsbuch für die Rechnerunterstützung in Konstruktion und Arbeitsplanung. Carl Hanser Verlag (1984).

[KRÖM89] D. Krömker, H. Steusloff, H. P. Subel (Hrsg.): PRODIA und PRODAT: Dialog- und Datenbankschnittstellen für Systementwurfswerkzeuge, Springer-Verlag (1989).

[LAST88] G. L. Lastra, J. L. Encarnação, A. A. G. Requicha: Applications of Computers in Engineering Design, Manufacturing and Management. North Holland (1988).

[LEMP85] H. Lempert: Nutzwertanalyse in Ifprolog. Studienarbeit, FB Informatik, FG GRIS, TH Darmstadt (1985).

[LORI83] R. A. Lorie, W. Plouffle: Complex Objects and Their Use in Design Transactions. Proc. Database Week: Engineering Design Applications (IEEE) (1983), pp. 115–121.

[LUXM88] G. Lux-Mülders, W. Hübner, M. Muth, U. Brand, T. Nöthing: An Approach for the Integration of General Purpose Graphics Systems and Window Management. The Visual Computer, Int. Journal of Computer Graphics 4, 3 (1988).

[MESS86] L. A. Messina, M. J. Prospero: Towards the construction of graphical interfaces on the basis of geometric models. Proc. of EUROGRAPHICS'86, Lisbon, North-Holland (1986).

[MESS88a] L. A. Messina: Verfahren zur Auswahl und Evaluierung von CAD-Systemen. Dissertation, FB Information, FG GRIS, TH Darmstadt (1988).

[MESS88b] L. A. Messina: A teachware concept for education in CAD, proposed to the Int. Conf. Computer-Assisted Learning, Dallas (1989).

[NEES78] G. Nees: Struktur und Organisationsformen von CAD-Systemen aus der bisherigen Praxis. In: J. L. Encarnação (ed.), Proc. CAD-Fachgespräch bei der GI-Jahrestagung 1978, Berlin. Report GRIS 78-3, TH Darmstadt, FB Informatik, FG GRIS (1978).

[NELS72] M. F. Nelson: ICES STRUDL II. In: Three-Dimensional Continuum Computer Programs for Structural Analysis, New York, ASME (1972), pp. 23–24.

[NEUM83] T. Neumann: Konzepte zur Erweiterung von Datenbanksystemen für die Unterstützung von CAD/CAM-Anwendungen. Dissertation, FB Informatik, FG GRIS, TH Darmstadt (1983).

[NINK65] W. Ninke: GRAPHIC 1 A Remote Graphical Display Console System. AFIPS, FJCC 22, Part 1 (1965), pp. 839–846.

[NOLL87] S. Noll et al.: PHI-GKS, Functional Description. TH Darmstadt, FB Informatik, FG GRIS (1987).

[NOLT76] H. Noltemeier: Modelle – Methoden – Daten. In: H. Noltemeier (ed.), Computergestützte Plannungssysteme, Würzburg, Physica-Verlag (1976), pp. 247–253.

[NOTE73] R. Notestine: Graphics and Computer Aided Design in Aerospace. AFIPS 42 (1973), pp. 629–633.

[OLSE87] D. R. Olsen (ed.): ACM SIGGAPH Workshop on Software Tools for User Interface Management. Computer Graphics 21, 2 (1987), pp. 71–147.

[PAHL78] P. J. Pahl, L. Beilschmidt: Informationssystem Technik. Programmierhandbuch. Report KfK CAD-81, Kernforschungszentrum Karlsruhe (1978).

[PEAS52] W. Pease: An Automatic Machine Tool. Scientific American 187, 3 (1952), pp. 101–115.

[PFAF85] G. Pfaff (ed.): User Interface Management Systems. Springer-Verlag (1985).

[PHBR88] PHIGS BR Functional Description, ISO/IEC JTC1/SC24 N224 (1988).

[PILK74] W. Pilkey, K. Saczalski, H. Schaeffer: Structural Mechanics Computer Programs, Surveys, Assessments, and Availability. Charlottesville, Univ. Virginia Press (1974).

[POTH86] W. Poths: Stand und Möglichkeiten von CAD/CAM im Maschinenbau. In: J. L. Encarnação (ed.), Proc. der Aktuellen Themen der Graphischen Datenverarbeitung. Springer-Verlag (1986).

[PROD88] T. Batz, P. Baumann, K.-G. Höft, D. Köhler, D. Krömker, H.-P. Subel: PRODAT – Das PROSYT-Datenbanksystem. In D. Krömker, H. Steusloff, H. P. Subel (Hrsg.): PRODIA und PRODAT: Dialog- und Datenbankschnittstellen für Systementwurfswerkzeuge, Springer-Verlag (1989).

[RCAD88] Referenzmodell für CAD-Systeme 1. Entwurf. Gesellschaft für Informatik e.V. (D) (1988).

[REIN85] D. Reinking: Quantifizierung der Produktivitätssteigerung beim Einsatz von CAD-Systemen im Konstruktionsprozeß. Fortschritt-Berichte VDI, Reihe 10: Angewandte Informatik. Dissertation Uni Karlsruhe, VDI-Verlag (1985).

[RENO78] T. Reno: General Motors Network Station a Low Cost Graphics System for Body Tooling. AFIPS 47 (1978), pp. 337–341.

[REQU82] A. A. G. Requicha, H. B. Voelcker: Solid Modeling: A Historical Summary and Contemporary Assessment. IEEE Computer Graphics and Applications, 2 (1982), pp. 9–24.

[RIED86] T. Riedel-Heine, D. Köhler: A Version Management System for Design Environments. Proc. EUROGRAPHICS'86, Lisbon, North-Holland (1986).

[ROSS76] Ross, D. T.: ICES System Design. Cambridg, MIT Press (1976).

[[SÄNG86] C. Sänger: Graphisch-Interaktive Schnittstelle zur Behandlung von Rechner-Architekturen. Diplomarbeit, FB Informatik, FG GRIS, TH Darmstadt (1986).

[SÄNG87] C. Sänger: Beschreibung der Prädikate der graphisch-interaktiven Dialogschnittstelle und der allgemeinen Prädikate des graphischen Dialog-Werkzeuges (GDW__GDDM). Ordner 2 IBM- DOK (1987).

[SCHE86] R. W. Scheifler, J. Gettys: The X Window System. ACM Transactions on Graphics 5, 2 (1986), pp. 79–109.

[SCHI77] B. Schips: Ein Beitrag zum Thema „Methodenbanken". Angewandte Informatik 11 (1977), pp. 465–470.

[SCHL74] E. G. Schlechtendahl: Comparison of Integrated Systems for CAD. Int. Conf. Computer Aided Design, IEE Conf. Publ. 111, Southampton (1974), pp. 111–116.

[SCHL76] E. G. Schlechtendahl: Grundzüge des integrierten Programmsystems REGENT. Angewandte Informatik 11 (1976), pp. 490–496.

[SCHL81a] E. G. Schlechtendahl: Der Systemkern REGENT als Basis zur Entwicklung technisch-wissenschaftlicher Programmsysteme. 9. Int. Kongress über die Anwendungen der Mathematik in den Ingenieurwissenschaften, Weimar, 1981. Hochschule für Architektur und Bauwesen, Heft 1 (1981), pp. 89–92.

[SCHL81b] E. G. Schlechtendahl, K. H. Bechler, G. Enderle, K. Leinemann, W. Olbrich: REGENT-Handbuch. Report KfK 2666 (KfK-CAD 71), Kernforschungszentrum Karlsruhe (1981).

[SCHL82] E. G. Schlechtendahl, G. Enderle: Ansätze zu Methodenbanken im technisch-wissenschaftlichen Bereich. Angewandte Informatik 8 (1982), pp. 399–409.

[SCHÖ86] J. Schönhut: Are PHIGS and GKS Necessarily Incompatible? IEEE Computer Graphics & Applications (1986).

[SCHR86] H. Schröder: Ein Graphen-Editor in Ifprolog. Studienarbeit, FB Informatik, FG GRIS, TH Darmstadt (1986).

[SCOT81] D. J. Scott: Computer Aided Graphics: Determining the System Size, Estimating Costs and Savings. Computers in Industry 2 (1981).

[SELL84] U. Sellmer, B. Schmid: Beispiel zur Bestimmung der wirtschaftlichen Anwendung von CAD-Systemen. CAD/CAM Sonderteil in Hanser-Fachzeitschriften (1984).

[SHNE82] B. Shneiderman: The future of interactive systems and the emergence of direct manipulation. Behavior and Information Technology, 1 (1982), pp. 237–256.

[SHNE83] B. Shneiderman: Direct Manipulation: A Step Beyond Programming Languages. IEEE Computer 16, 8 (1983), pp. 57–69.

[SHNE87] B. Shneiderman: Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison Wesley (1987).

[SPUR81] G. Spur, F.-L. Krause: Aufbau und Einordnung von CAD-Systemen. VDI-Berichte 413 (1981), pp. 1–18.

[SPUR84] G. Spur, F.-L. Krause: CAD-Technik, Lehr- und Arbeitsbuch für die Rechnerunterstützung in Konstruktion und Arbeitsplanung. Carl Hanser Verlag (1984).

[STEP88] Project Plan for ISO TC 184/SC4/WG1 (STEP), N-213; Owner: J. A. Weiss, McDonnell Douglas Corp., P.O. Box 516, USA – St. Louis (1988).

[STRA86] D. H. Straayer: Setting Standards. Computer Graphics World, Nov 86, pp. 73–78.

[SUTH63] I. E. Sutherland: Sketchpad: A Man-Machine Graphical Communication System. AFIPS, SJCC 23 (1963), pp. 329–346.

[THOM83] J. J. Thomas, G. H. Hamlin et al.: Graphical Input Interaction Technique Workshop Summary. Computer Graphics 17, 1 (1983), pp. 5–30.

[THOM88] M. Thomas: Menüsystem und Nutzwertanalyse in Turbo Prolog. Studienarbeit, FB Informatik, FG GRIS, TH Darmstadt (1988).

[TICH85] W. F. Tichy: RCS – A System for Version Control. Software – Practice and Experience 15, 7 (1985), pp. 637–654.

[TOMI86] T. Tomiyama: Integrated Data Description Schema – Issues on Representation of Knowledge for CAD Systems. Winter School on Conceptual Modelling, Visegrad (1986).

[VDI_87] VDI-Richtlinie 2216, Einführungsstrategie und Wirtschaftlichkeit von CAD-Systemen (1987).

[VERN84] F. B. Vernadet: A commented and indexed bibliography on data structuring and data management in CAD/CAM: 1970 to mid-1983. National Research Council Canada, Division of Electrical Engineering, ERB-956, NRCC No. 23373 (1984).

[WARM78] E. A. Warman: CAD/CAM Management and Economics, SIGGRAPH'78.

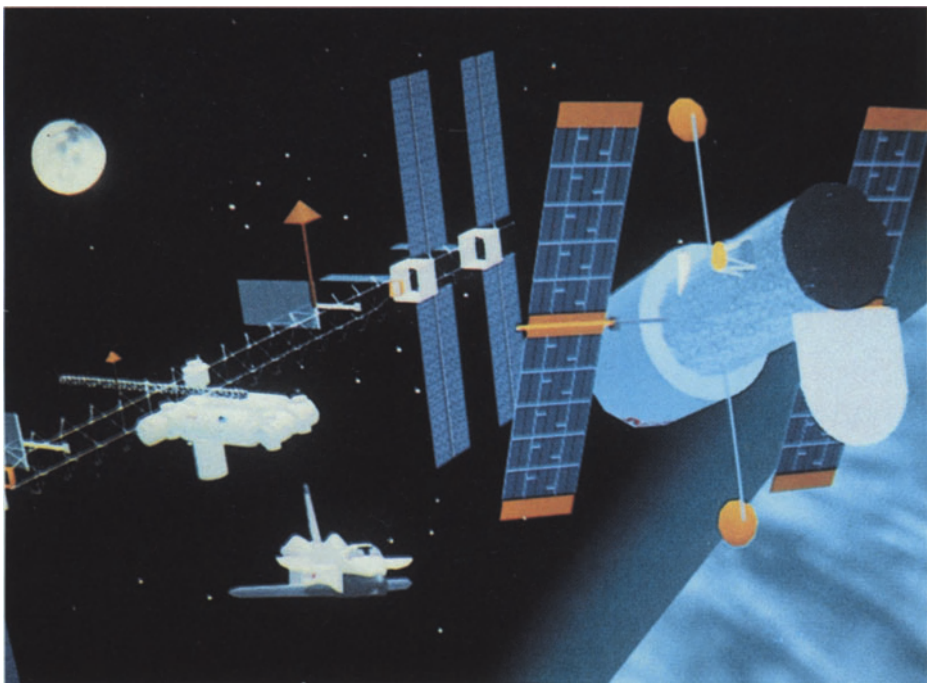
[WARM87] E. A. Warman, H. Yoshikawa (ed.): Design Theory for CAD. IFIP WG 5.2, North Holland (1987).

[WEDE86] H. Wedekind: Systemanalyse – Die Entwicklung von Anwendungssystemen für DV-Anlagen. Carl Hanser Verlag, München (1973).

[WILD86] H. Wildemann: CAD/CAM als Instrument der Wettbewerbsstrategie. ZWF 81, 10 (1986), Carl Hanser Verlag (1986).

[WILL86] A. S. Williams: A comparison of some window managers. In F. R. A. Hopgood et al., Methodology of Window Management, Springer-Verlag (1986).

3 The Process Aspect of CAD



Space flight scenery
(courtesy of Evans & Sutherland, Salt Lake City, USA)

3.1 Modeling of the Design Process

3.1.1 A Crude Model of the Design Process

CAD provides computer support for the design process. Hence, if we want to talk about CAD, we must first talk about the process of design; that is, we must construct at least a crude model of the design process. The problem, however, is that design processes are quite different from one another, depending on the product (a bicycle versus a nuclear power plant), on the company's size and organization (a large architectural engineering firm versus a specialized engineering bureau), and on the type of design (the restatement of a basically fixed design versus the completely original design of a new product). The purpose of establishing modeling concepts for the design process is to provide the systems analyst with a means of describing the global system into which a CAD system must fit (and to set forth a basis for the terminology used in the subsequent chapters). Both the designer of a CAD system and its potential user must be able to agree on a description of the interfaces of the computer-aided part of the design process with the remaining part of the process. Such interfaces will be easy to describe if the design process can be adequately represented by a sequence or chain of actions where each action passes its results on to its successor. We will see, however, that the design process is far more complex, and that neither a chain nor a tree is sufficient to represent its essential characteristics, even though it may sometimes look like a chain or a tree in certain respects. In view of the complexity of the design process, it is perhaps not very surprising that there have been many attempts to establish a systematic description of design, resulting in a number of proposals, which are similar, but without complete agreement in detail (see [PAHL77], for example). An important step in the direction of establishing a common view is achieved with VDI guideline 2221 [VDI__69]. The complex structure of the design process will have to be reflected in the structure of CAD systems, if such systems are to support the design process as a whole and not only isolated parts of it.

We will now set up a very general (and hence rather simple) model of a "typical" design process. Using the terminology of Grotenhuis and van den Broek [GROT76], this is an intuitive conceptual model. Figure 3.1 is a first attempt at such a model. The basic assumptions of this model are:

- the design goal is fixed (at least temporarily),
- a certain kind of knowledge is required to construct the design, and
- the design process produces information (the "design"), which in one way or another can be documented and used for production.

That which is labelled "design" in Fig. 3.1 is not yet the product itself. It is a model of the product which allows us to talk about the product before it exists.

Using Nijssens approach [NIJS76], such a model may be deduced from the real world by a sequence of two operations: perception and selection (see Fig. 3.2). We perceive only a subset of reality: the perceptible reality. The subjectivity of this step in modeling is emphasized by the definition of a "system", which we quote from [NYGA80]:

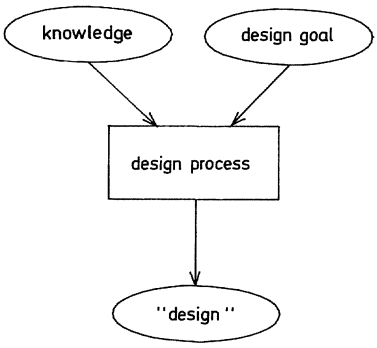


Fig. 3.1. A crude model of the design process

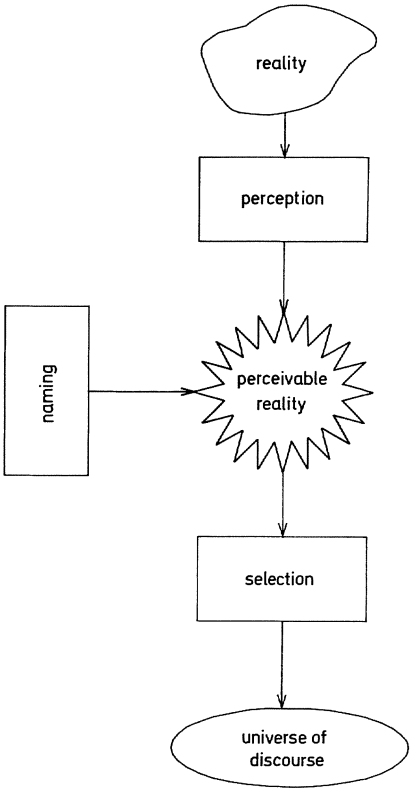


Fig. 3.2. Modeling of the real world

– "A *system* is a part of the world, which a *person* (or group of persons) – during some time and for some reason – chooses to regard as the whole consisting of *components*, each component characterized by *properties* which are selected as being relevant and by *actions* which relate to these properties and to other components."

We can talk about perceptible reality by assigning names to its entities and the attributes of the entities. One of these entities is the anticipated product. In general,

however, we are not interested in all aspects of perceptible reality; we select those aspects which are relevant in some pragmatic way. This process of selection generates what Nijssen calls the "universe of discourse". It is not a formal model but an "intuitive conceptual model" in the sense of Grotenhuis and van den Broek.

3.1.2 A More Refined Model of the Design Process

Our first crude model of the design process, however, does not yet reflect important characteristics of many design processes. We will modify the model to account for the following points:

- The design process is not self-contained. The design process is always embedded in another process (called its environment), and is initiated and controlled by a higher level process (the "company" process or "world" process). This aspect has been pointed out in particular in [WAEC69].
  When the design process is started, it will be accompanied by the submission of the design specification to the designer. This specification is not identical to the final goal, but is rather a formulation of the goal. It is possible that due to misinterpretations, or to incomplete or incorrect formulation in the design specification, the goal cannot be reached. For long-running and large-scale design projects the specification cannot be assumed to remain constant. The specification may not only be developing in more detail, but may actually be changing. As an example, new environmental protection laws taking effect during the design period of a chemical plant will affect the specification. The design process must involve precautions to accommodate such specification changes (at least to a certain extent). The design specification may be influenced not only by such external forces. In the course of the design certain aspects of the specification may turn out to be undesirable with respect to the design goal. For instance, tight tolerances may cause very high costs. A reevaluation at a higher level might lead to less stringent specification requirements and thus to a better design. In order to allow for such corrective measures, the model of the design process should include the presentation of preliminary design achievements to the higher-level stages of the overall process.
- The design process is most often iterative. Decisions on certain product characteristics are made in a heuristic way at an early design stage on the basis of incomplete knowledge about their consequences with respect to the design goal. We call this the "synthesis" part of the design process. As a result, the "design" must be analyzed and evaluated in the light of the design specification. In the software-oriented world these activities are associated with the terms "validation" or "verification". If the goal is not met, the design decisions must be appropriately corrected.

Figure 3.3 reflects these points. It shows that the design process is of a control-loop type. The inner loop operates on a fixed design specification and consists mainly of the operations "synthesis", "analysis", and "evaluation". The deviation of the preliminary design from the specification is fed into the synthesis operation. A second loop is closed not within the design process itself but rather in the higher-level process.

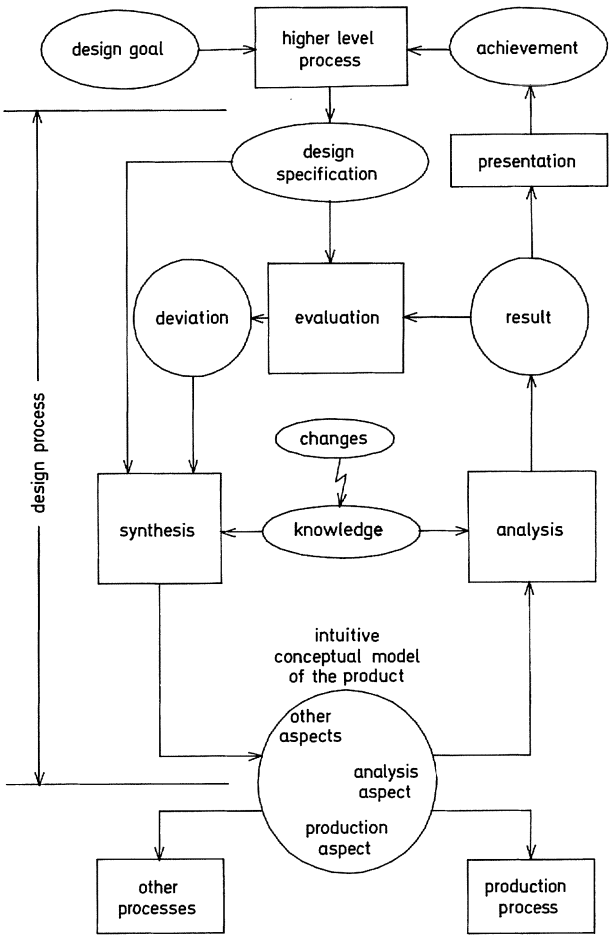


Fig. 3.3. A refined model of the design process

Thus, the design specification is (or at least may be) a moving target. For the sake of economy in the design process it is essential that appropriate specification methods be used to minimize the rate of change of the target.

Figure 3.3 illustrates additional important aspects. The design process does not only generate the information needed for production of the product. The conceptual model must also represent all the information necessary for the analysis part of the design process, and for all the other processes which may follow. Testing, marketing, and maintenance, for instance, require information produced in the design process. The conceptual model, which is generated by the synthesis and checked against the specification via analysis and evaluation, thus becomes the central point of all subsequent operations. It is for this reason that literature on CAD emphasizes the importance of the data base and the particular requirements posed by CAD applications [GRAB79].

Another point reflected in Fig. 3.3 is the fact that the available knowledge is not necessarily fixed. As with the potential changes in the specification this is particularly true for long and large projects, such as a chemical or nuclear plant, and for anything which is being designed for the first time. Design and production schedules for large projects would be intolerable if the start of the design process had to be delayed until all the required knowledge was collected. The knowledge used for the design of a particular product – like the design specification – is the result of another process. This other process is necessary to provide the resources for successful execution of the design process; knowledge is one of these resources. Continuous improvement of design methods is especially important for CAD, and provisions must be made to incorporate such improvements in the design process.

Not only the set point (the specification) of the design process control loop, but also the resources (the knowledge) may vary during the process. Using this analogy, it is obvious that the lifetime of the specifications and of the knowledge should be large compared to the cycle time in the loop from synthesis via analysis and evaluation back into synthesis; otherwise a lot of spurious and costly transients will occur before the design reaches a new stable situation. With respect to the knowledge we will need to consider this point in connection with the problems of introducing CAD in industry. The introduction of such innovative techniques is restricted by the requirement that it must be gradual enough not to conflict with the current design processes.

There are some aspects of the design process which are not shown in Fig. 3.3:

- Every process has not only a functional aspect (which is illustrated) but also a resource aspect. The process can be executed only if the required resources are available and in a suitable state. We will deal with this aspect in more detail in Sect. 3.2.4. Resources in this context may include the designer, paper, pocket calculators, computers, time, money, etc. In addition, knowledge about facts and methods may be interpreted as one of the resources.
- The design process may itself create other (dependent) design processes by specifying the design for a component of the whole product (the design of the control system of a power plant may serve as an example). Synchronization of these dependent processes and allocation of resources to them are part of the original design process. The interaction between two environment processes and the corresponding design processes is schematically shown in Fig. 3.4.
- Furthermore, the design process for a particular product does not stand alone. It is executed in the environment of other design processes (for similar or completely different products) within the organization. All these individual design processes are embedded in a "company" process which coordinates design with manufacturing, marketing, etc., to achieve the company goals. The different processes must be synchronized and supplied with resources by the "company process".

We will now take a closer look at the knowledge. Knowledge is mostly a set of rules such as

- when you recognize situation $a \rightarrow$ try a certain refinement of the model;
- when you recognize situation $b \rightarrow$ analyze and evaluate a property of the model;
- when you recognize situation $c \rightarrow$ correct part of the model.

These types of rules may be associated with the main parts of the design process: synthesis, evaluation, and analysis. This distinction is clearly pointed out by Suss-

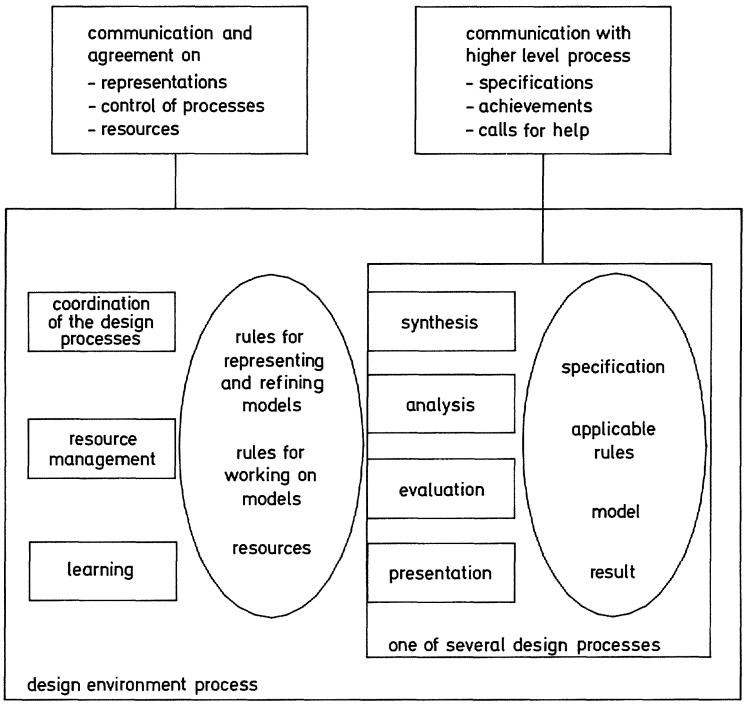


Fig. 3.4. Design process and design environment

mann [SUSS78]. Synthesis and analysis are not merely inversions of each other. Synthesis is an attempt to refine the model in such a way that the subsequent analysis will be likely to produce a satisfactory result. Such attempts may fail, which will become evident in the evaluation phase of the process. Upon failure, part of the model refinement will have to be redone. Backtracking will be necessary to find out which refinement steps were responsible for the mismatch between specification and result. If no satisfactory solution is found within the limits of the available resources (time, money, etc.) the process will have to call upon the higher-level process for help. It is interesting to note that in computer science the concept of "refinement" has become a key issue. This reflects the fact that designing a program and designing any other product are quite similar tasks on a basic level. The notion of refinement might also be expressed as a selection of a subset of all known rules, to be added to the set of those rules which are already being considered applicable. Analysis, as opposed to synthesis, is the application of previously selected rules. Thus we may associate synthesis, analysis, and evaluation with the following rules:

- *Synthesis:*
  - when you recognize a certain situation → include a certain subset of all known rules to the set of applicable rules;
- *Analysis:*
  - apply the set of applicable rules;

- *Evaluation:*
  - if result satisfies specification then finish; otherwise remove from the set of applicable rules those which are the probable reason for mismatch or call for help.

Note that the rules in the various phases obviously belong to different levels or types of rules. The objects to which the analysis rules apply may be considered as the primitives, while the synthesis and evaluation rules operate on sets of rules. It is probably this multilevel sense of rules (pointed out in [HOFS80]) which makes it so difficult to introduce formal methods not only into analysis but into the other parts of design as well. Synthesis, in particular, requires "intelligence". We quote Hofstadter [HOFS80] in order to illustrate what we mean by this term:

"... essential abilities for intelligence are certainly:

- to respond to situations very flexibly;
- to take advantage of fortuitous circumstances;
- to make sense out of ambiguous or contradictory messages;
- to recognize the relative importance of different elements of a situation;
- to find similarities between situations despite differences which may separate them;
- to draw distinctions between situations despite similarities which may link them;
- to synthesize new concepts by taking old concepts and putting them together in new ways;
- to come up with ideas which are novel."

With the separation we have now introduced between the set of applicable rules and the set of known rules, we can easily express what is meant by *changes in the knowledge* (which we call "learning" in accordance with [WAEC69]) and by *specification*:

- *Learning:*
  add new rules to the set of known rules;
- *Specification:*
  specify the rules which have to be applied in any given case.

We have seen that the whole design process can indeed be formulated in terms of rules. Such a unified approach to design, and computer-aided design in particular, is taken if artificial intelligence and pattern recognition methods are applied [LATO78]. A human, as an information processor, can work with rules immediately. He is particularly suited for "recognition of situations", which is generally a tough job for computers. Attempts have been made to provide systems with the capability to recognize situations and to choose properly among a potentially applicable set of rules. Such systems are called "expert systems" [LUML82]. However, except for a few famous applications, they are still in an experimental stage.

3.1.3 Design Processes and Design Environments

The aspects of learning and environment introduced in the previous paragraph lead to a further refinement of our design process model, which is represented in Fig. 3.4.

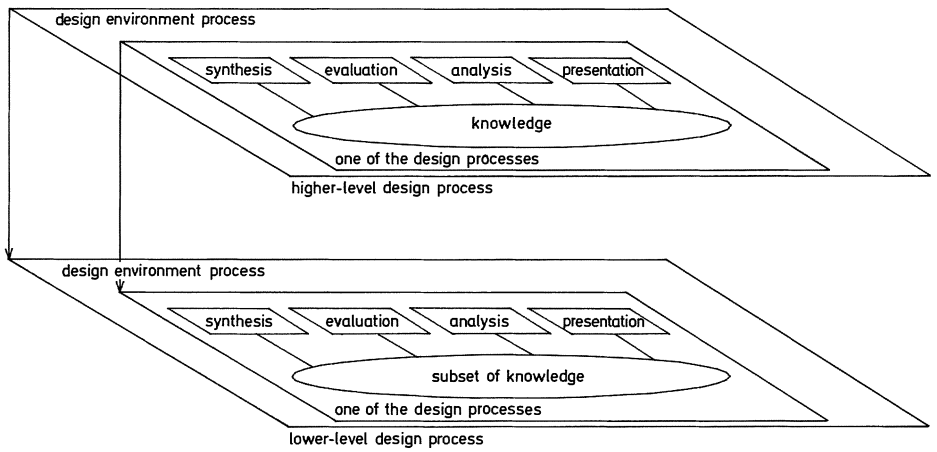


Fig. 3.5. Cooperation of design processes and environment processes

Here we do not show the control-loop relationships between the various tasks (which were the key aspect in Fig. 3.3). We simply show that synthesis, analysis, evaluation, and representation all operate on the same set of information which we call the "knowledge" associated with the design process for a given product. Several such design processes potentially using the same basic knowledge may be going on in parallel within a particular design environment. The design environment is again considered as a process with the following tasks:

- it is the recipient of requests (coming from other processes) for doing some design work, or in other words for the creation of a new design task of the particular type it can perform;
- it must agree with these other processes about the methods for representing design specifications and achievements, and for the creation, control, and termination of the newly created design tasks;
- it creates and coordinates design tasks;
- it manages resources in order to make the necessary resources available for the tasks. It attempts to avoid resource allocation conflicts, and to improve the efficiency of resource usage;
- it is responsible for providing, maintaining, and improving the knowledge (which is a particular resource).

The design environment is generally taken for granted, simply because it exists in all organisations doing design work. Moreover, the above-mentioned functions of the design environment seem to be trivial enough that one need not talk about them. The situation, however, is quite different if we consider the computer as part of the design process. In fact, using the computer one has to define the representation of specification, achievement, and so forth, very explicitly. Such agreements already exist in human-based design due to a long development and learning process.

Figure 3.5 illustrates the hierarchical cooperation of a design process (for a product) with one or more subordinate processes (for various parts of the product).

The need for communication of design information between the design processes in separate design environments produces problems whose difficulties were not appreciated properly until the early 1980s (see Chap. 7).

3.1.4 Differences Between Conventional Design and CAD

Information processing by a human does not require a formal representation, while computers can process information only if it is represented in some formal way. The question of whether all information (design specification and knowledge) can be completely formalized is perhaps a philosophically interesting speculation.

But in any case the development of a formal language for representing the information is a task which in itself consumes resources. People, time, and money are needed for this task. For this reason, in the typical case, only part of the design goal and only part of the knowledge will be represented in a formal way.

We must be aware of the essential difference between science and engineering. In science, the question of "cost" in the general sense is of secondary interest. In engineering, however, economy is a priority. Therefore, in CAD we must consider economy as important as any other aspect.

As a consequence, complete design by computer will be possible only in exceptional cases. Computer-aided design, however, will be most successful if a good synergetic cooperation between the designer (or more often, designers) and the computer (or sometimes computers) is achieved.

Another important difference between conventional design and CAD (besides the need for formal representations) has been mentioned earlier: the need to establish the environment in which the design process can work. Limitations in computer capacity (memory, disks, graphic display units, or processor power) are part of the environment. Even if we could conceive an "ideal" CAD system for certain design processes in terms of functional aspects, we cannot realize it because we have to cope with the environment. The most critical computer limitations appear to be:

- the inability of computers (or their programs) to "recognize situations";
- their inability to work with rules. Instead, computer programs require rules to be cast into an *algorithmic* form which has most often to be developed;
- their unsatisfactory efficiency in handling model changes within the synthesis task (as compared with their excellent capabilities for working on fixed models within the analysis task).

These deficiencies may become less important if artificial intelligence methods are more widely introduced into CAD [WARM78]. Currently, however, CAD methods generally call for the operation of the two main design activities with:

- synthesis preferably associated with human designers; and
- analysis preferably associated with the computer.

3.1.5 A Network Model of the Design Process

In the previous paragraphs we discussed a hierarchical structure of design environment processes; a design process could generate subordinate processes only in an envi-

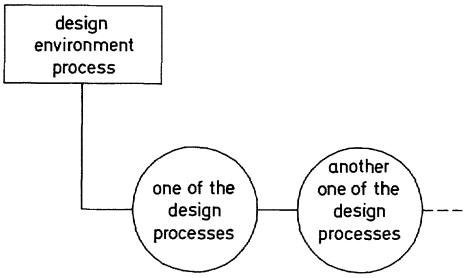


Fig. 3.6. An alternative representation of environment and design processes

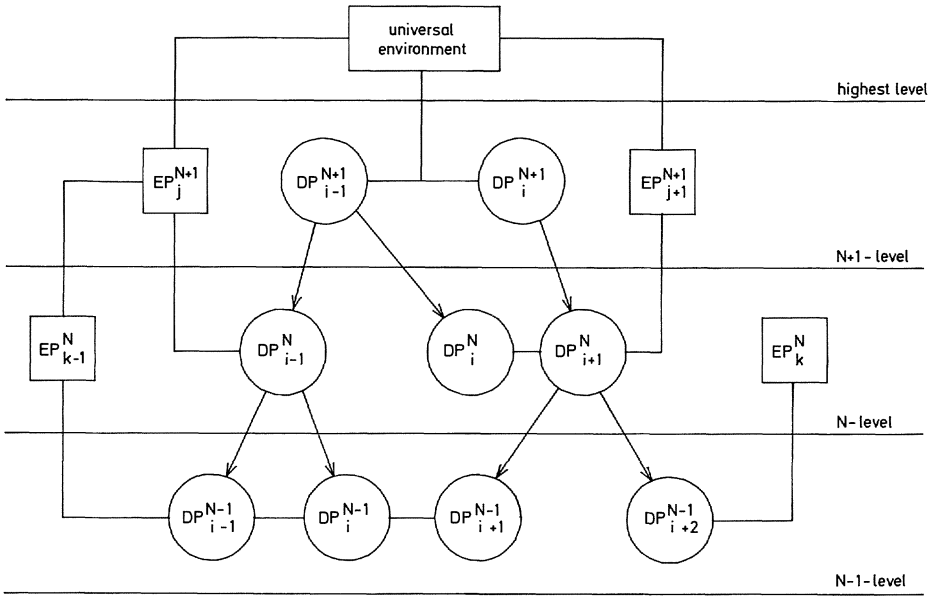


Fig. 3.7. A network model of the design process

ronment subordinate to its own. This, however, is only a special case of the more general situation. Any design process may contact any design environment process and request the creation of a new subordinate process. In order to represent these more general situations in a graphical representation, we replace the schema of Fig. 3.4 by Fig. 3.6. The line connecting the environment process to the individual design process indicates the same 'belongs_to' relation as the embedding used in Figs. 3.4 and 3.5. With this graphical schema we are able to represent a network of processes within a structure of levels as illustrated in Fig. 3.7. This schema has been influenced by proposals for the architecture of so-called "open systems" [GIES85]. Open systems are systems which permit processes to establish communications with other processes on the same level, while using facilities of a lower level. Networks of processes are also discussed in [MIS_81].

In Fig. 3.7 the design process DP_{i-1}^{N+1} (e.g., design of a vehicle) has created two subprocesses DP_{i-1}^{N} and DP_{i}^{N}, which now do work for DP_{i-1}^{N+1}.

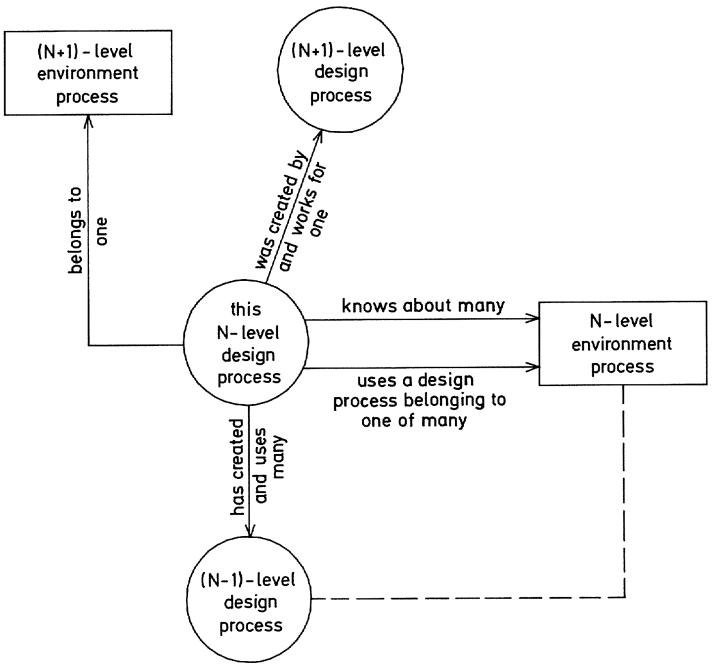


Fig. 3.8. The elementary building block of design processes

These two subprocesses are of a different type, as indicated by their respective environment processes (one perhaps being the overall design and the other being the shape design of the vehicle, for example). Parallel to process DP_{i-1}^{N+1} another design process DP_i^{N+1} is being executed in the same universal environment. Note that in this particular example both (N+1)-level processes require subprocesses at the (N−1)-level of the same type (namely of the type provided by the environment process EP_{k-1}^N). Thus, while on the N-level process DP_{i-1}^N does not necessarily know of the existence of the other process DP_i^N, their subprocesses may create conflicts in the use of the resources required on the (N−1)-level, which have to be resolved by the environment process EP_{k-1}^N.

So far we have not dealt with the problem of the creation of subprocesses: how can process DP_{i+1}^N create its subprocess DP_{i+2}^{N-1}, for instance? A strict interpretation of the schematic representation in Fig. 3.7 would mean that process DP_{i+1}^N would have to send a corresponding request either to DP_i^{N+1} or to its environment EP_{j+1}^{N+1}. If these processes cannot satisfy the request because they do not know of the existence of the environment EP_k^N, the request would first have to be passed upwards in the hierarchy of process levels until a process is reached to which both EP_k^N and the requesting process DP_{i+1}^N belong, This, however, may reduce the efficiency of all processes considerably. For this reason, the individual design processes usually have access to knowledge about environment processes on the same level. In our example, DP_{i+1}^N would probably know about the existence of the environment process EP_k^N and its capabilities in order to request the creation of subprocess DP_{i+2}^{N-1} directly.

Such knowledge was passed to the processes when they were created. Part of this knowledge is:

- knowledge about the capabilities of the environment processes; and
- knowledge of how to address these environment processes, and how to communicate with them.

Thus DP^N_{i+1} may know directly about EP^N_k. It would then request directly from EP^N_k the creation of a subordinate process DP^{N-1}_{i+2}, without the need for communication up and down the hierarchy. This concept allows us to model a design process and its interfaces without considering the totality of all processes. We may simply look at one process and its interface to other processes. The schema illustrated in Fig. 3.8 may be considered as a model building block for constructing small or large networks of design processes.

3.2 CAD Processes

3.2.1 Design Process and CAD Process

Using the constructs derived in the preceding paragraphs, we are now able to introduce CAD into the model. A design process, having identified and specified a certain subtask, may want to create a subordinate design process. If this subordinate design process operates in a computer environment, we call it a CAD process.

The general schema as sketched in Fig. 3.9 is, however, far too abstract to be useful. Let us consider two extreme cases which occur quite often in practice:

- the application of CAD systems on a large central computer in batch mode; and
- the use of a dedicated CAD system in the environment of the design office in an interactive mode.

In the first case, as illustrated in Fig. 3.10, the computer environment is not immediately suitable for CAD application. Instead, the general purpose computer (or,

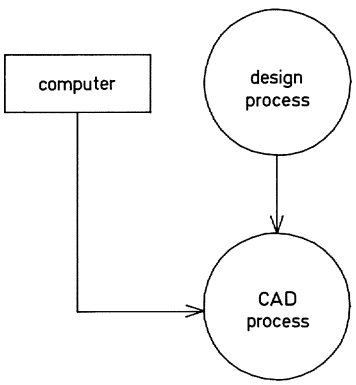


Fig. 3.9. A primitive CAD model

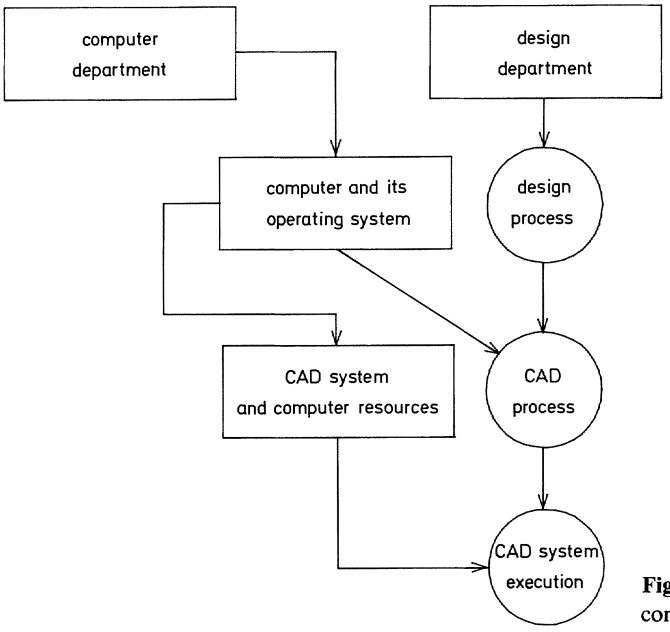


Fig. 3.10. CAD on a central computer

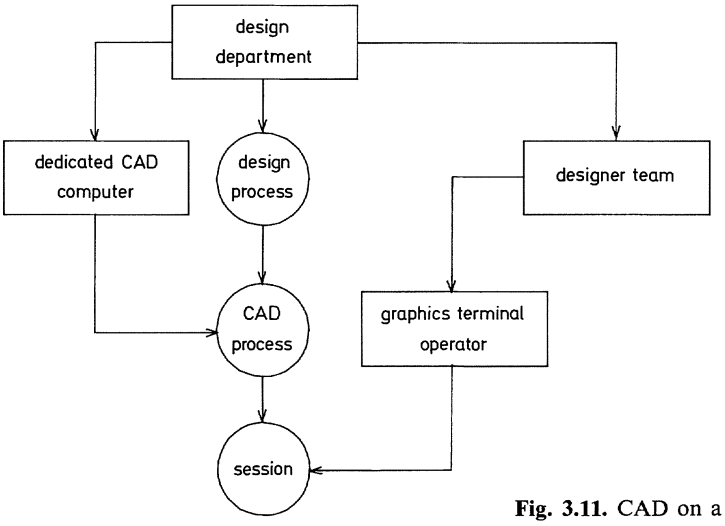


Fig. 3.11. CAD on a dedicated computer

more precisely, its operating system) must first be addressed in the language of the operating system, and requested to establish the appropriate environment: this means to make available the necessary programs, data files, and communication facilities which, collectively, we call a CAD system. Each CAD application requires not only knowledge about the CAD system itself – its capabilities and its means of communication – but also requires a purely computer-oriented job training; this fact is a particular burden of this type of CAD environment.

A much easier way to establish a CAD process is shown in Fig. 3.11. A specially programmed or "dedicated" computer within the design environment presents the CAD system in usable form to the designer, immediately or with only a few general commands required. The designer can then execute his CAD task without the problems imposed by the many other processes running on a large general-purpose computer. This system is more readily available and can be tuned more specifically to a small group of designers and design tasks. On the other hand, it may lack of the flexibility, computing power, and large data bases of a big computer. Figure 3.11 illustrates in particular the interactive mode. The CAD execution process itself addresses a human operator via a terminal (usually a graphics terminal), and thus creates a new subprocess which we call a session. It is important to distinguish among the various levels of processes, and not to confuse the design process itself with the process of a session, even though both processes may be driven by the same person.

3.2.2 Design Process Characteristics and their Influence upon the CAD Process

According to Hatvany [HATV73], the main components of a CAD system are

- a person,
- computer hardware,
- software,
- a certain type of problem.

The type of problem is a characteristic which the CAD process has inherited from its superordinate design process. Here, we will discuss the type of problem in a very general sense. We will basically follow the classification schema used in [KRAU77] and illustrated in Fig. 3.12. Much work regarding the classification of individual phases of the design process was done at several German universities. A survey of several attempts to classify the design activities in a systematic manner is given by Pahl and Beitz [PAHL77]. We might also refer to the work of Simon [SIMO68], Roth [ROTH71], Hansen [HANS76], Koller [KOLL76], Rodenacker [RODE76], and Baumann and Looscheelders [BAUM82].

In the context of the previous sections, however, it seems more appropriate to use the classifications indicated in Fig. 3.13.

The development of a product is the response to a certain request which is to be satisfied. As an example of such a request let us use J. F. Kennedy's well-known statement (on May 25, 1961, before a joint session of the US Congress):

"I believe that this nation should commit itself to achieving the goal, before this decade is out, of landing a man on the moon and returning him safely to the earth".

This is not a specification of a vehicle which would be suitable to do the job! As a first step, the functional requirements of the product have to be worked out. For this task, the environmental conditions in which the product will have to operate need to be considered. This leads to a functional structuring of the anticipated product, and results in a functional specification. This process, which we call functional design, is informal, and highly intuitive. This does not mean that computer support is completely precluded at this stage. Though the process itself is informal, representa-

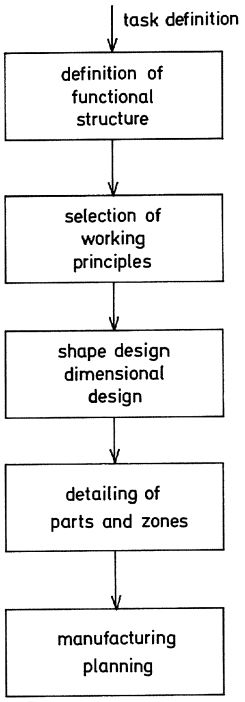


Fig. 3.12. Classification of design activities

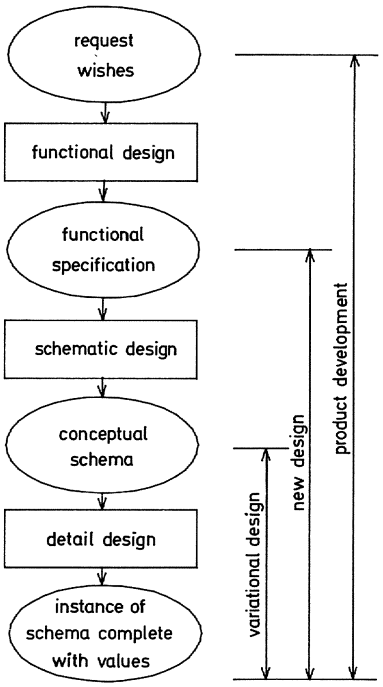


Fig. 3.13. Relationship between design activities and conceptual schema

tions of the elements in this process (the requests, the environmental conditions, and the resulting specifications) may be formalized.

Design, however, in the more strict sense used in this book, does not start until after the functional specification exists (see Fig. 3.13). The functional specification determines part of the conceptual schema of the product to be designed. The selection of working principles (such as welding versus screwing to hold two pieces together) and the gross shape and dimensional design, along with the selection of basic manufacturing methods, is part of the schematic design. As a result of the schematic design, the conceptual schema of the product is complete. This means that we know which constituent parts (entities) make up the product and which properties (attributes) of these constituents must be determined. The relationships among the entities are now fixed.

The final part ("detail design") deals with the assignment of specific values for all the attributes of an instance of this schema. For example, while the schematic design has determined the fact that certain edges of an object must be rounded and that the material is an essential property, the detail design will assign a value to the rounding radius and a material name to the material attribute.

The difference between schematic design and detail design will show up in the appropriate CAD systems used in these areas. In the detail design, theoretically at least, one could conceive an algorithm which would produce the unknown attribute values

of the conceptual schema from the representation of the specification within this schema. This algorithm may not be straightforward and may require a lot of iterations. However, the result (or at least one result if one exists) is totally predetermined. In schematic design the situation is different. Under what conditions a schema can be derived in algorithmic form from functional requirements – or whether this is at all possible – is a question we will not discuss here. In any practical case, the design of a schema has a pronounced heuristic and pragmatic character. Human judgement is the principal decision mechanism. Thus CAD systems in this area must be interactive from the basic principles, while in detail design CAD systems may or may not be interactive.

3.2.3 The Environment of CAD

3.2.3.1 The Organization

Considerable variety may be found with respect to the organizational embedding of CAD. This depends not only on the size and organization of the company, but even reflects differences in attitude towards CAD (as for example between the U.S. and Europe). As Allan [ALLA78] pointed out, American companies have a tendency to use CAD as a technical service whenever the need arises. The responsibility for CAD is more directly associated with the design departments themselves. In Europe, CAD (or CAD/CAM) is regarded as "an extension of management". Responsibility for the introduction and use of CAD methods and systems is preferably associated with higher-level management. This latter attitude may reflect the fact that the application of CAD is often accompanied by an increasing trend towards formalization and standardization, both in the products and in manufacturing. Full benefit will only be gained from CAD (and from these side effects) if not only the design aspect, but also work planning and manufacturing are involved.

3.2.3.2 The Human Environment

The human factor is dominant in the early phases of introduction of CAD in an organization. The spectrum of skills required to perform a certain design task will generally change when computer support is introduced. The most obvious change is that a certain amount of knowledge about computers and how to deal with them will be required. Exactly how much computer knowledge is necessary in the design environment will depend mainly on the following factors:

- access to a central data processing department versus the installation of one or more computers in the design department;
- use of black-box ("turn-key") CAD systems versus the introduction of a CAD system that supports user-defined software extensions; and
- complexity and amount of knowledge required to perform the design task.

Another change in skill requirements, however, is of much greater concern with respect to the people working in the design office. Since computer-based systems are

so well suited to performing analytical work along prescribed algorithms (programs) but so ill-suited for the actual "design" work (which is basically decision making), the introduction of CAD will cause a change in job content, as pointed out in [HATV77]. High-level designers will be able to increase productivity and product quality by using CAD; however, they may be subject to more stress during their work because they may miss certain periods of relaxation which had been caused by the more-or-less routine work prior to the introduction of CAD. On the other hand, there will be less need for low-level designers whose capabilities are limited to routine work.

Besides skills, a necessary condition for the successful introduction of CAD in a company is the motivation of the design personnel to use the new CAD tools and methods instead of the familiar conventional ones.

The introduction of CAD in general requires:

- higher-level skills (planning and decision making rather than execution and analysis); and
- motivation.

3.2.3.3 Computer Resources

In a CAD environment, three computer configurations are mainly found:

- the local computer;
- the remote computer; and
- the local satellite of a remote host computer.

Any one of these configurations may be found in different operational modes:

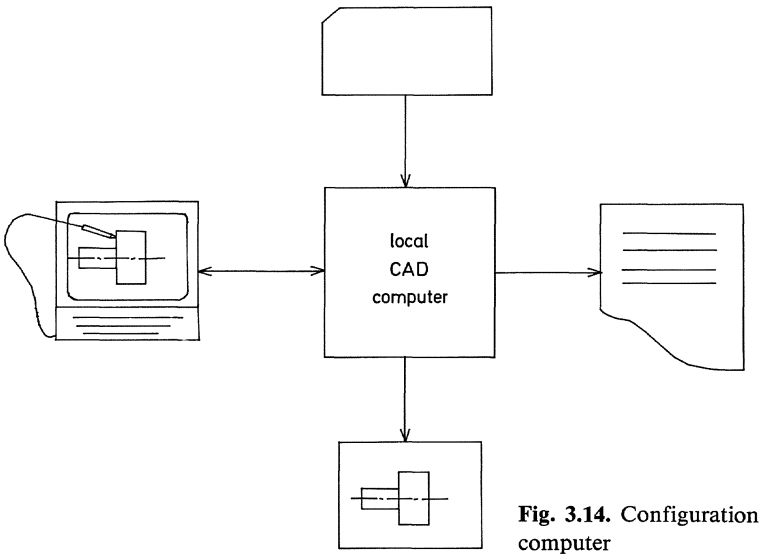


Fig. 3.14. Configuration of a local CAD computer

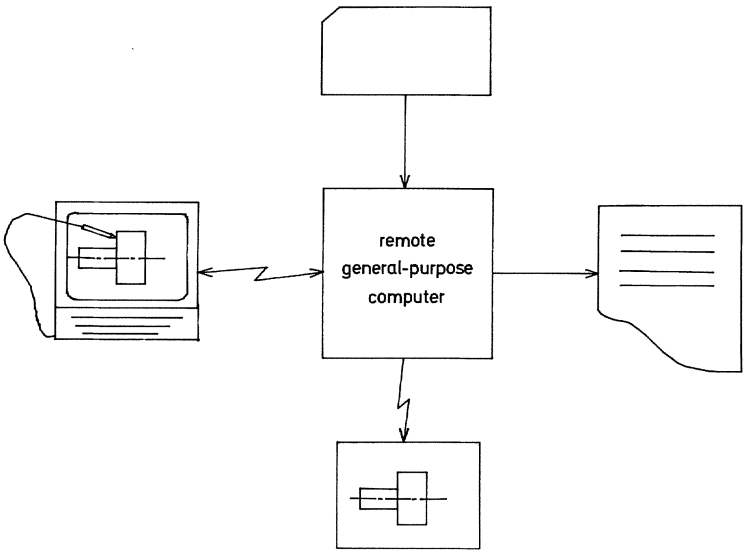


Fig. 3.15. Configuration of a remote CAD computer

- batch mode only;
- interactive mode only; or
- both batch and interactive mode.

Three of the many variations of CAD computer configurations are shown in Fig. 3.14 through Fig. 3.16. Figure 3.14 indicates the most primitive case which has practically disappeared: a local batch computer, with some means of program and data input (the keyboard input of data is just one of the many possibilities), and with text output on a printer and plot output on a plotter. Figure 3.15 shows a configuration with a local terminal (both alphanumeric and graphic) attached to a remote computer center. Figure 3.16 shows what is generally considered the most powerful configuration: a local computer with all alphanumeric and graphic input and output capabilities, backed up by a powerful remote computer.

The decision as to which configuration and which operational mode is "best", depends very much on the individual situation. This is particularly true for the first five years or so after the introduction of CAD into a company. Organizations which make significant use of a large general-purpose computer for non-CAD applications will in all likelihood start by attaching remote terminals to the central computer. Organizations with less computer background will probably increase the computer capacity of the design office from personal computers to one or more CAD workstations until they find it advantageous to connect some of these individual computers to a bigger background computer. The final configuration in a large organization will quite often have the structure shown in Fig. 3.16. (Note that the background computer may in fact be a network of computers, not necessarily a single machine.)

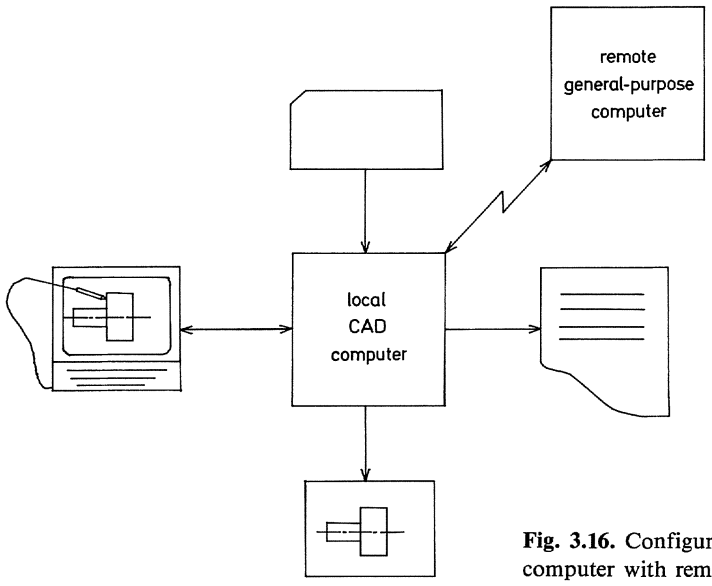


Fig. 3.16. Configuration of a local CAD computer with remote support

3.2.3.4 The Interaction Phases of the CAD Process

In the cooperation between a design process and a subordinate CAD process we may often distinguish various degrees of interaction in different phases. These phases are illustrated in Fig. 3.17. The first phase consists of the transmission of a prepared task specification to the CAD process. This "primary input" often includes a considerable amount of information which has to be checked for completeness and correctness before it is processed any further. The next step is usually a highly interactive communication between the two partners, involving adjustments and modifications of the specification until both can agree. The next step is the actual execution of the task. Interaction should be less prominant in this phase. Finally, the presentation of results often requires a high degree of communication when it emerges that more details are wanted than were included in the first presentation, or the way of representing the results should be modified for a better visualization of the essential aspects.

These phases may be more or less pronounced in a particular environment, so that they do not always show up in the architecture of CAD systems; but this principal structure is always useful to compare real CAD systems against. Besides the two highly interactive phases of a CAD process mentioned above there are two more situations which are associated with a high degree of interactive communication:

- information retrieval; and
- synergetic cooperation of human and computer (as is typical for computer-aided drafting).

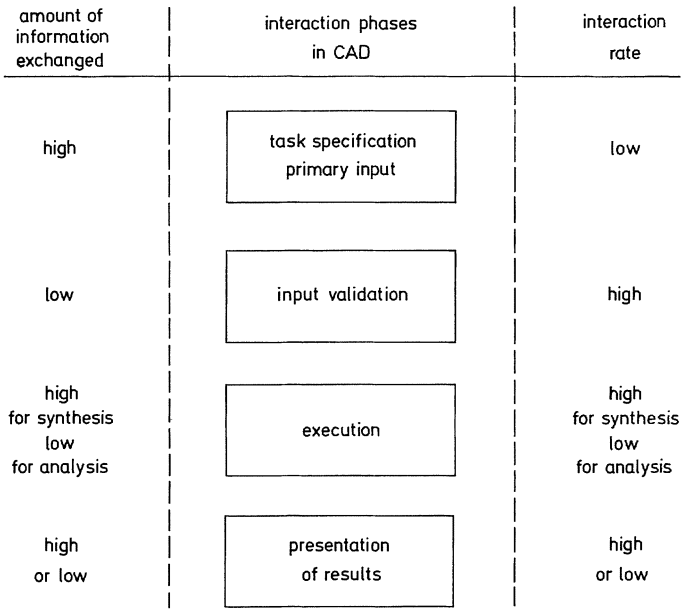


Fig. 3.17. The interaction phases in CAD

3.2.4 The State of CAD Processes

3.2.4.1 The Lifetime of Processes

In the previous sections we have not explicitly dealt with the aspect of time. Time, however, is an essential aspect of processes. Each process has a beginning and an end: we call this the lifetime of a process. The process exists only during its lifetime. At any moment during its lifetime, the process is in a certain state. The lifetime of a process is not independent of the lifetime of other processes. Note that the lifetime of a process includes both its "active" and its "dormant" phases (see Sect. 3.2.4.3). As Fig. 3.4 suggests, a process can exist only during the lifetime of its environment. Furthermore, a process which was created by another process as a subtask should (in most cases) return a result and terminate before the original process is terminated. In certain cases, however, a process may create another process without the need for a result; in such a case the creating process may well terminate before the created process. The basic requirement of process lifetime is illustrated in Fig. 3.18 for three examples.

First, the general situation is illustrated. The second example shows the environment process which makes a look-up table available (a book in the office or a data base on a computer). Although there is only one administration process which creates, maintains, and deletes from the table, several independent look-up processes may be executed. The third example is related to the design environment. Independent of a particular design process, the need for a CAD system has been realized and a process for the installation of a CAD system created. As a result, the CAD system has become

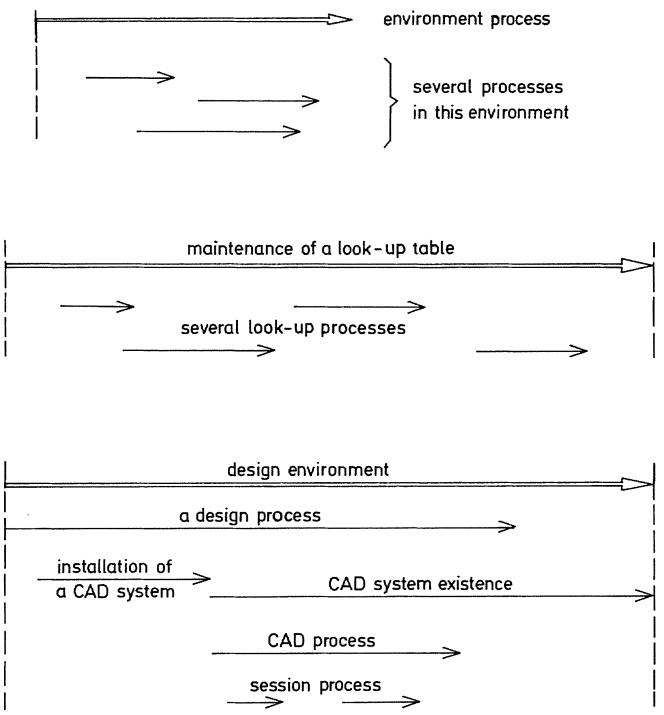


Fig. 3.18. Three examples illustrating process lifetime

available in the design environment. Some design process may now create a CAD process utilizing the system, during which process several sessions will be held. Several other design processes may easily utilize the same CAD system in parallel with the design process shown in the figure, provided that no conflicts arise.

3.2.4.2 The Representation of the Process State

At any point during its lifetime, a process is either active or inactive. To call a process active means that the process is changing its own state, or that the process is communicating with another process. We will have to deal with this concept of the state of a process in more detail. A designer can usually formulate the state of the design process he is working on. Certain design goals may have been achieved, while others are still to be reached; certain documents may have been produced but only in a "preliminary" version; inconsistencies may exist in the design itself due to incomplete updating of documents according to the most recent design changes, or there may be inconsistencies between the achieved solution and the design goals as a consequence of inadequate choices made in an earlier phase of design; even the knowledge about such inconsistencies is part of the changing state of the design process as analysis proceeds.

In general, one will be able to express the state of the design process in terms of the state of certain "things" (such as design drawings, written documents, of knowledge in human brains) which constitute the specific resources of the design process:

- The resources which represent the state of the process must be reserved for exclusive use by this process alone.

The same situation exists for a CAD process, except that the resources which represent the state of the CAD process are restricted to:

- machine-readable storage media and
- the knowledge of the "computer-aided designers", usually called operators in the CAD process.

We will deal with the knowledge of the operators in more detail when we discuss the communication aspect of interactive CAD systems. The state representation of a CAD process is related closely to the discussion of data structures and their representation. At this point it is sufficient to note that machine-readable storage media may be of very different types and may be looked at on different levels. The lowest level is represented by hardware, like bits in computer primary memory and on external storage devices. This aspect is of minor interest to the CAD user. However, only in rare cases can he avoid dealing with the state representation on the level of a computer-operating system, data base, or file management system, or on a programming language level. On these levels, the objects in a data base, the sequence of values in a file, or the values of variables in a program together with the corresponding underlying schema represent one part of the state of the CAD process at any instant. The second part of the CAD process state representation is the associated knowledge in the brain of the "computer-aided designer". Consequently, a major task in CAD is:

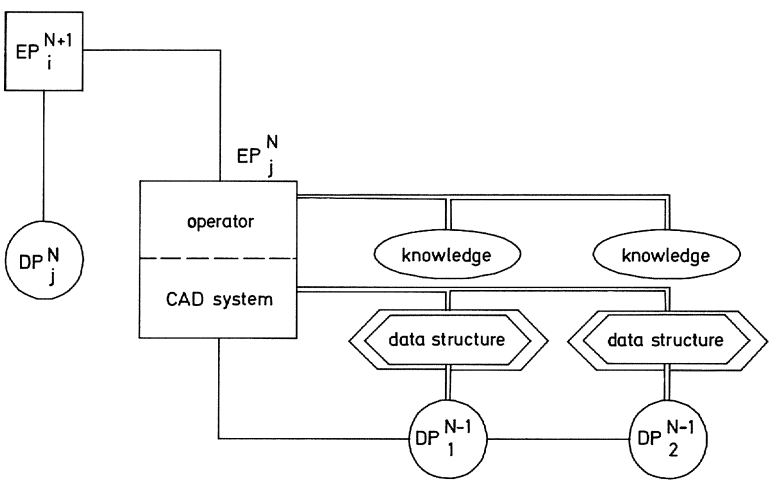


Fig. 3.19. The representation of the state of CAD processes

- to communicate the schema of the CAD process state to the computer in machine-readable form, and
- to fill an instance of this schema with values in a particular CAD process.

Figure 3.19 is a more detailed representation of the schema shown in Fig. 3.6. It distinguishes more precisely between the CAD processes themselves and their states. Furthermore, the CAD environment is separated into its two main components: "operator" and "CAD system". The state of process DP_i^{N-1} is defined as the combination of the data structure of the CAD system and the associated knowledge of the operator. While the schema is the same for all CAD processes DP_i^{N-1}, the content is different for each of them. The machine-readable resources representing the state of the processes are associated with the overall CAD system (that is, the environment process), while the values belong to the individual processes.

3.2.4.3 The Operating State

It is advantageous to define the various situations of a process in terms of an "operating state". This is common practice in process control, and we will use a simplified version of the state diagram used in the definition of the PEARL language [PEAR81]. Other programming languages which support the concept of a process might be used as a reference as well [WINK79], [WINK80]. The correspondence between PEARL and the notions used here is as follows:

PEARL	*this book*
task	environment process
activity of a task	CAD process

Figure 3.20 shows the operating states of the environment process (we have omitted the scheduling feature of PEARL). The environment process EP(N) (in Fig. 3.19) is said to be dormant if no CAD processes exist in this environment. It is runable if at least one process exists and is executable or being executed; it is suspended if at least one process exists, but none of them wants to proceed. In addition it is useful to introduce the state "unknown", which actually is not a state of the environment itself, but is rather associated with the relation between this environment and some other

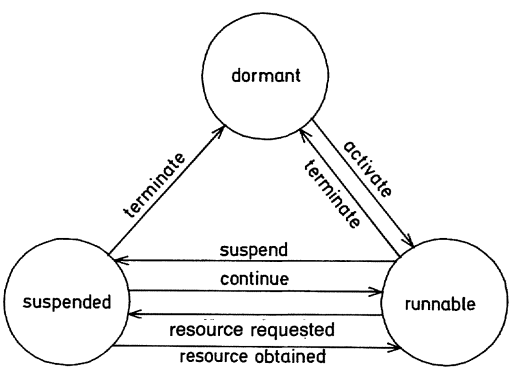


Fig. 3.20. The operating states of an environment process

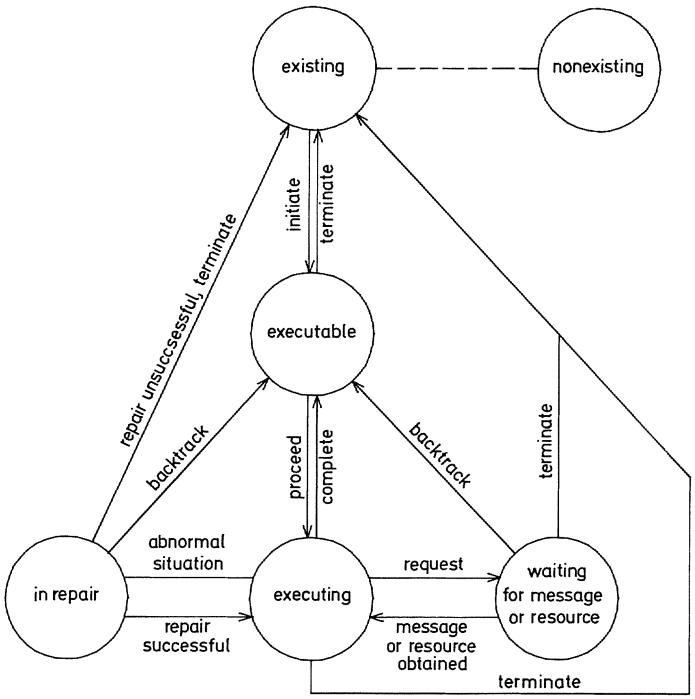


Fig. 3.21. The operating states of a CAD process

(environment or CAD) process. An environment is unknown to some other process if the latter does not know about its existence, or at least does not know how to communicate with it. The operating state of an environment is thus closely related to the operating states of the CAD processes which belong to it (Fig. 3.21). Every CAD process must pass through the operating state "existing". It is in this state immediately after it has been created in the environment upon the request of some other process. In order to become executable, the CAD process must be initialized. For initialization, values which are meaningful and consistent with the operations to be performed are assigned to certain objects in the corresponding data structure. This corresponds to the communication of the design goal to a subordinate design process, as discussed in Sect. 3.1.2. If there are any steps to be taken, the process will pass to operating state "executing", from which it will return when the task is completed. In the executing state, the need for support by other processes may occur. Resources may be needed; new subprocesses may have to be created and their responses awaited; "help" may be needed from a higher-level process or the environment process. In such cases, the CAD process will pass to the operating state "waiting". The operating state "in repair" is reached if abnormal conditions arise. It might be emphasized that this operating state should be investigated much more intensively than has been done in the past. Computer-based systems often become complicated not because their normal tasks are complicated, but rather because abnormal situations and repair mechanisms have not been taken into account in sufficient detail.

From both the "waiting" and "in repair" states the process can normally return to "executing". However, if the message cannot be received, or the resource cannot be obtained, or the repair is not successful, the process will have to return to either "executable" or "existing". The first is certainly preferable, because it leaves the process in a consistent state from which it can proceed (for instance, by repeating the step which has just failed).

3.2.5 The Problem of Resources

3.2.5.1 Resource Availability and Conflicts of Resource Requirements

In the previous paragraphs we have mentioned "resources" several times as something indispensable for a process. Examples of such resources are:

- time, money, manpower, storage capacity, or a processor; and
- a certain piece of hardware, a certain file, a certain compiler, a name.

The difference between the two groups of resources mentioned above is their ability to be substituted by other resources. This will later cause differences in how these two groups of resources have to be treated. At this time we will deal with their common aspect: we consider as a resource,

> anything needed which may have limitations of availability in the environment considered [SCHL78].

The resource problem is less evident if only one process exists (or can exist) within the environment. In this simple case, practically all the resources of the environment are available to the process. However, when several processes exist in parallel, conflicts may arise already due to the fact that the state representation of each process requires resources which cannot be shared. Serious conflicts are even more likely if several of the processes are in the executable operating state (including the "executing", "waiting", and "in repair" states). Computer science has developed several constructs, like semaphores [DIJK68] and others, which deal with the problem of resolving conflicts [BRIN73]. Furthermore, Petri nets (or P-nets) may provide the necessary tools to handle the problems of coordinating the individual parallel tasks in a design process [ZUSE80]. However, to the authors' knowledge these techniques have not yet been introduced into CAD systems. The reason why CAD has disregarded this problem thus far is probably that CAD systems which support several CAD processes in parallel and in real time are uncommon. Furthermore, the notion of a "process" is usually associated with a "job" or a "session" on a computer, not with the longer-lasting design task. Finally, constructs for the coordination of concurrent processes have found their representation in programming languages [BRIN75], [PEAR81] which are as yet unfamiliar to the CAD community, while in data base systems these problems tend to be taken care of entirely by those systems and, hence, are hidden from the user. In Sect. 4.3 we will have to deal with the problems of resource management in an environment which does not provide semaphores or similar constructs for resource management.

Names are a special kind of resource. Names are used in processes as substitutes for objects, while the objects themselves are each represented by some lower-level process. The action of replacing the name by the actual object is called binding [SALT78]. Resource problems may arise in two ways:

- the use of identical names for different objects may cause an inconsistency;
- during binding (replacement of the name by the object itself) it may turn out that the required resources are not available.

Examples of binding actions are:

- the inclusion of subprograms from a library into a program module in a process. The words "binding", "mapping", or "linkage editing" are common, depending on the computer manufacturer's terminology;
- the establishment of a connection from a terminal to the main computer (by telephone dialing for instance);
- the opening of an actual data file by a CAD program; or
- the substitution of a computer memory address for a programming language variable name during compilation.

The aspect of resource availability is important with respect to CAD systems. CAD systems which are designed to operate on a large central computer may not be applicable in an environment which does not provide sufficient memory capacity. On the other hand, an interactive CAD system which is used successfully on a small computer may be unacceptable when operated from a remote terminal on a large central computer where it has to share resources (central processor time, communication channels) with other processes. Software such as compilers for certain programming languages, or specific data base management systems or subroutine packages might also constitute resource requirements which restrict the applicability of CAD systems. The same even applies to human resources when a certain kind of knowledge for operating the CAD system is required. As a consequence, a CAD system is fully described only if its resource requirements are spelled out in addition to its functional capabilities. It is not surprising that several CAD systems exist which are quite similar in functional respects but differ in their resource requirements.

3.2.5.2 The Efficiency Aspect of Resources

Besides the functional aspect of availability of resources, efficiency of their utilization is also an important consideration in CAD systems. Efficiency is related to cost, and hence introduces a commercial aspect. CAD systems which operate efficiently in one environment may operate less efficiently in another. In the early years of CAD the size of primary computer memory was limited because of high costs. For this reason, CAD systems were developed which could operate with a minimum of primary memory by storing all but the most immediate data on peripheral devices. The tremendous decrease in primary memory cost [SCHU78] has caused a shift towards larger memories for the optimal use of resources, while on the other hand time is becoming more valuable. The decrease of memory cost together with the spreading of 32-bit computer architectures and virtual storage has emphasized the use of more

primary storage while minimizing time-consuming accesses to secondary storage. It should be noted that the economic factors related to resources may have a considerable influence upon CAD systems.

3.2.5.3 CAD Machines and CAD Tools

Engineers are used to thinking of machines. By analogy, a CAD system may be compared to a machine. In CAD the product is design information, and the resource representing the product is paper or a certain region in a data base (rather than a piece of hardware). The resources which this "machine" uses in the production process are mainly computer hardware and software (instead of hydraulic forces and lubrication oil). Both the CAD system and the conventional production machine need control or at least supervision by a human.

The analogy goes further. Machines may be designed for "stand-alone" use. This is the common situation when so-called "turn-key" CAD systems are used [ALL__78]. But machines are quite often used in a larger environment where a transfer system stores the intermediate product and transports it from one machine to another, while in the meantime some checking may be performed and the piece of work may have to be oriented in a different way before it is inserted into the next machine. The transfer system finds its analogy in CAD in a data base management system, the intermediate human checking being done with a suitable query language, while the different orientations of the same pieces of work correspond to the different "views" of the same objects in the data base according to different subschemas (see Sect. 3.3.2).

3.3 Modeling in CAD

3.3.1 Developing a Schema

3.3.1.1 Basic Considerations

In the previous sections we have tried to develop a conceptual model of the design process. The model is far from being formal, since that was not the intent. But it provides a suitable basis for talking about CAD, for developing more formal models of CAD, and for designing CAD systems. The task of developing a model recurs in every design task (see Fig. 3.22 and Sect. 3.1.2). Because computer systems are lacking in the ability to do synthesis, the development of a model – or more precisely: the set of models that belongs to the scope of a CAD system – is generally a task to be performed by the designer of a CAD system. The user of the CAD system can develop product models only within the restrictions imposed by the designer of the CAD system. The description of the whole set of possible models that can be developed by the designer when using a CAD system is called the conceptual schema. The design of the schema, however, does not only depend on the objects to be designed, but also

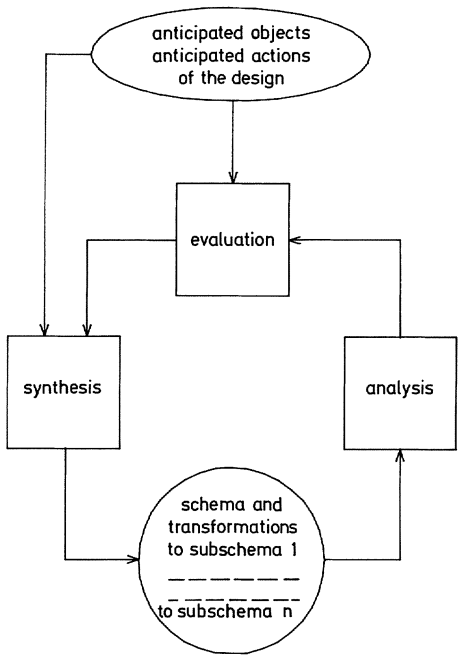


Fig. 3.22. The CAD process and its schema

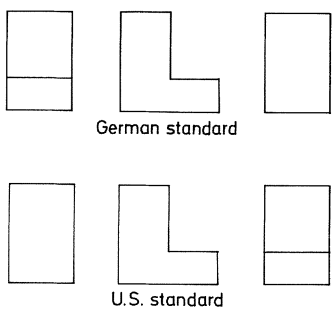


Fig. 3.23. Design drawing standards

on the anticipated actions to be taken. In conventional design the actions to be taken are all related to human information processing. Over many years a suitable schema for a very large class of objects has been developed: the standards for design drawings. Note that the standards for design drawings are not identical all over the world. Figure 3.23 shows the same object according to German and American standards. These two standards serve the same purpose equally well, yet they are different. They illustrate that "correctness" is not an applicable criterion for evaluating a conceptual schema. "Suitability" is more appropriate. We group from [BRUN56]:

- "The categories in terms of which we quote the events of the world around us are constructions or inventions. The class of prime numbers, animal species, the huge range of colors dumped into the category 'blue', squares and circles: all of these are inventions and not 'discoveries'. They do not 'exist' in the environment. The

objects of the environment provide the cues or features on which our groupings may be based, but they provide cues that could serve for many groupings other than the ones we make. We select and utilize certain cues rather than others."

The essential question to be answered by analysis and evaluation of a conceptual schema is:

- Is the schema suitable for efficient transformation to and from the various subschemas required by the different design steps anticipated?

The examples in the subsequent chapters will illustrate these considerations.

3.3.1.2 A Sample Problem

This sample problem may appear to be too trivial. Yet it exhibits all the essential features of schema planning in CAD applications. Figure 3.24 represents an object which has some similarity with a hammer (although we do not claim that it is a good hammer). It would not be too difficult to manufacture this hammer in a workshop with the information given in the figure. Let us take a closer look at this information. It contains

- structural information:
  the object consists of two subobjects (head and shaft);
- geometrical information:
  the geometric shape of head and shaft, and the geometrical position of these two pieces after assembly; and
- information regarding manufacturing:
  the material information (presumably together with the geometrical data) will influence the selection of the raw material for both pieces. The tolerance data will influence the quality control.

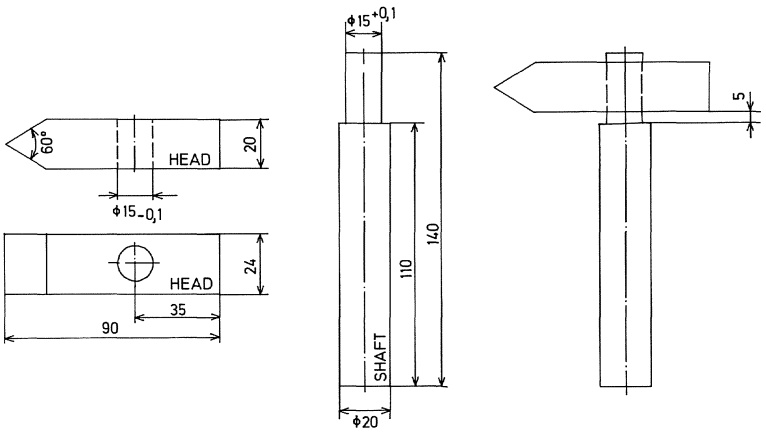


Fig. 3.24. The graphical representation of an object

Note that the drawing itself is not a complete description of the hammer. It is complete only in a certain environment. In order to complete the geometrical information, one must additionally apply the rules for representation of bodies in design drawings. Only with this additional knowledge can one conclude that the shaft is basically a circular cylinder with flat ends which are perpendicular to the center line of the cylinder. Furthermore, one must know that the standard length unit is the millimetre. If we neglect the implied rules, we find tremendous difficulties in describing the geometry. A verbal description without any graphic support would be quite lengthy if at all feasible. A particular problem of CAD is that computers do not know about the rules of design drawings and cannot read a design drawing as a human does. Hence a different schema is required to represent the information.

3.3.1.3 Naming of Objects and Attributes

A fundamental difference between a design drawing and a schema suitable for computer application is the naming requirement. While on the design drawing entities may be pointed at ("this length is 140 mm"), the entities in a schema and their attributes must have names assigned with them [SALT78]. We obtain one possible schema of the hammer by simply replacing each number by a respective name. We have many choices in doing this, the first choice being the set of allowable names. In Fig. 3.25 we choose as names identifiers built from an arbitrary number of capital letters and numbers. However, we might as well have chosen another naming system, such as assigning a positive integer value to each of the entities. In a more general sense, even a position may be taken as the name ("the lowest line of the HEAD representation"). Since information processing with computers is generally based on information representation in the form of character strings and numbers, names of this form are most common. In the context of graphic data processing, we will discuss the use of graphic representation of names in more detail. In this chapter we will restrict ourselves to the usual form of names, namely character strings.

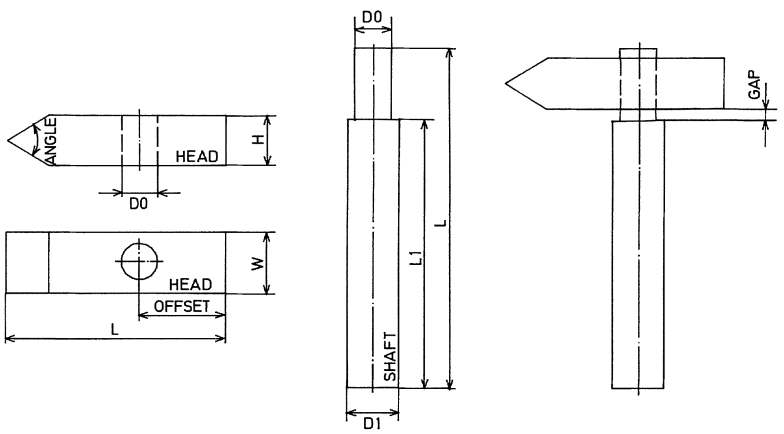


Fig. 3.25. The graphical representation of a schema

In any case, names must be unique in the environment where they are used. In Fig. 3.25 we select D0 and L for two entities each. Nevertheless, the names are unique if we prefix (or qualify) them with the name of the object to which they belong: HEAD.D0 and SHAFT.D0 are two unique names. The same principle applies if we consider the possibility of two different processes in the same environment using the same name for independent entities. Again the technique of qualifying the name (in this case with the name of the appropriate process) will make the name unique. Problems will arise only if we restrict the rules for naming so that qualification is not possible. In this case conflicts are likely to occur.

3.3.1.4 Alternatives for a First Schema

A logical next step would be to transform Fig. 3.25 into a machine-readable representation. We choose a Pascal-like notation:

```
TYPE  MEASURE = RECORD
      VALUE, TOLERANCE          : REAL
                        END;
TYPE  HEAD__SCHEMA = RECORD
      ANGLE, H, W, OFFSET, L    : REAL;
      D0                        : MEASURE
                        END;
TYPE  SHAFT__SCHEMA = RECORD
      D1, L1, L                 : REAL;
      D0                        : MEASURE
                        END;
TYPE  HAMMER__SCHEMA = RECORD
      GAP                       : REAL;
      HEADPART                  : → HEAD__SCHEMA;
      SHAFTPART                 : → SHAFT__SCHEMA;
                        END;
```

We now ask ourselves: Is this a suitable schema? As mentioned before this question can be answered only if we consider the actions to be taken. Consideration of the objects alone is insufficient. First of all, we note that we are able to identify objects of the types defined above by combining the name of each object with the name of a schema:

```
VAR  HAMMER: HAMMER__SCHEMA;
     SHAFT  : SHAFT__SCHEMA;
     HEAD   : HEAD__SCHEMA;
```

We call HAMMER an "object" of type HAMMER__SCHEMA in order to distinguish between a schema and an instance of this schema. We use "object" as a synonym for "data structure", whenever the representation of a real world object in

terms of data is concerned. In the literature, we can also find "data structure" being used instead of "schema", i.e., for a whole class of objects. Since we generally allow several processes to exist in one environment, each of them being characterized by its appropriate state, we must distinguish between the abstract schema and its individual instances, one of which is associated with each process (see Sect. 3.2.4.3).

We can also assign values to certain quantities:

```
HAMMER.GAP    : = 0.0005;
SHAFT.D0.VALUE: = 0.015;
HEAD.D0.VALUE : = 0.015;
```

Here, implicitly, we have introduced the convention of using the ISO standard for physical units (i.e., m for length) instead of the mm unit which was the design-drawing standard. At this point we could use a data structure built according to the above schema as a memory. All we need is a query language which allows us to retrieve data from the storage and to represent them in a form readable to man or program. This is in fact a fundamental function of data bases. Note, however, that we do not use the "pointers" of the schema hammer. Indeed, we have not needed them so far. It would be more appropriate to include the HEAD__SCHEMA and SHAFT__SCHEMA in the HAMMER__SCHEMA as follows:

```
TYPE COMBINED__SCHEMA = RECORD
                    GAP: REAL;
                     D0: REAL;
                  SHAFT: RECORD
        D0__TOLERANCE: REAL;
              D1, L1, L: REAL
                    END;
                   HEAD: RECORD
        D0__TOLERANCE: REAL;
ANGLE, H, W, OFFSET, L: REAL
                    END
    END;
```

Note that the COMBINED__SCHEMA differs from a simple combination of the three previous schemas: the nominal value of diameter D0 has been removed from both the SHAFT__SCHEMA and the HEAD__SCHEMA and is now given only once. This reflects the functional requirement that the two parts must have the same nominal value for this diameter in order to fit together. This is a consistency condition of the original data structure, which results from a redundancy in D0 in just the same way as in the design drawing (Fig. 3.24). While the COMBINED__SCHEMA implicitly guarantees consistency, the first solution with separate HEAD__SCHEMA and SHAFT__SCHEMA requires an explicit check of the consistency, whenever HEAD and SHAFT are combined into HAMMER.

3.3.2 Influence of the Operations upon Schema Planning

We will use the example HAMMER from the previous section to illustrate the type of considerations necessary during schema planning. Let us plan for the following actions:

- we wish to deal with hammers with different values of D0;
- we wish to combine heads and shafts having different length dimensions but the same value of D0. We do not expect any changes in the angle value of 60 degrees assigned ANGLE;
- we want to produce design drawings similar to Fig. 3.24 from the data base;
- we want to compute the weight of shaft and head for each hammer;
- we want to query the data base, and
- we will apply these actions one at the time to one hammer at a time.

Before we continue we must make some additional assumptions:

- The algorithm (the program) for producing the design drawing must know all the lines and texts to be displayed. It must know where to place these lines and in what line width.
- The algorithm for calculating the weight must know the material density. It will work only for simple volume shapes such as cylinders, quadrilaterals, and prisms. The head material will always be steel, shafts may be made from steel or wood.
- The query action is satisfied by a schema like the one described in the previous chapter. The same applies to the action of assigning values to the quantities in the data structure.

A graphical representation of the situation is given in Fig. 3.26. Within the overall schema we distinguish the input and query subschema (which is identical to the original schema for HAMMER, SHAFT, and HEAD derived in the previous chapter but including the WEIGHT in order to facilitate queries for the weight after it has been determined by the weight analysis algorithm). The drawing subschema contains all the graphical data, the weight analysis subschema all data relevant for this action. Each subschema presents the data in the form which is suitable for the respective action. Comparing this schema with the previous one and with the planned actions, we note:

- the original schema has become a subschema for input and query, but the weight has been included as an additional attribute;
- it has become advantageous to use the more complicated triple of HAMMER, HEAD, and SHAFT, rather than the COMBINED_SCHEMA, since we want to combine different shafts and heads to build all sorts of hammers;
- two additional subschemas have been developed, one representing all data required for the weight analysis algorithm based on elementary geometrical shapes plus material information, the other one representing all data required to produce a design drawing;

The situation, however, is as yet unsatisfactory because

- we cannot derive the data required to fill the drawing and the weight analysis subschema from the input subschema;

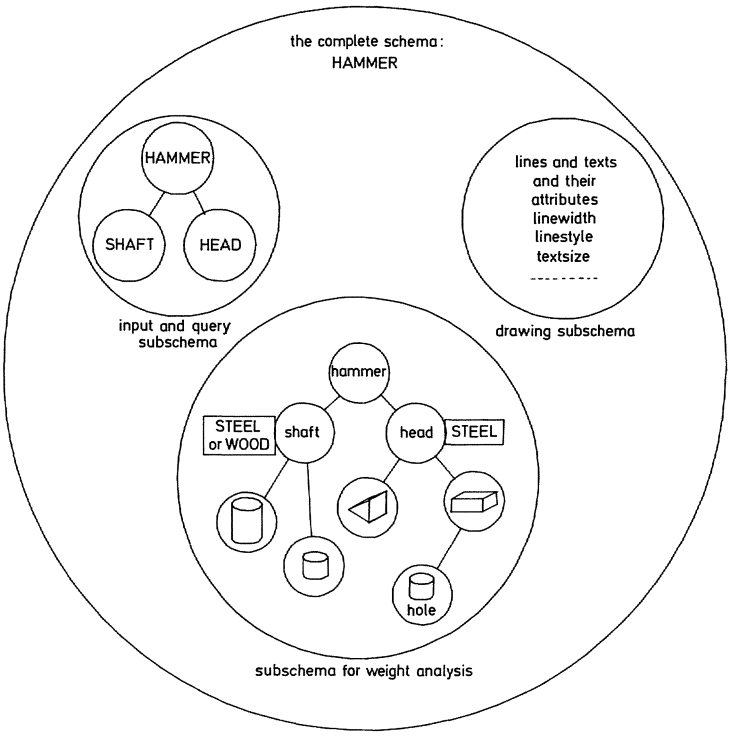


Fig. 3.26. Refinement of a schema by the definition of subschemas

- we may calculate the weight, but the value cannot yet be returned to the input and query schema;
- no provisions have been made to avoid the construction of a hammer with a shaft of 15 mm D0 and head of 12 mm D0; the important query schema contains ANGLE and WEIGHT, although ANGLE should always be 60 degrees, and WEIGHT can never be a legal input value.

The two attributes weight and material may be considered as examples, illustrating the freedom that we have in schema planning. Instead of the solution illustrated in Fig. 3.26, we might have included the weight in the weight analysis subschema and/or the material information in the input and query subschema. Depending on our choice, either the input or the query action has to access one or both of these subschemas. Efficiency considerations, based on the estimated number of occurrences of the various actions, must generally be applied to decide which of these solutions is to be preferred.

3.3.3 Subschema Transformations

3.3.3.1 Subschema Transformations as Part of the Schema

How can we produce the data required to fill the drawing schema? These data are necessary for the plotting operation. Two extreme positions may be taken:

- we generate all drawing data from scratch; or
- we generate all drawing data from the data in the input and query schema.

The first approach (as shown in Fig. 3.27) is certainly feasible. In fact, it was the standard approach taken in the early stages of CAD. However, two drawbacks are quite evident: expected economical advantage of introducing CAD is lost if the user has to handle basically the same information many times and cast it into a different subschema once for every action; furthermore, the problem of inconsistency arises since it may easily happen that the drawing displays a length value of 200 mm for the shaft while the corresponding value in the SHAFT subschema is 140 mm.

The second approach (shown in Fig. 3.28) corresponds to the CAD user's paradise. But this paradise is almost nonexistent. How could we possibly derive the size of the drawing text from the input and query subschema? We might, of course, build this knowledge into the "drawing data generator". But this eliminates the flexibility which is needed whenever the outcome of this built-in procedure is unacceptable to the user (in the context of Sect. 3.1.2, we would say that the achievement presented to the higher level process is unsatisfactory).

It will be necessary to guide the drawing data generator when it produces the drawing by adding the information on how the object should be displayed while all information about what to display may be taken from the input and query subschema. In general, we find that transformation of information from one subschema to another

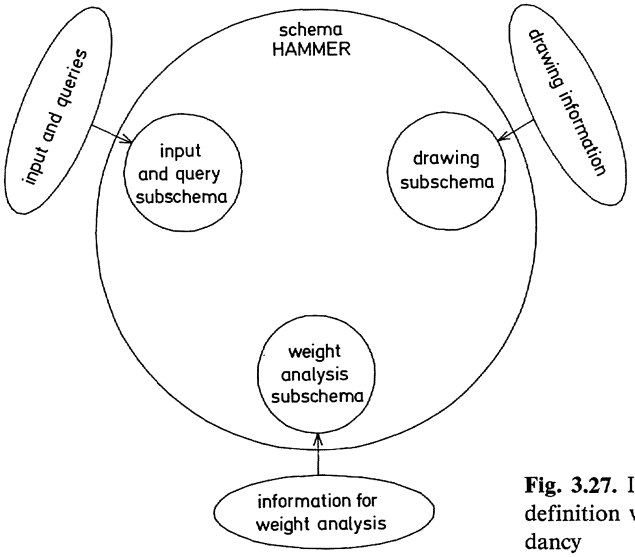


Fig. 3.27. Information flow for object definition with complete schema redundancy

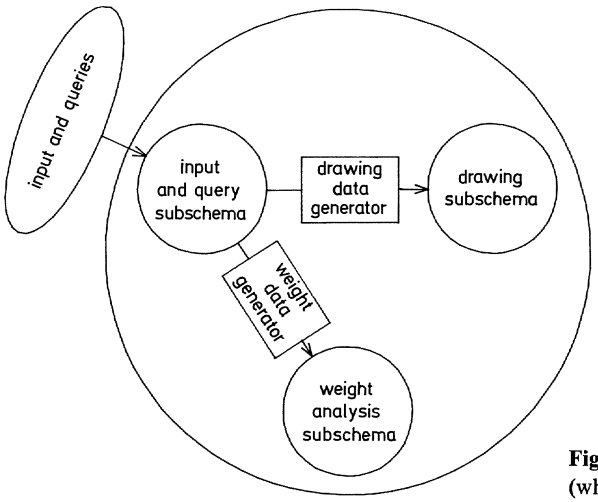


Fig. 3.28. The CAD user's paradise (which does not exist)

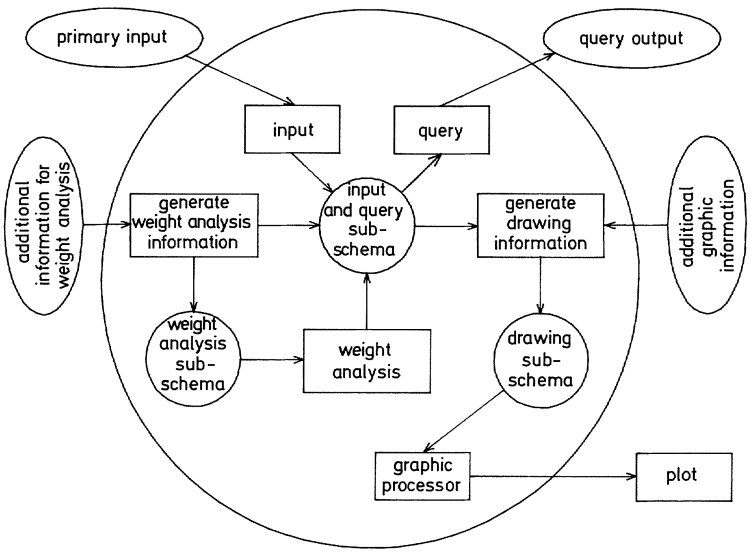


Fig. 3.29. The relationship between subschemas and operations

requires additional information. The transformers which take data from one subschema and generate data of another subschema (with additional information) obviously must know two subschemas rather than one. For the sample case discussed so far the situation is shown in Fig. 3.29.

3.3.3.2 The "n-square" Problem of Subschema Transformations

In CAD system design, one often tries to minimize the number of subschemas in one schema. The reason for this tendency is evident if one assumes that information may

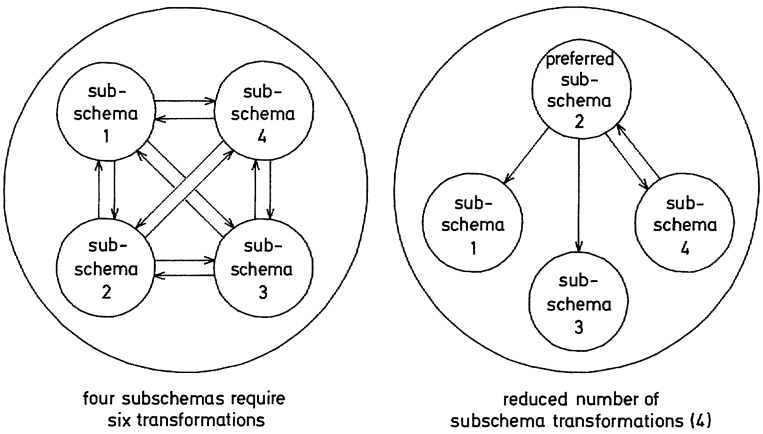


Fig. 3.30. The "n-square" problem of data transformation

have to flow from any subschema to any other subschema. In this case, with n being the number of subschemas, a total of $n \cdot (n-1)$ transformations would be necessary. Because of the quadratic increase in the number of necessary transformations (which would roughly result in a quadratic work increase for the implementation of the CAD system), the CAD system designer will often attempt one of two solutions:

- use only one subschema even if this is unnecessarily complicated and wastes resources in many instances; or
- use one preferred subschema, from which and to which all transformations are done.

The first solution is the basis of many successful CAD systems. Finite element programs are typical representatives of this class. The great advantage of the finite element approach lies in the use of a single subschema (or very few) for even the most complicated problems (triangles and rectangles, for instance, in the two-dimensional case). The same approach is taken in a number of line drawing systems which are based entirely on polylines for storing geometrical information, even for such regular objects as circles and rectangles.

The second solution corresponds to the preferred mental model of today's data base management systems [ECKE77]. This preferred model is a flexible schema on which all other subschemas are based. In many realizations, however, only a one-way transformation is implemented. In line drawing systems, for instance, the polyline is most often the preferred subschema for graphical information. Circles, arcs, and texts may easily be transformed into polylines. But one does usually not bother about the reverse.

An organization planning to introduce a solid modeling system will usually be faced with the problem of which subschema for solid model representation it should choose. Both constructive solid geometry and boundary representation techniques (to mention only the two most widely used ones) have their advantages and disadvantages. Constructive solid geometry models may be transformed into boundary representations, but the reverse is practically impossible.

In a general case, subschema transformations may have to use data from more than just one subschema in order to produce data for an additional subschema. For four different subschemas the situation is illustrated in Fig. 3.30. The subschema transformations must of course be part of the environment in which the schema exists. In fact, it is advantageous to consider the subschema transformation as part of the total schema itself.

3.3.4 Flexibility – A Measure of Prudence – Versus Efficiency

CAD experts are quite familiar with a very serious problem: after some effort has been put into the planning of a CAD system (or maybe even after implementation) suddenly the goal changes. There may be many reasons for such changes:

- technical development calls for a design change of the objects;
- economical development sets different priorities;
- increased insight into the effect of introducing the CAD system opens a door to new wishes.

It is therefore good policy for CAD system developers to anticipate such new wishes, and to clarify possible problems beforehand. It is often prudent to plan for goals which are somewhat beyond what is actually requested. This will allow the classification of future wishes into:

- options which should immediately be included in the plan, even if they are not actually required;
- options which may be added later at low cost, provided that such future modification is taken into account in early planning;
- options which would require a more or less new approach.

In the example of the hammer used above, we have already mentioned the first type of these options: although the angle at the front edge of the hammer should always be 60 degrees, it is good strategy to include this information in the schema instead of building this knowledge into all the algorithms (for drawing and weight analysis). Storing the angle value without using it for computation would be dangerously redundant, as both the data structure and the algorithms would know the same information. If the angle value is made part of the schema, the algorithms should use it even if it always is the same constant value. Efficiency of execution may be increased in one case, while the algorithms would require an extra look at the data structure in the other case. But the gain in efficiency is probably less important than the increased flexibility. The safety is not reduced by this approach, since the input subschema can easily be restricted so that the value of "angle" can never be input but is rather initialized with the desired 60 degree value. The second type of options may result from the consideration that the front edge of the hammer should perhaps be rounded in more advanced versions. This could be achieved by a slight modification of the schema and somewhat more sophisticated algorithms. The designers of the CAD system might choose to provide the schema for this more advanced goal, but still implement the algorithm for the primitive version only. This approach would make a steady enhancement easier than if the old schema had to be replaced by a new

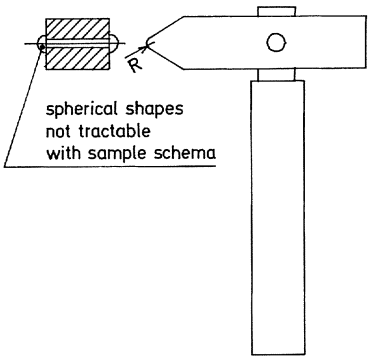


Fig. 3.31. Modification of an existing schema

one. The third class of options may be illustrated by the consideration that head and shaft of the hammer might be pinned together as shown in Fig. 3.31 in some later version. Since the pin would involve spherically shaped bodies for which (according to our assumptions) the algorithms for weight analysis are not available, and hence the corresponding subschema is as yet undetermined, this option should not be included in the planning of the schema. The purpose of this example is to illustrate the arguments which should be considered in planning a schema. However, the appropriate decisions must be taken in each individual case. There are no general rules, with one exception:

- mistakes in the planning of the schema of a CAD system will drastically limit the success of design processes using the system.

3.3.5 Schema Planning and Design Process Planning

3.3.5.1 Subprocess Planning and Data Validity

Thus far, in the discussion of schema planning, we have not adequately considered the process aspect. Schema planning is strongly influenced by the planning of the design process. We can say that:

CAD modeling = schema planning + design process planning.

We will again use the example of the hammer in order to illustrate this point. The basis of our considerations is the schema illustrated in Fig. 3.29. Let us assume that the task to be solved is the design of a hammer with properties which:

- (class A) in some parts can be represented by certain values of quantities in the input subschema (such as geometrical data);
- (class B) in other parts can be formally represented as restrictions, with respect to values which are associated with the query schema but not found in the input schema (such as the weight, which should never be input); and
- (class C) in certain other parts cannot be represented in a formal way within the schema (the design drawing "doesn't look good").

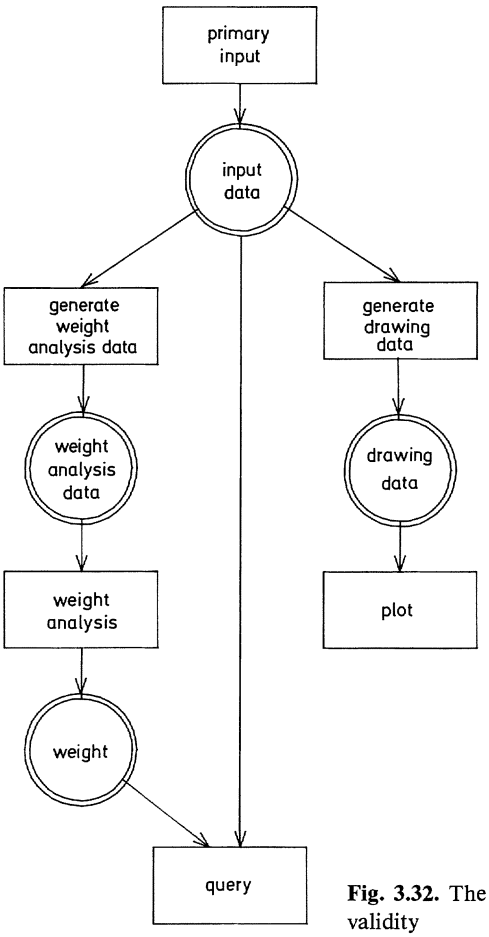


Fig. 3.32. The precedence of operations required for data validity

Most design goals include all these three aspects. Class A properties can be specified directly in the input. Class B properties may be obtained only by an interactive process, but this process may potentially be formalized. Thus the successful design process may or may not include a human. Class C properties require that a human be included in the process to perform the task of evaluation (see Fig. 3.3). In this example, we ignore the possibility of automatic iteration of the weight and leave the evaluation task entirely to a human. Process planning consists mainly of combining the elementary operations available in an environment into subprocess units. It is obvious that certain primitive operations must precede others: without prior input no other operation would be meaningful; without prior generation of weight data, the weight analysis would not be useful. The necessary sequence of precedence is illustrated in Fig. 3.32.

One extreme alternative would be to combine all operations into a single process. In this most simple case the process would pass from the operating state "existing" immediately after creation through "executable" to "executing", and would return to

"existing" after completion. Finally the process would become "nonexisting". The individual operations would be scheduled within the process in a predetermined way consistent with the precedence requirements to guarantee that each operation uses only valid data. The data validity problem is thus completely resolved once the internal scheduling is fixed in accordance with the needs. This approach is the basis of any batch-oriented CAD systems. The great advantage of this approach is that the validity problem can be solved once for all processes in the environment. A second easy way to solve the problem is to redo all dependent operations after modification of the data on which they depend. This approach would lead to a subprocess structure as follows:

subprocess 1: primary input immediately after creation of the process, generate weight analysis data, weight analysis;
subprocess 2: generate drawing data, plot; and
subprocess 3: query.

With this partitioning of the operations into subprocesses, no validity problems arise since both subprocesses 2 and 3 can start only with valid data. However, this approach is far too restrictive in many cases. It may even be unacceptable: it may well happen that the weight is totally irrelevant in a particular case. No designer would readily accept the necessity to provide additional information for weight analysis if he is not interested in weight. Furthermore, the weight analysis may be costly, and should not be carried out unless wanted. Thus we may prefer the following subprocesses.

- subprocess 1: primary input immediately after creation of the process;
- subprocess 2: generate weight analysis data;
- subprocess 3: weight analysis;
- subprocess 4: generate drawing data, plot;
- subprocess 5: query (with options for weight or no weight data).

In the subschemas this approach would be reflected in the following way:

```
TYPE HAMMER_SCHEMA = RECORD
                GAP : REAL;
          HEAD_PART : →HEAD_SCHEMA;
         SHAFT_PART : →SHAFT_SCHEMA;
       WEIGHT_VALID : BOOLEAN INITIAL (FALSE);
             WEIGHT : REAL
                    END;
```

or perhaps

```
TYPE HAMMER_SCHEMA = RECORD
                GAP : REAL;
          HEAD_PART : →HEAD_SCHEMA;
         SHAFT_PART : →SHAFT_SCHEMA;
CASE WEIGHT_VALID : BOOLEAN INITIAL (FALSE) OF
              FALSE : ( );
               TRUE : (WEIGHT: REAL)
                    END;
```

The query operation would first have to check WEIGHT_VALID and then present either the value or an error message. Furthermore, the operation of adding weight information to the schema would have to be modified to set the "weight analysis data validity" value to true. This value would then have to be initialized with false upon creation of the data structure. A request for weight analysis would first check whether the data to be used are valid. This approach eliminates the possibility of algorithms using invalid data. However, it introduces the less obvious but (for this reason) possibly more serious problem of inconsistency. Let us assume that weight analysis has been performed once (hence the weight in the query subschema is valid); and let us assume that a new set of weight analysis data is input afterwards. As a consequence (except for some rare cases), the weight value in the query subschema is inconsistent with the weight analysis data; it still reflects an outdated situation.

The question whether results of intermediate operations (such as the weight analysis) should be stored in a system or regenerated whenever needed is fundamental. Storing intermediate results introduces a redundancy which helps to improve efficiency, as it saves the unnecessary repetition of operations. However, the same measure introduces the danger of using invalidated information if the origin of the intermediate results undergoes a change.

There are several methods to ensure the correct precedence of operations:

- *Method A:* Leave it to the user.
  In this case no security measure is taken to prevent the user from requesting a weight analysis without valid data, or to query for the weight value before it has been computed. The results of such requests are unpredictable. Nevertheless, in many cases this solution is acceptable. The user will probably be surprised by the result of an invalid request, and will recognize the mistake. Changes are often very small that the result would be so close to what the user expects that he does not recognize the problem. Obviously, this approach should be taken only when the user is experienced enough to judge correctly. In a more complicated situation this probably can not be justified.
- *Method B:* Modify the schema by adding a validity value, and add a validity initiation operation.
  This approach influences the subschemas for "input and query" and for "weight analysis data". Both subschemas would have to be enhanced as illustrated for HAMMER below.
- *Method C:* Add a "validity check" subprocess and a validity subschema.

The validity subschema might have the form

```
TYPE VALIDITY = RECORD
INPUT_DATA, WEIGHT_ANALYSIS_DATA,
WEIGHT, DRAWING_DATA : BOOLEAN INITIAL (FALSE)
        END;
```

The subprocess responsible for the validity of the data would be called upon at the beginning and end of other operations. At the beginning of an operation the validity subprocess would be asked whether all data to be used are valid; at the end the subprocess would be requested to set the validity of the new data to true and the

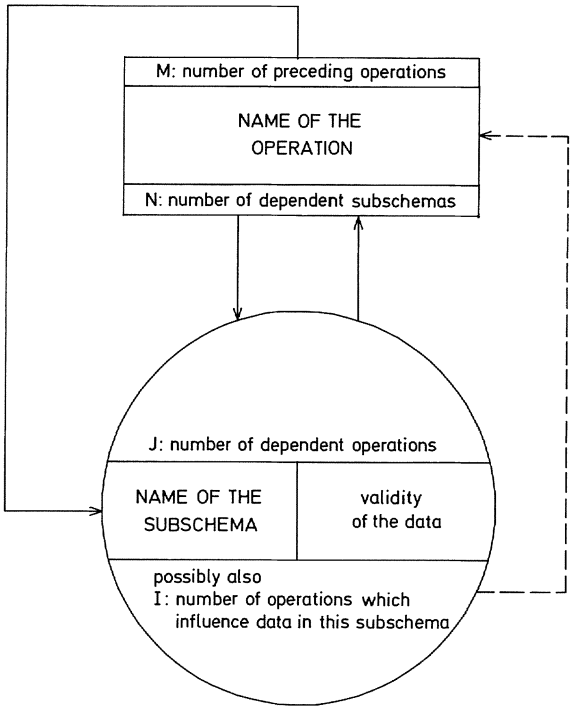


Fig. 3.33. A schema representing the precedence of operations

validity of all dependent data to false. In this case, the knowledge of precedence would have to be built into the validity check algorithm. In a more complicated situation this knowledge would preferably be built into a data structure, which may have a schema as represented in Fig. 3.33. The concept of implementing the dependency of data upon other data and preceding operations has been implemented in a knowledge-based approach for solid modeling and production planning by Kimura et al. [KIMU88].

3.3.5.2 The Information Packages

The ideas in the previous section lead us to consider another aspect of information in CAD systems: How should we group the information which is to be exchanged among the various subprocesses? In particular, how should we group the information which flows between user and system? Information is always transported in certain packages (for input, output, and storage). In the implementation, these packages may appear as statements, commands, or records. In the example used in this chapter it is not possible to modify the material properties for the hammer shaft without repeating the whole weight analysis – even for the head piece which remains unchanged. Similarly, the additional drawing information must be input as a complete set of data. It would not be possible to change the width of an individual line without regenerating the complete drawing data structure.

The aspect of how information should be grouped has been given much attention in the area of artificial intelligence. In particular, investigations related to chess playing by humans and computers [FREY78] have shown that a human stores and handles information not always in the most obvious way, but rather on various levels of abstraction in so-called information chunks. In chess play, for example, certain situations are not memorized as a set of pieces located on certain fields of the two-dimensional array on the chess board, but as a single chunk that represents the whole strategic situation. Communication between a human and a CAD system would probably be optimal if information packages corresponding to such chunks in the design task could be exchanged. On output, graphical representation of information comes very much closer to this ideal than lists of figures do.

Large information packages are quite adequate for batch processing, which is characterized by much work in the preparation of a complete and correct set of input data. In interactive processes, however, the information packages must be reduced to a size which is easily tractable by the user without great risk of mistakes during input, or misunderstanding or frustration during output. Thus, in interactive CAD systems, one is forced to break the information packages and hence the individual subprocesses down to small sizes. As a consequence, the number of subschemas and of subprocesses becomes large, and the problem of data validity and consistency becomes more important than in batch-oriented systems.

The information packages which are exchanged between the user of a CAD system and the system itself are not merely groups of information units. They must be formulated subject to certain rules. On input (to the system) they must be transformed into a machine-readable representation, on output (from the system) they must be transformed into a form which is perceptible by the operator (in most cases a visible form). The set of rules for the formulation and representation of the information packages constitutes the operator language of the CAD system.

One particular question in the design of the communication language is whether it should have a descriptive or more of a command character. For interactive communication, the general trend is to split the user input into a large number of commands containing relatively few descriptive data. The advantage of this approach is that it makes echoing and correcting input a relatively simple task. The commands may often be classified into the following groups:

- commands for building a model;
- commands for starting significant processing;
- commands for display of results; and
- commands for addressing system utility functions (such as "help").

A more descriptive type of language would tend to eliminate the commands for initialization of processing steps. A system based upon this approach would have to determine for itself what processing steps ought to be taken, on the basis of the types of results that are asked for. A precedence schema, as illustrated in Fig. 3.33, could be used in a CAD system to derive such information. An example of this kind of approach to CAD has been proposed by Pomberger [POMB82].

Modeling in CAD is a task which consists of the following main subtasks, which must be performed in a parallel and coordinated way:

- planning of the schema:
  - identification of the objects to be treated,
  - specification of the relations between the objects,
  - specification of the attributes of objects and relations,
  - specification of the allowable range of values for the attributes,
  - refinement of the schema, introduction of subschemas;
- planning of the process:
  - identification of the subprocesses,
  - specification of the subschemas required for the subprocesses, including the efficiency aspect,
  - specification of precedences among the subprocesses,
  - specification of the data validity requirements for each subprocess; and
- planning of the language:
  - specification of the information packages for communication between man and machine,
  - specification of the rules for formulating and representing these information packages.

3.3.6 Resulting Data Base Management System Requirements

The close interrelationship between the operations in a design process and the schema, poses special problems which do not arise (for instance) in commercial applications to the same extent. When computer methods are to be applied, the conceptual schema of the design process, or at least a subschema of it, will have to be represented on two levels:

- in the programs;
- on external storage media.

In the programs, a subschema representation takes the form of the set of declarations of records, arrays, and variables (specifying names and attributes) which represent the entities in the schema. For external storage, we find two solutions: file management systems or data base management systems (DBMS). Files contain records which represent the individual entities and relations, while the associated subschema is implicitly defined by the corresponding declaration of records in all programs which access the file. In a DBMS, however, the schema is explicitly stored as an integral part of the data base itself.

When we discuss the exchange of data between programs, we have to consider filing systems and DBMS's as alternatives. The DBMS provides several well-known advantages (see also Sect. 2.5):

- relief of the application program from details of data and storage space administration;
- relief of the application program from access efficiency considerations;
- reduced data redundancy;
- reduced redundancy of the schema representations;
- improved consistency;

- shared access by separate parallel processes; and
- improved data security.

These advantages do not imply that filing systems are completely outdated. They are still superior to a DBMS for rapidly accessing large amounts of data.

DBMS's have a wide-spread application in commercial data processing, while they are as yet less frequently found in a CAD environment. The reasons are mainly the following:

- In commercial applications, the number of entities of a single type is generally large, but the number of entity types is limited. In design, however, the situation is the opposite. The number of entity types is extremely large (how many different parts are there in a car?!) while there are usually just a few instances of each entity type;
- the relationships between entities are generally much more complex in a CAD schema (functional relationships, topological relationships, and geometric relationships) as compared to commercial data processing. In short, CAD requires that significantly more complex schemas be handled;
- most DBMS's tend to store data of similar type together (all points together, all circles together). CAD applications, however, need many varieties of data types at the same time. This causes long searches to bring together what belongs logically in the same context but has been scattered throughout the data base because of the grouping according to type. The consequence is inefficient processing;
- as described in Sect. 3.2.2 and illustrated by Fig. 3.13, design consists largely of the development of a schema. Only in detail design (variational design) can the schema be considered as fixed. Otherwise, the schema is subject to continuous development, while in most DBMS's the schema is assumed to have a relatively long lifetime without modification. As a consequence, DBMS's which are suited for commercial data processing applications may be expected to satisfy the requirements of CAD for variational design, but not necessarily for the whole design process.

Hence, a data base management system for CAD application must provide facilities for

- handling very complex schemas with up to a thousand different types of entities, or even more;
- allow for frequent modification of the schema, with the associated necessary transformation of the previously defined data base content into the new schema.

3.4 Summary

This chapter had two main purposes. First, we introduced a concept (or a model) of the design process which can be used as a basis for describing CAD processes. Second, we introduced terminology which will be used for describing design processes, CAD processes, and CAD systems. A key issue was the structuring of the design pro-

cess into the activities of specification, synthesis, analysis, evaluation, and presentation. These activities are centered around a conceptual model of the objects to be designed. However, when the design process starts, the conceptual model is only partly determined (namely by the specification). It will be developed, refined, and possibly modified as the design proceeds in the synthesis activity. Analysis does not change the conceptual model, but rather determines values for object attributes in the model. Evaluation compares the results with the specified goals, and influences the synthesis activity. The achievement of the design is presented to the higher-level process from which the specification was issued.

The important concept of the environment was introduced. The environment is considered as a process in which the design process is embedded. The environment provides the required resources for the design process and coordinates resource requests if more than one process is executed in parallel. Learning was identified as an activity of the environment.

The fundamental difference between a human-based design process and CAD processes is the need for formal representation. While man can work on an informal basis, CAD is possible only if the conceptual schema and the rules to be applied in synthesis and analysis have been formalized. This requirement applies both to conventional CAD systems and to systems based on artificial intelligence (AI) methods. While in conventional CAD the synthesis is mainly the task of a human operator, AI methods offer the chance to transfer this task to the CAD system. The reason for this difference stems from the fact that conventional CAD systems are based on the common programming languages and the algorithmic approach to problem-solving that underlies them. Furthermore, conventional data base management systems and programming languages generally consider the determination of a schema as a task which has to be completed before any other operation using the schema. In synthesis, however, we apply rules which are not in algorithmic form and we should be able to manipulate schemas freely. For this reason, AI methods have a potential for progress in the synthesis area of CAD, but more research is needed before such methods are ready for production.

Finally, we investigated modeling in CAD. Modeling, or developing a schema, is a task which cannot be performed by simply studying the anticipated properties of the objects to be designed (although this is one prerequisite). The schema and its various subschemas are significantly influenced by the operations. Along with functional aspects, the consideration of efficient utilization of the available resources (manpower being the most precious) will influence schema planning. As a third constituent (besides schema and operations), we have identified the CAD system language. CAD system design requires the development of these three constituents in parallel.

3.5 Bibliography

[ALL_78] J. J. Allan III, K. Bø: A Survey of Commercial Turnkey CAD/CAM Systems. Dallas Productivity Int. Corp. (1978).

[ALLA78] J. J. Allan III: CAD in the U.S. and in Europe. In: Tech Report GRIS 78-3, Fachgebiet Graphisch-Interaktive Systeme. TH Darmstadt (1978).

[BAUM82] H. G. Baumann, K.-H. Looscheelders: Rechnerunterstütztes Projektieren und Konstruieren. Heidelberg, Springer-Verlag (1982).

[BRIN73] P. Brinch Hansen: Distributed Processes: A Concurrent Programming Concept. Computing Surveys 5, 4 (1973), pp. 223–245.

[BRIN75] P. Brinch Hansen: The Programming Language Concurrent Pascal. IEEE Trans. Softw. Eng. 1, 2 (1975), pp. 199–207.

[BRUN56] J. S. Bruner: A Study of Thinking. New York, John Wiley & Sons (1956), pp. 232.

[DIJK68] E. W. Dijkstra: Cooperating Sequential Processes. In: F. Genuys, Programming Languages. New York, Academic Press (1968).

[ECKE77] K. Ecker: Organisation von parallelen Prozessen – Theorie deterministischer Schedules. Reihe Informatik 23, Bibliographisches Institut Mannheim, B. I.-Wissenschaftsverlag (1977).

[FREY78] P. W. Frey: Chess Skill in Man and Machine. Heidelberg, Springer-Verlag (1978).

[GIES85] E. Giese, K. Görgen, E. Hirsch, G. Schulze, K. Truöl: Dienste und Protokolle in Kommunikationssystemen – Die Dienst- und Protokollschnittstelle der ISO-Architektur. Heidelberg, Springer-Verlag (1985).

[GROT76] G. Grotenhuis, J. van den Broek: A Conceptual Model for Information Processing. Inß G. M. Nijssen, Modelling in Data Base Management Systems. Amsterdam, North-Holland Publ. Co. (1976), pp. 149–180.

[GRAB79] H. Grabowski, M. Eigner: Anforderungen an CAD-Datenbanksysteme. VDI-Z 121, 12 (1979), pp. 621–633.

[HANS76] F. Hansen: Konstruktionswissenschaft – Grundlagen und Methoden. München, Hanser-Verlag (1974).

[HATV77] J. Hatvany, W. M. Newman, M. A. Sabin: World Survey of Computer-Aided Design. Computer Aided Design 9, 2 (1977), pp. 79–98.

[HATV73] J. Hatvany: The Engineer's Creative Activity in a CAD Environment. In: J. Vlietstra, R. F. Wielinga (eds.), Computer-Aided Design. Amsterdam, North-Holland Publ. Co. (1973), pp. 113–126.

[HOFS80] D. R. Hofstadter: Gödel, Escher, Bach: An Eternal Golden Braid. New York, Vintage Books (1980), pp. 24–27.

[KIMU88] F. Kimura et al.: Representation of Design and Manufacturing Process by Data Dependency. IFIP WG 5.2, Workshop on Intelligent CAD Systems, Cambridge (1988).

[KOLL76] R. Koller: Konstruktionsmethode für den Maschinen-, Geräte- und Apparatebau. Heidelberg, Springer-Verlag (1976).

[KRAU77] F.-L. Krause, V. Vassilakopoulos: A Way to Computer Supported Systems for Integrated Design and Production Process Planning. In: J. J. Allan III, CAD Systems. Amsterdam, North-Holland Publ. Co. (1977), pp. 5–34.

[LATO78] J.-C. Latombe: Artificial Intelligence and Pattern Recognition in Computer Aided Design. Amsterdam, North-Holland Publ. Co. (1978).

[LUML82] J. Lumley: Expert Systems. Systems International 6 (1982), pp. 53–56.

[MIS__81] J. Misra, K. Mani Chandy: Proofs of Networks of Processes. IEEE Trans. Softw. Eng. SE-7, 4 (1981), pp. 417–426.

[NIJS76] G. M. Nijssen: A Gross Architecture for the next Generation Database Management. In: G. M. Nijssen, Modelling in Data Base Management Systems. Amsterdam, North-Holland Publ. Co. (1976), pp. 1–24.

[NYGA80] K. Nygaard, P. Nandlykken: The System Development Process. In: K. Hünke, Software Engineering Environments. Amsterdam, North-Holland Publ. Co. (1980), pp. 157–172.

[PAHL77] G. Pahl, W. Beitz: Konstruktionslehre. Handbuch für Studium und Praxis. Heidelberg, Springer-Verlag (1977). "DIN 66253 Teil 1": Programmiersprache PEARL. Basic PEARL. Berlin, Beuth (1981).

[PEAR81] DIN 66253 Teil 1: Programmiersprache PEARL. -Basic PEARL. Berlin, Beuth (1981).

[POMB82] G. Pomberger: Ein Modell zur Simulation von Konstruktionsprozessen. Angewandte Informatik 1 (1982), pp. 26–34.

[RODE76] W. G. Rodenacker: Methodisches Konstruieren. Konstruktionsbücher Bd. 27, 2. Aufl., Heidelberg, Springer-Verlag (1976).

[ROTH71] K. Roth, H. J. Franke, R. Simonek: Algorithmisches Auswahlverfahren zur Konstruktion mit Katalogen. Feinwerktechnik 75 (1971), pp. 337–345.

[SALT78] J. H. Saltzer: Naming and Binding of Objects. In: G. Goos, J. Hartmanis (eds.), Operating Systems. Lecture Notes in Computer Science, Vol. 60. Berlin, Springer-Verlag (1978), pp. 99–208.

[SCHL78] E. G. Schlechtendahl: Rules for Designing CAD Software Machines. Proceedings of the International Conference "Interactive Techniques in Computer Aided Design". Bologna, Italy (1978).

[SCHU78] C. Schuenemann: Speicherhierarchie-Aufbau und Betriebsweise. Informatik-Spektrum 1, 1 (1978), pp. 25–36.

[SIMO68] R. Simon: Rechnergestütztes Konstruieren. Dissertation TH Aachen (1968). Amsterdam, North-Holland Publ. Co. (1977), pp. 5–34.

[SUSS78] G. J. Sussman: SLICES: At the Boundary between Analysis and Synthesis. In: J. C. Latombe, Artificial Intelligence and Pattern Recognition in Computer Aided Design. Amsterdam, North-Holland Publ. Co. (1978), pp. 261–299.

[VDI__69] VDI 2221: Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte. Düsseldorf, VDI-Verlag (1985).

[WAEC69] R. Wächtler: Die Dynamik des Entwickelns (Konstruierens). Feinwerktechnik 73 (1969), pp. 329–333.

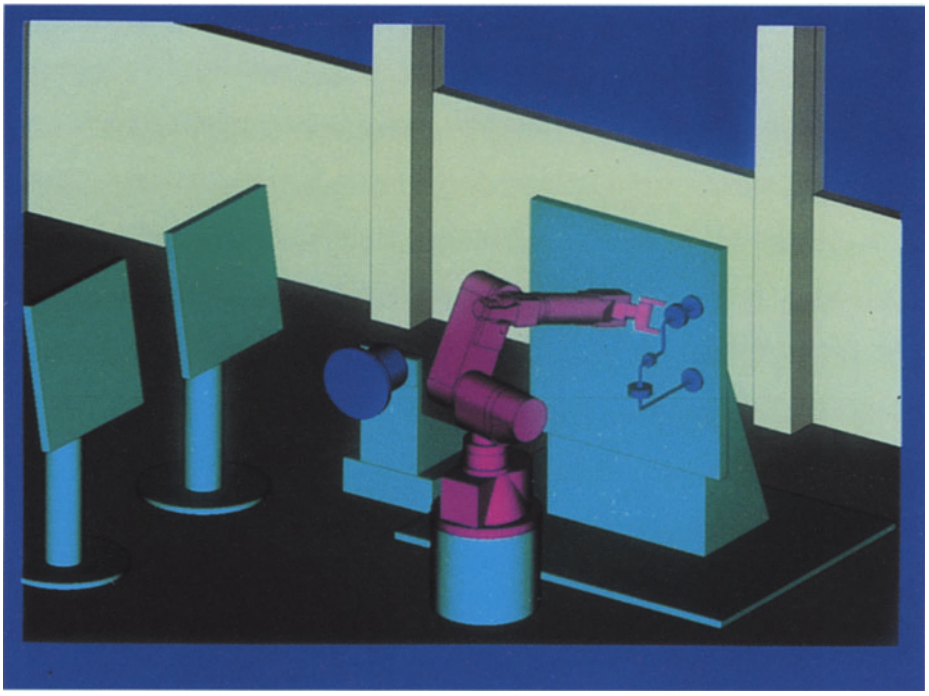
[WARM78] E. A. Warman: Computer Aided Design: An Intersection of Ideas. In: J. C. Latombe, Artificial Intelligence and Pattern Recognition in Computer Aided Design. Amsterdam, North-Holland Publ. Co. (1978), pp. 1–18.

[WINK79] J. F. H. Winkler: Das Prozeßkonzept in Betriebssystemen und Programmiersprachen I. Informatik Spektrum 2, 4 (1979), pp. 219–229.

[WINK80] J. F. H. Winkler: Das Prozeßkonzept in Betriebssystemen und Programmiersprachen II. Informatik Spektrum 3, 1 (1980), pp. 31–40.

[ZUSE80] K. Zuse: Petri-Netze aus der Sicht des Ingenieurs. Braunschweig, Vieweg & Sohn (1980).

4 The Architecture of CAD Systems



Simulation of a robot model
(courtesy of Kernforschungszentrum, Karlsruhe, Germany)

4.1 The Gross Architecture

4.1.1 Components

Just as it would be difficult to define "the typical program" or "the typical house", there is no such thing as "the typical CAD system". The architecture of a particular CAD system will certainly depend on:

- the tasks to be solved by the system;
- the computer resources available for its implementation (both hardware and software);
- the experience of the CAD system designer;
- rules established within the company or on a wider scale, which restrict the freedom of the system designer.

Graphical representations of CAD systems depend very much upon the aspects which their authors wished to emphasize. The representation given in Fig. 4.1 is based on [NOPP77]. This representation emphasizes the time sequence of the execution of several CAD programs as the design process proceeds through its various phases. The representation shown in Fig. 4.2 relates the main system components to the major activities in a design process as described in Chap. 3:

- specification,
- synthesis,
- analysis,
- transformation,
- presentation, and
- evaluation.

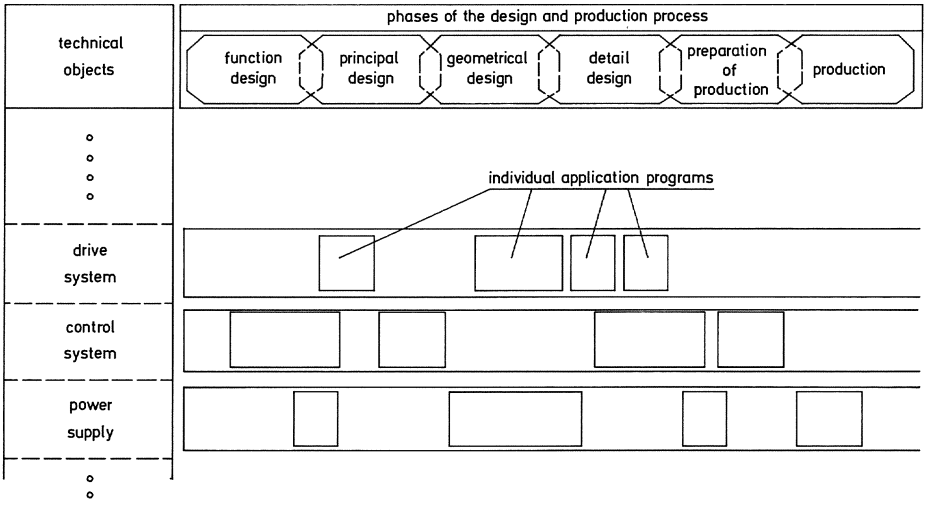


Fig. 4.1. The program chain schema of CAD systems

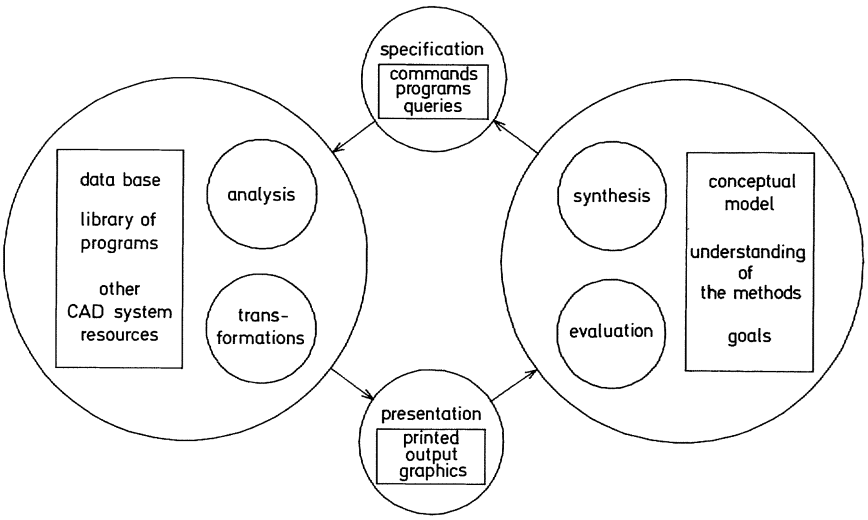


Fig. 4.2. CAD activities and CAD system components

The specifications are passed from the system user (the operator) to the CAD system as commands, queries, or programs, expressed in a language that is understood by both partners (man and machine). Within the CAD system, analyses and transformations are performed accordingly. The state of the process is represented in the data base, while the program libraries and other system resources are used during execution. Results are presented in graphical or printed form to the operator who performs the evaluation by comparing the results with the goal. The operator (usually) performs the synthesis task as well. For this purpose he must have a conceptual model of the objects handled by the CAD system, and of the state of the CAD system process. He must also have a conceptual understanding of the methods that are implemented in the system as programs, and he must know how these methods affect the objects. With this knowledge, he can formulate the specification of the next steps to be taken and send this request to the CAD system. In certain cases, however, the synthesis task is at least partially located within the CAD system. For special products in a special environment, it may be possible to formulate a synthesis algorithm which can be programmed. Furthermore, systems based upon artificial intelligence methods attempt to support synthesis in a wider range of problems [LATO78]. But the standard case corresponds to the situation shown in Fig. 4.2.

The components of a CAD system may be viewed in various aspects. The functional aspect relates the system components to the model of the design process, the hardware aspect concentrates on pieces of equipment, while the software aspect deals with programs and data. There is typically (but not necessarily) a correspondence between the functions and the hardware/software components which perform them. Table 4.1 summarizes this correspondence.

Table 4.1. Correspondence between functional, hardware, and software aspects

Functional	Hardware	Software
knowledge, memory	peripheral storage devices: disks, tapes, etc.	program libraries, data files, data base systems
state representation of process	same as above	data files, data base systems
analysis	central processor and primary memory	programs, modules, program packages
man → machine communication:	terminal, keyboard function keys, menu cursor, tablet, mouse	character strings, text editor, commands case constructs, pointers
control and data control identification		
machine → man communication:	printer, terminal, display (alpha-numeric and graphical) indicator lights	character strings, menu techniques, prompting, graphical representations
inquiries, messages		
presentation of:		
text	printer, terminal, COM	WRITE statements
pictures	plotter, graphics display COM	graphics packages
mixed text and pictures	plotter, graphics display	graphics package
information transport	mail (conventional and electronic), networks	mail service (conventional and electronic), file transfer
evaluation, synthesis	person	human brain

4.1.2 Interfaces

4.1.2.1 Development and Installation of a CAD System

Before a CAD system can be used it must be developed and installed. In the case of turnkey systems [ALLA78], these activities are reduced to the analysis of commercially available systems, and the evaluation of how well they will suit the actual needs. In other cases the initial implementation and the gradual extensions of a CAD system uses components of the environment – and sometimes components of the system itself – in what is known as "bootstrap" technique (Fig. 4.3). In designing, implementing, and enhancing CAD systems, such techniques very effectively reduce the time and cost of system development and maintenance.

4.1.2.2 The Invocation of a CAD System

Let us assume that a CAD system has been installed in a computer environment. Before he can operate with the CAD system on a given project (new or old), the designer must invoke the CAD system: he must bring it to execution. In order to do

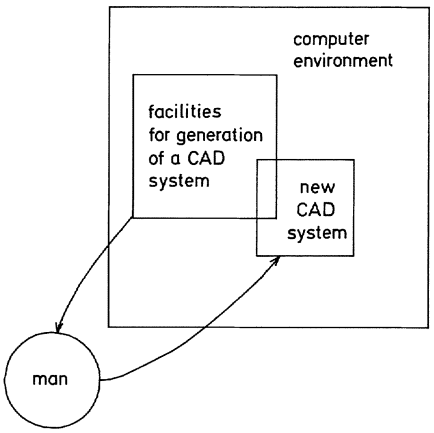


Fig. 4.3. The implementation of a new CAD system in an environment

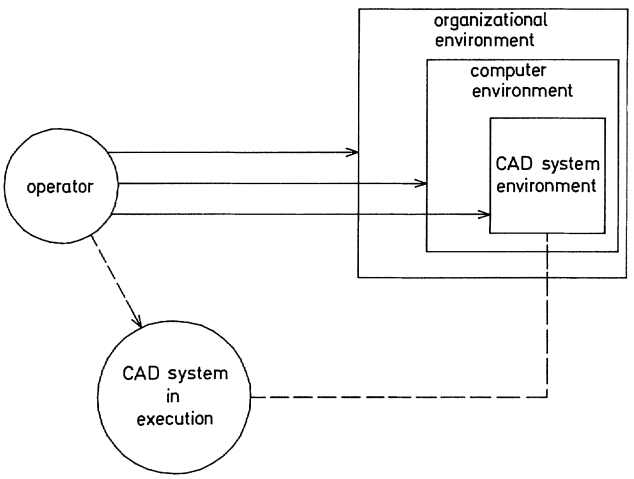


Fig. 4.4. Interfaces during invocation of a CAD system

this, the designer (whom we call the operator while he is using the CAD system) must deal with a number of obstacles before he can address the CAD system itself. There may be organizational barriers (the access to the CAD system may require him to move physically to another room or another building), but in any case there will be barriers due to the computer environment.

If he uses a central computer, both the CAD system and the data structure representing the state of his CAD process may be on resident storage devices, and the obstacle is merely a language barrier: he must use a limited subset of the computer's operating system control language to hook the CAD system both to the data (files) of the process and to the communication channels (terminals) he wants to use. If he uses a dedicated CAD computer in his office, he may even have to deal with hardware: he may need to load a couple of floppy disks (one for the CAD system, one for his data) and perform some manipulations to start the computer. Compared to the large

central computer, he may find that the operating system of his dedicated computer is much easier to use. In any case the communication interface of the computer environment always constitutes a part of the interface between the operator and the CAD system (see Fig. 4.3). Only after the invocation of his CAD system can the operator address the system itself (Fig. 4.4).

4.1.2.3 Functional Interfaces in a CAD System

Figure 4.2 is an overview of the main functional components of a CAD process during execution. The representation, however, is not very helpful with respect to the interfaces between system components. Either the interface is not shown, or it is represented simply by a solid line. But a solid line between two circles can mean almost anything. We will now investigate in more detail the interfaces between some CAD system components, although such a detailed analysis complicates the picture considerably (even if we concentrate on just a few components). A graphical representation of the interfaces among all components would become unreadable. When investigating an interface between components, we will apply the following consideration:

An interface between two components represents the use of a shared resource according to one or more agreed-upon schemas.

Thus, in order to discuss an interface between two components, we must say something about this common resource and about the related schema. Note that the two components do not necessarily have to use the same schema. In a general situation, particularly in open systems configurations [OSI__78], the interface resource may provide a schema transformation allowing each of the interfacing components to use a different schema to represent the same information. The situation occurs quite frequently when data are exchanged between computers of different manufacturers, which use different coding for data representation (see Chaps. 4.3.4 and 7). Except for the extra schema transformation, which may require some additional information for correct operation, there is no basic difference from the situation where both partners use the same schema. For this reason, we will discuss only the simple case where one schema is used by both communicating partners.

We can use Fig. 4.5 as a model for discussing the execution of a CAD system. The figure concentrates on the interfaces between the main functional components. For the moment, let us ignore the box on the left-hand side of the figure. As in Fig. 3.29, the CAD system programs (which perform the analytical work) and the subschema transformations play a central role. They operate on an internal representation of the process state by using primary memory (with computer words as the resource), according to a certain subschema. This subschema is defined by the declaration of some global data structure in the programs. For long-term saving of the process state, a data base management system is used. The rules for transformation of the schema into the data base's internal representation are carried out within the data base management system; the data definition language of the system has been used to define this transformation. All CAD execution programs and the communication processor use the command subschema of the data base to access data in the data base system.

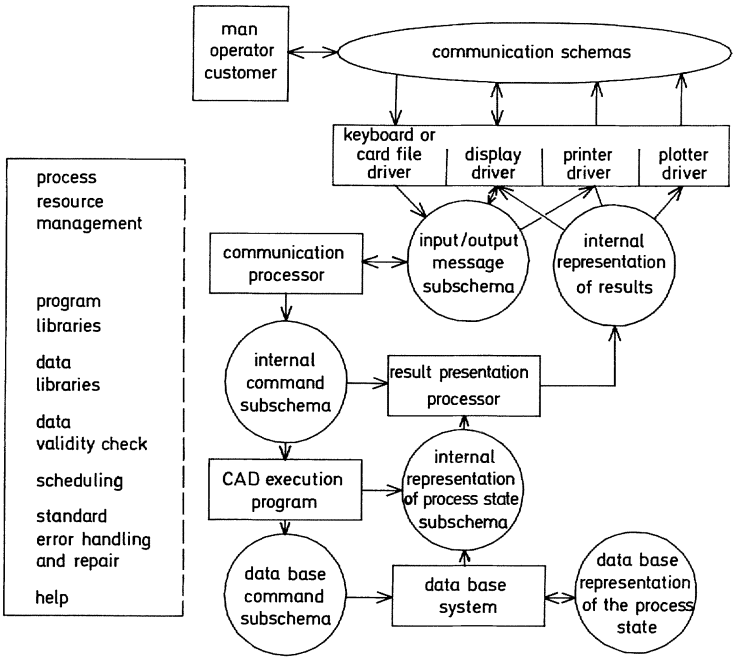


Fig. 4.5. Interfaces between main CAD system components

Transformation of process state data from the data base into an internal representation takes place in a similar way.

The internal state representation is also the interface between the CAD programs and a result presentation processor (one or several), which produces an internal representation of results suitable for output on a number of devices. The output device processors (printer, plotter, display) will then transform the results into optically perceptible information in a schema as required by the operator or other persons (a customer, for instance).

Communication from the operator to the CAD system is mainly in the form of a conversation between the operator and a communication processor (interpreter, compiler) within the CAD system. The purpose of this communication is to build up or to change the internal state representation (the "model"), and to inquire about the actual model state [LILL81]. If more than one input device is to be used, it is advisable to have a common interface for representing messages (in either direction) between the drivers of the input device and the actual communication processor. Often a character string representation is used as a common command schema for all input devices. Input originating from any input device is first converted to the character string representation, and can then be processed by the command interpreter in just the same way as if it had come from the keyboard. Part of the conversation will take place in a local way (errors in basic input syntax can be detected by the device drivers, and correction will be requested so that only "legal" commands are passed to the communication processor). The communication processor controls the execution of

the CAD programs by passing information from the input to them in a special interface.

The schematic described so far represents what might be called the "abstract CAD machine". Only the main functions are considered here. All processes within the executing CAD system, however, need to be supported by a number of utility functions which designers of CAD systems tend to hide from the users. The resource management in the CAD system provides the necessary environmental conditions for the successful operation of the "concrete CAD machine".

4.1.2.4 Man-Machine Communication Channels

From Fig. 4.5 it is evident that several communication processes will occur between the operator and the CAD system at any given time. Problems may arise if these communication processes share the same resource for visualization. In the extreme case where there is only a single display available, the input echo and computer-operating system messages – among others – compete with each other for the limited display space (in printer output all these processes may share the same paper, but at least they do not have to compete with each other for the lines). In order to avoid severe problems, it has become a good practice to use at least two separate visualization surfaces, one for the presentation of results and one for the messages of the communication processes, including the input echo. An ergonomically better alternative is the use of multiple windows on the same display screen. Examples of solutions to this problem of conflict are listed in Table 4.2.

Table 4.2. Use of available hardware for man-machine communication

Hardware available	Used for message visualization	Used for data presentation
printer	printer	new page before and after blocks of results
printer + plotter	printer	plotter
keyboard + display	display	display
refresh display	display, possibly limited message area	remaining area of display
two displays	one of the displays	the other display

4.1.2.5 Data Transfer Interfaces of CAD Systems

With the use of CAD systems becoming more and more abundant throughout industry the communication interface between CAD systems and between CAD and subsequent phases of the industrial production process (manufacturing, assembly, quality assurance, and maintenance) has become a key issue. This topic will be dealt with in more detail in Chapter 7.

4.1.3 CAD Tools and CAD Machines

Even in an established CAD environment, one is likely to be faced with CAD tasks that cannot be catered for satisfactorily by any of the CAD systems available in that environment. Generally one finds certain components in the systems which (at least in principle) could be used beneficially. The alternatives are: either to tackle that design problem by conventional means, or to buy a new CAD system for that purpose, or to extend the functional capabilities of one of the available systems by adding new software.

Considerable economic savings are possible if, for a new CAD task, an existing CAD system can be used (possibly with some adaptation) or – if some amount of new development cannot be avoided – the work of development effort can be reduced by utilizing available tools. The necessity for and the benefits of such tools were pointed out by Hatvany [HATV77]. Two categories of such tools may be distinguished:

- tools which are used only during the CAD system development; and
- tools which become part of the CAD system and will be used during its operation for extending its capabilities. We will call these tools "machines".

4.1.3.1 Tools Used in CAD System Development

In an environment where the development of CAD systems is a routine job, one is likely to find special software systems for this work. These tools include:

- specification aids,
- testing aids,
- documentation aids,
- precompilers, and
- program generators.

Although the specification is (or should be) completed before the development of a CAD system is started, while the documentation is performed in parallel with the development, specification aids are often suited for documentation purposes as well. This applies particularly to relatively informal methods like SADT [ROSS76] or PSL/PSA [TEIC77]. Such systems are already being used for production purposes. Formal specification methods based on abstract data types [GUTT77] or on "traces" [BART78] are still undergoing rapid development. For a survey of specification and planning methods, see [LUDE78], [BALZ81], [HESS81]. Special tools for testing have also been developed [VOGE80].

Precompilers usually serve two purposes. They may guarantee the consistency of data declarations in different programs which are intended to operate on the same objects. For this purpose, data declarations are retrieved from a data base and inserted into the programs (in the most simple case, by including a piece of declarations text or by expanding macros). The second purpose is to permit the algorithms to be written in a language that is better suited to express the operations, or simply shorter than

the available programming language. The higher-level program text is then compiled into a language for which standard compilers are available (e.g., FORTRAN, etc).

Examples of collections of software tools are the National Bureau of Standards Software Tools Database [HOUG80] and the summary reported in [HUEN80].

4.1.3.2 Tools Used in CAD System Extension

Typical low-level components which one would choose off-the-shelf to combine into an operable CAD system are:

- a data base management system;
- a file management system;
- a program package for handling common data structures in primary memory;
- a command interpreter for character-string type commands and for interactive graphic communication;
- a graphics package; and
- mathematical subroutine packages.

One would also like to select from an available set of higher-level tools for such tasks as:

- finite element analysis;
- two-dimensional geometry handling;
- computer-supported drafting;
- three-dimensional geometry handling;
- smooth surface design; and
- hidden line removal.

It is a well-known problem in the world of CAD that even if such machines are available (and quite often they are), it is a major task to put them together into a satisfactory or even barely functional system. They just do not fit together. This is one reason for the continual rewriting of programs all over the world. The question is obvious: Is it possible to design such software machines so that they can be put together into an operational system, in any arbitrary combination, more easily than the present state of the art allows? We will devote Sect. 4.3.2 specifically to this question.

Many of the points to be discussed are not new and may be found in other areas. In particular a technically oriented reader should note the similarity between the software machine concept developed here and real machines in an industrial process.

4.2 Data Models

4.2.1 Mapping

4.2.1.1 The Ideal Situation

We will assume that a conceptual model of the objects to be handled in a CAD system has been derived. We will further assume that this model satisfies all functional requirements, that is, it is suitable for all operations to be performed on a conceptual level. Finally, we have defined which information the system should receive from the user and supply to the user, and how this information is to be packed into information packages (see Sect. 3.3.5.2). We are now faced with the problem that a computer does not operate in the abstract space of the abstract data types [LISK75]. The objects of abstract data types must somehow be mapped onto the hardware of the computer. The same statement applies to the operations to be performed with the objects, and to the information packages. In practice, the mapping from abstract space to hardware is not done in a single step. An intermediate mapping level (at least one) is introduced. The intermediate level is the level of a language which can be understood by both a person and the machine. The closer this language is to the concepts used in the abstract space, the more easily a person can use it. This is the level of "high level languages" or "problem-oriented languages". Of course, it would be advantageous if only one intermediate mapping is required. Figure 4.6 illustrates this situation.

In the ideal situation, a person would only have to deal with the mapping between the conceptual level and that of the high-level language. The mapping from this level

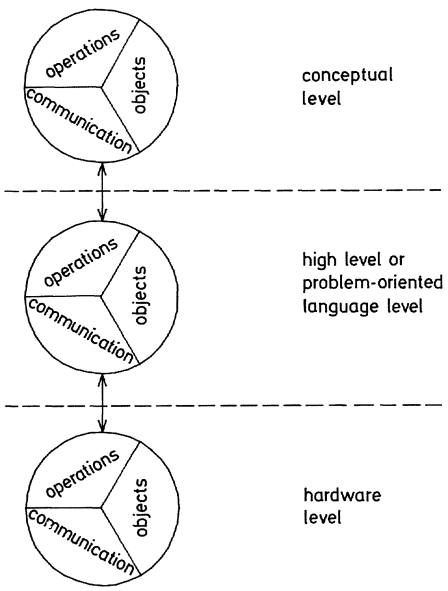


Fig. 4.6. The ideal two-step mapping between the conceptual world and hardware

to the hardware level would be hidden in the language compilers and the operating system. In reality, the ideal situation is rarely found. This is particularly true when FORTRAN is considered as the main (or perhaps the only) available high-level language. Since FORTRAN is still the most common programming language for CAD, we cannot simply ignore the deviations from the ideal. We have to identify such deviations in order the make them evident.

4.2.1.2 Reasons for Non-Ideal Mapping

There are two basic reasons for deviations from the ideal situation: inadequate functional capabilities of the high-level language, and loss of efficiency of the system execution process. The following examples will illustrate these cases.

(1) SCALAR TYPES IN FORTRAN. Let us assume that we wish to express a calendar date on the language level. With PASCAL this could be done as follows [JENS78]:

```
type DATE= record
DAY: 1: 31;
MONTH: (JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC);
YEAR: 1900:2200
end;
```

Thus all calendar dates from Jan. 1, 1900 through Dec. 31, 2200 are mapped from the conceptual model space into the language level. The reverse mapping, however, might produce some illegal calendar dates such as Feb. 30, 1930, which makes the above record definition somewhat unsatisfactory and indicates the limitation of even advanced programming languages like PASCAL or C.

In FORTRAN, however, the situation is worse, since only a very limited number of data types is available (basically INTEGER and REAL). Hence we must map all abstract data types onto these few data types. A common way of doing this is to introduce an intermediate mapping between the abstract level and the language level. Two possibilities are shown in Figs. 4.7a and 4.7b using the data type "date" as an example. In the case of Fig. 4.7a, we map day, month, and year onto one integer each. In Fig. 4.7b, we define the year 1900 as a reference (mapped onto the integer value 0) and count the days from 1 through 366. Thus all calendar dates between Jan. 1, 1900 and Dec. 31, 2200 find their mapping on the language level, but the reverse mapping is really inadequate. A rather complicated algorithm is required to retrieve from the single integer value the separate information of day and month. The inverse mapping may not even be unique: other objects of the conceptual space may also have been mapped onto integers. Considerable confusion and significant economic losses for program debugging arise from this deficiency in FORTRAN. Due to a programming error, it may easily happen that "green" (mapped onto integer 3, say) is added to the "third of February" (mapped onto integer 34) with unforeseeable consequences.

(2)STORAGE SPACE CONSIDERATIONS. We are still using the above example, but will now introduce the resource aspect of storage space. Mapping a calendar date onto two integers (as in Fig. 4.7b) instead of three immediately saves one third of the storage space. This statement may be derived from even a rough understanding of

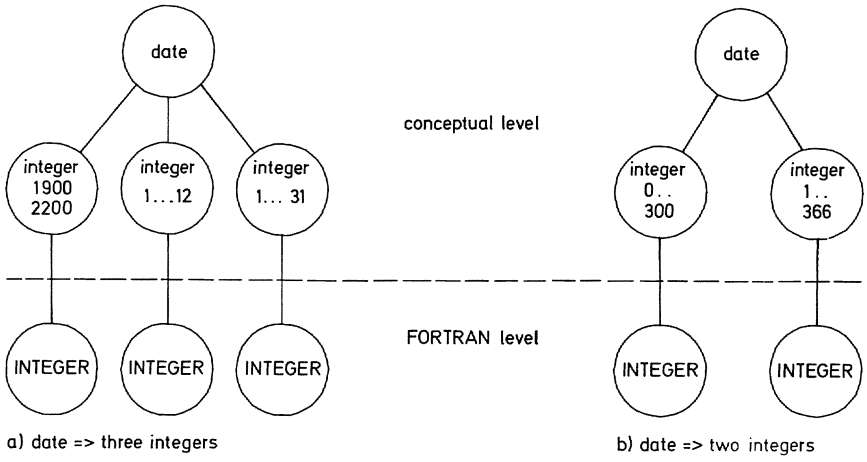


Fig. 4.7. The mapping of calendar dates (as an example) onto integers

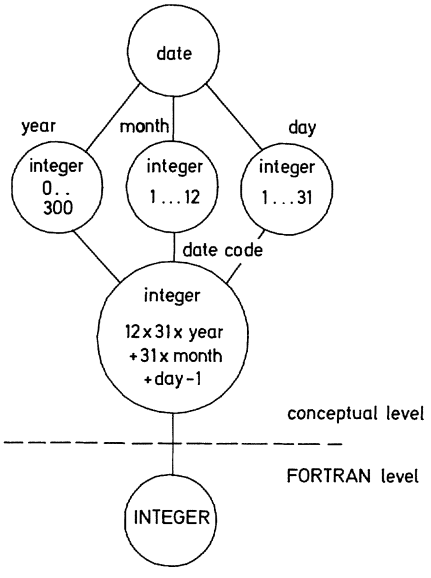


Fig. 4.8. The mapping of calendar dates onto a single integer

compilers, which tells us that every integer quantity needs one computer word for storage. The same line of argument may be carried further; mapping of the calendar date onto a single integer (as indicated in Fig. 4.8) will save two thirds of the storage space, as compared to Fig. 4.7a. The saving in storage space must, however, be paid for with extra processing cost whenever the normal calendar representation (month, day, year) has to be retrieved from the single integer.

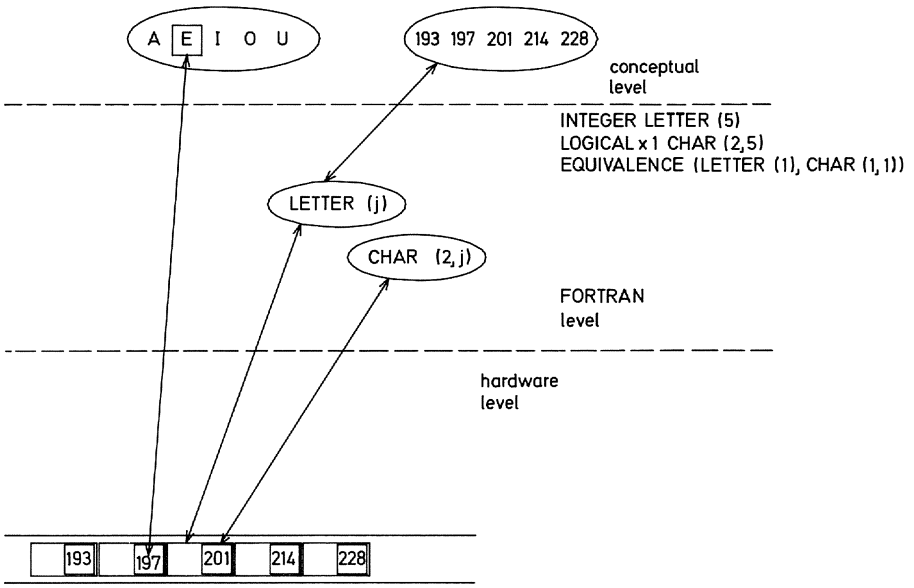


Fig. 4.9. Deficiencies of programming languages cause mapping around the language

4.2.1.3 Mapping Around the Language

A most peculiar situation arises if the high-level language is totally unsuited for mapping certain objects, while adequate support could be provided on a lower level (assembly language and hardware). The system designer is then tempted to map around the language level in order to achieve what he needs despite the properties of the high-level language. A typical example that is very important for CAD is the lacking capability of FORTRAN to specify record structures (ordered sets of data values of different types). The use (or rather mis-use) of EQUIVALENCE for this purpose is a poor and dangerous way to map around the limitation of that language immediately onto the computer storage space. We assume a version of FORTRAN which does not support character handling to illustrate such mappings. Let us write a program which reads characters, one at a time, and checks for the occurrence of A, E, I, O, or U. We assume that the computer has a byte-structured memory and that INTEGERs are stored in two bytes, LOGICAL*1 data in one byte.

Most FORTRAN programmers know how the compiler will map arrays of integers and logicals onto a sequence of computer storage words by bytes. They also know that reading a character will result in setting certain bits in a storage byte of the computer, and that this sequence of bits might be interpreted as an integer. Using this knowledge, the program designer can now map the character of the conceptual level onto computer storage bytes (see Fig. 4.9): he maps the bytes onto a two-dimensional array of logicals (CHAR) which – using an EQUIVALENCE – is embedded in an array of INTEGERs. The INTEGER array corresponds to a sequence of INTEGERS in the conceptual world. Instead of testing for 'E', which is not possible in our as-

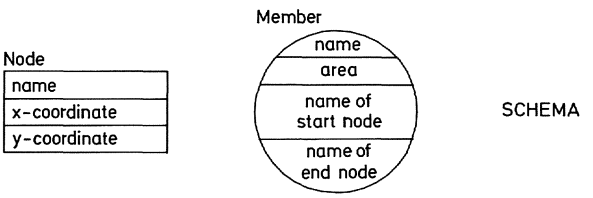
sumed FORTRAN, the program might test for 197. Note: the fact that we have included this example does not imply that we recommend the procedure described. However, the technique of mapping around the language level is common practice especially in FORTRAN programming, and in fact is sometimes unavoidable. Experienced FORTRAN programmers can do almost everything in this language, particularly if a couple of FORTRAN callable Assembler routines are added. Of course, it would be much better to use a language which provides the necessary features without requiring techniques – such as EQUIVALENCE – which are known to be a frequent source of error.

4.2.1.4 Mapping Between Aspects

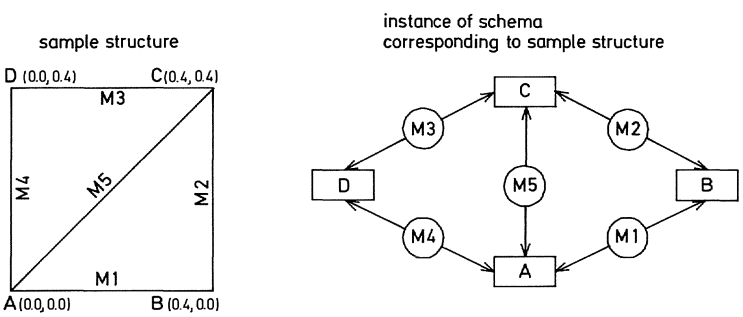
We will use a plane truss to illustrate the mapping between the aspects of operation, representation and communication. On the conceptual level, we characterize the geometry of a plane truss by

- the set of nodes (node name and location for each node); and
- the set of frame members (member name, names of starting node and end node for each member, cross-sectional area).

The location is always assumed to be given by X and Y coordinates in a unique Cartesian coordinate system. The corresponding schema is graphically represented in Fig. 4.10a together with a sample truss structure (Fig. 4.10b). We assume that a DBTG data base management system as described in [OLLE78] is to be used for implementation purposes. In the DBTG data definition language, the schema definition would be as follows:



a) Graphical representation of the schema



b) Two representations of a sample structure

Fig. 4.10. Sample schema: plane truss

```
RECORD NAME IS NODE;
LOCATION MODE IS SYSTEM;
WITHIN MODEL;
02 NAME; TYPE IS CHAR 20;
02 X; TYPE IS REAL;
02 Y; TYPE IS REAL;
RECORD NAME IS MEMBER;
LOCATION MODE IS SYSTEM;
WITHIN MODEL;
02 STARTNODE; TYPE IS CHAR 20;
02 ENDNODE; TYPE IS CHAR 20;
02 AREA; TYPE IS REAL;
```

The program which will perform the design calculations for the truss structure is assumed to be written in PASCAL. Although PASCAL supports sets, we prefer to map the two sets (of nodes and members) onto a list structure (a simple queue in this case). In the program, the names are mapped onto pointers, while the original character string names appear as additional attributes, and the coordinates are real values. The schema representation in the PASCAL program would read as follows:

```
type NODE = record
    NEXT_NODE       : →NODE;
    NAME            : array [20] of char;
    (X,Y)           : real
                      end;

type MEMBER= record
  NEXT_MEMBER       : →MEMBER;
     NAME           : array [20] of char;
  (START_NODE,
     END_NODE)      : →NODE;
     AREA           : real
                      end;

type SET_OF_NODES       : →NODE;
type SET_OF_MEMBERS     : →NODE;
```

With respect to communication, we distinguish between input and output. For input, each node can be mapped onto an 80 character record beginning with NODE, followed by the node name and the two coordinate values. A member is mapped onto a similar record beginning with MEMB and followed by the member name and the two node names. The input for a structure as shown in Fig. 4.10b would be as follows:

```
NODE A 0.  0.
NODE B 0.4 0.
NODE C 0.4 0.4
NODE D 0.  0.4
MEMB M1 A B
MEMB M2 B C
MEMB M3 C D
MEMB M4 D A
MEMB M5 A C
```

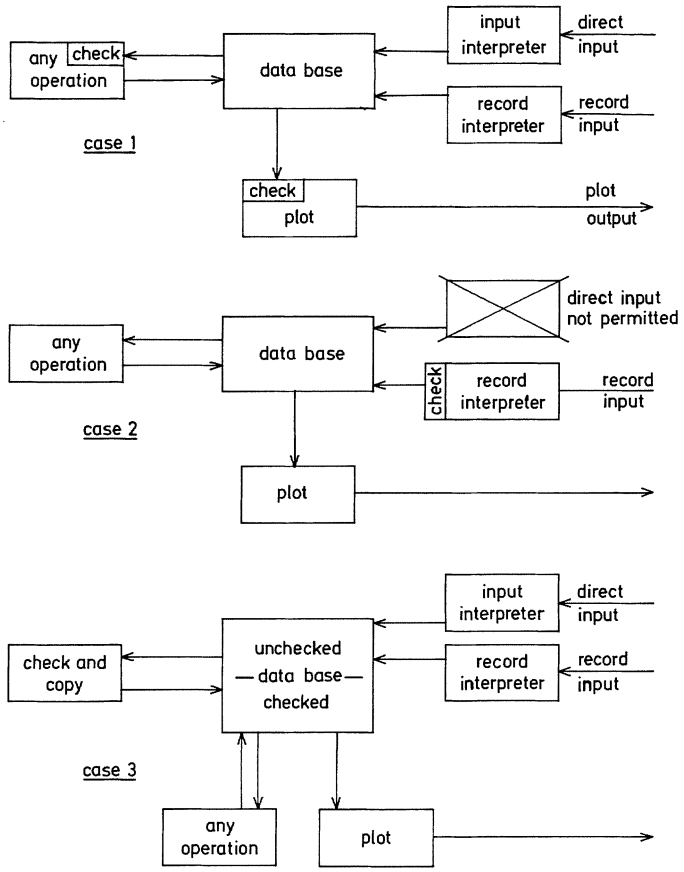


Fig. 4.11. Three different CAD system architectures with respect to data validation

In addition we may use the interactive input of the data base management system. If we intend to use interactive graphical input, we have to define how the communication should be performed in terms of actions with mouse, keyboard and function keys. For output we will be likely to use a graphical mapping. The schema is mapped onto labelled points placed in a geometrically correct way (with some viewing transformation) on a sheet of paper or a display screen. The members are mapped onto straight lines between the starting and end nodes. It is worthwhile to remember that the transformation functions which convert the object representation from one of these subschemas to another must know both subschemas (see Sect. 3.3.3).

Note that the mapping from the DBTG schema to the declarations in the PASCAL programs is not a complete one-to-one mapping. The PASCAL declarations of the records NODE and MEMBER guarantee that each member can refer only to nodes (not to other members) and, in particular, to exactly two nodes. These properties of the abstract model are not adequately reflected in the DBTG schema. Here, a limitation of DBTG-oriented data base management systems with respect to their application in CAD becomes evident. Thus we are forced to include in our system a checking

(validation) routine which determines whether an instance of the DBTG schema is consistent with this restriction or not.

Due to this deficiency, we now have a choice between various architectures, as shown in Fig. 4.11. They differ in how the validation process cooperates with the other processes (input, output, data storage, and other operations). In Case 1, direct input to the data base is permitted. Thus the above mentioned restrictions may be violated and must be checked before a program uses the data. In Case 2, we allow input to the data base only via an interpreter, which uses the PASCAL record declarations to represent the objects. In this case we can guarantee that the DBTG data base content is consistent with the restrictions. Both of these architectures have their pros and cons. In Case 1, the user may be confronted with an error message at a late stage in the problem solution process and may lose time in having to go back to input (not to mention the fact that the repeated checking consumes unnecessary computer resources when applied again and again to unmodified data). In Case 2, we lose the option of direct input to the data base, and we may be unnecessarily constrained to provide the input in a form that is consistent not only at the end of all input but also at all intermediate steps.

A third way of structuring the system is shown as Case 3 of Fig. 4.11. Here the consistency check is used as a significant intermediate step between input and all subsequent operations, which thus guarantees that all operations act on valid data only. This advantage is achieved at the expense of imposing more restrictions on the user of the system. He can no longer switch back and forth between input and analysis; but he must realize that there is a fundamental difference between the raw input data and the validated data, and that data validation is an important intermediate step.

4.2.2 Binding

According to [SALT78] the term binding stands for the replacement of a name by the object it actually represents. Thus binding occurs when the variable name PI in a program is replaced by its value 3.14159, binding also occurs at a lower level when the decimal value 3.14159 is substituted by its bit representation in computer storage. Other examples of binding are: the substitution for a subprogram name (in CALL SUB(...), for instance) of the subprogram itself; the association of a file that is addressable in a piece of program with an actual data set on a specific peripheral storage medium; or the assignment of the address of a data structure to the corresponding pointer variable.

In the design of a CAD system it is essential to pay attention to the time at which such binding occurs (the "binding time"). The following times, at least, should be distinguished:

Case 1: Binding at programming time (the time when the program is written in a programming language);

Case 2: binding at module binding time (the time when the program and all its subroutines are bound together to form an executable module);

Case 3: Binding at job preparation time (the time when a job is prepared for execution); and

Case 4: Binding at job execution time (the time when a process is executed).

These different times are often not identified as being separate. For instance, if we write a simple FORTRAN program which reads some data, performs certain calculations, and prints results, only the programming time and the run time are obvious. But let us take a more common case for illustrative purposes. Consider a program which needs an "equation of state" for a material (e.g., the pressure as a function of density and temperature for water) for performing calculations. The following approaches correspond to different binding times of the equation of state to the program.

Case 1: Binding at programming time.
The equation of state is explicitly coded into the program as an internal procedure or simply as inline code. This corresponds to binding at programming time. The equation of state (and hence, the material to be treated by the program) cannot be modified at a later time. The possibility of including a case option to select among a number of predefined equations of state does not alter the basic situation, as it merely replaces one materials by a limited and predetermined number of choices.

Case 2: Binding at module binding time.
The equation of state is declared as an external procedure in the program. Different procedures corresponding to different materials are loaded in different subprogram libraries. During the binding of all subprograms to an executable module ("linkage editing"), only one of these libraries is used. Thus, the function which computes the equation of state for the desired material may be selected shortly before the program is submitted to execution. Here we have no limitation at all with respect to the material. The only condition is that a unique pressure must be computable from any given density and temperature values.

Case 3: Binding at job preparation time.
For the moment let us assume that all the equations of state that might be used are of a parametric type: they involve the same mathematical expression, with only some parametric values to be adjusted for each material. In this case we can delay the binding of the actual equation of state until after the module binding time. For instance, we could let the program read the parameters from a file which is not associated with an actual data set until immediately before running the job. We have used the parametric version of an equation of state here since most programming languages allow for the delayed binding of data, but not of routines. In some systems, however, it is possible to delay the binding of routines as well [INTS75].

Case 4: Binding at job execution time.
Let us now assume that the material to be used or the equation of state is not known until part of the program has been executed (possibly because the program selects a mixture of other materials depending upon certain design criteria, and the equation of state is not determined until the mixture is defined). Again, as in Case 3 above, most programming languages would allow a delay of binding if a parametric representation of the equation of state is used. In this case, the parameter values would have to be computed in the program, or obtained from a data base and assigned to the variables which map

the parameters. Certain systems, like REGENT [LEIN78], would allow us to treat routines in just the same way as data, in that they would allow us to generate, compile, bind, and execute a complete subprogram in a single program step.

The following example shows the binding of an equation of state which computes pressure p as a function of density rho and temperature T for various materials (using PL/1 as a programming language):

Case 1: Binding at programming time combined with a selection between predefined options at execution time.

```
DECLARE P ENTRY(DECIMAL,DECIMAL) RETURNS(DECIMAL) VARIABLE;
P1: PROCEDURE(rho,T) RETURNS(DECIMAL);
      /*code for material 1 */
      RETURN( p);
      END P1;
P2: PROCEDURE(rho,T) RETURNS(DECIMAL);
      /*code for material 2 */
      RETURN( p);
      END P2;
P3: PROCEDURE(rho,T) RETURNS(DECIMAL);
      /*code for material 3 */
      RETURN( p);
      END P3;
      . . . . . .
      SELECT (material) ;
            WHEN(mat1) P = P1;
            WHEN(mat2) P = P2;
            WHEN(mat3) P = P3;
            END;
      any statement using P(rho,T);
```

Case 2: Binding at module binding time.

```
DECLARE P ENTRY(DECIMAL, DECIMAL) RETURNS(DECIMAL) EXTERNAL;
      any statement using P(rho,T);
      /*the operating system language is used to assure that the desired version of
      the function P (corresponding to the desired material) is bound into the load
      module */
```

Case 4: Binding at job execution time.

```
      /* the following programming language is an extension of PL/1 provided by
      the CAD system REGENT [SCHL78];
      the attribute DYNAMIC indicates that binding is to be performed at execution
      time */
      DECLARE MATERIAL_NAME CHAR(4);
      MATERIAL_NAME= any_expression_defining_the_material_name;
      BEGIN;
            DECLARE P ENTRY(DECIMAL, DECIMAL) RETURNS(DECIMAL)
            DYNAMIC MODULE(MATERIAL_NAME);
            any statement using P(rho,T);
      END;
```

The next (admittedly somewhat artificial) example shows how the value of 3.14159 may be bound to a variable PI at different times, again using PL/1 as the programming language:

Case 1: Binding at programming time.

```
DCL PI DECIMAL STATIC INTERNAL INITIAL(3.14159);
```

Case 3: Binding at job preparation time.

```
DCL PI INTERNAL;
DCL INPUT FILE STREAM;
GET FILE(INPUT) LIST(PI);
/*operating system language is used to allocate file INPUT to an existing data set
containing the value of PI */
```

Case 4: Binding at job execution time.

```
DCL PI INTERNAL;
DCL TERMINAL FILE STREAM;
GET FILE(TERMINAL) LIST (PI);
/*the value of PI is input by the user at the terminal */
```

It is obvious that *Case 1* is the preferred solution, because we know at programming time that PI should always have this value. Delaying the binding introduces a flexibility which in this case can only produce errors. In other cases we might be willing to sacrifice the safety and efficiency of early binding for increased flexibility. The designer of a CAD system will always have to choose between the criteria of:

- flexibility (enhanced by late binding), and
- safety (enhanced by early binding), as well as
- efficiency (enhanced by early binding).

It is important to realize that this decision has a very great influence upon the architecture of a CAD system. Two CAD systems may behave quite differently even if they perform the same fundamental tasks on the surface, depending on the choices made with respect to binding time. Also graphical packages like the graphical kernel system GKS are influenced very much by the decisions taken with respect to binding. An essential characteristic of GKS in this respect is the way in which graphic representation attributes are bound to primitives and segments [BONO82].

4.2.3 The Block Structure Dilemma

As shown in Fig. 4.1, CAD implies the execution of a sequence of programs with appropriate interfaces. Each program will produce more data which are to be added to the data base. Much of this data will be used by some other program later in the program chain without the need for consultation or modification by a person. It is generally accepted as good practice that such sequences of actions should be contained in an enclosing block which receives the first input and returns the final result (block structure approach in system design). We will see, however, that this approach is not always feasible or useful.

First let us note that for transferring data from one program (or process or block) P1 to another program P2, the interface must be declared in the containing block. We will call this interface P1__TO__P2 with the schema P1__TO__P2__SCHEMA. Similarly, all other interfaces have to be declared in the containing block. The following is a PASCAL-type formalization:

```
procedure CAD__PROCESS(INPUT, RESULT);
   type INPUT__SCHEMA          = record.....end;
   type P1__TO__P2__SCHEMA = record.....end;
   type P2__TO__P3__SCHEMA = record.....end;
   .....
   type PN__TO__PLAST__SCHEMA = record.....end;
   type RESULT__SCHEMA             = record.....end;
      var P1__TO__P2: P1__TO__P2__SCHEMA;
      var P2__TO__P3: P2__TO__P3__SCHEMA;
      .....
      var PN__TO__PLAST : PN__TO__PLAST__SCHEMA;
      var INPUT              : INPUT__SCHEMA;
      var RESULT             : RESULT__SCHEMA;
         call P1(INPUT        ,P1__TO__P2);
         call P2(P1__TO__P2,P2__TO__P3);
         .....
         call PLAST(PN__TO__PLAST,RESULT);
end CAD__PROCESS;
```

A solution of this type, however, is not satisfactory for several reasons:

- when we realize that a CAD process may take many months or even years for completion, it is evident that we cannot wait until the schema of the data to be transferred from PN to PLAST is defined before we start executing P1 and P2. At the beginning of the activities, we do not even know how PLAST will look;
- there is no good reason why CAD__PROCESS should know the schema of any of the transferred data unless it has to request or modify part of that data. On the contrary, details of the transport data schemas should be hidden from the enclosing process in order to avoid inadvertent modification;
- Associated with the declaration of the data in the above example is the allocation of resources for the representation of these data (storage space in the case of programming languages). Thus, in the above example, resources would be allocated to all interface data throughout the whole time, even though such a need exists only from some point within the producing program to some point within the consuming program. This is a waste of resources.

The problems associated with block structures have been investigated in [TOWS79], [WULF73], [PARN72] and others. From a system designer's point of view, one would like to define an "envelope" for data. Such envelopes should be able to accommodate data of different schemas (letters enclosed in envelopes are transported by mail regardless of their contents, provided that the exterior of each envelope follows certain standards). The possibilities of allocating such envelopes and disposing of them at any suitable time would be desirable. Envelopes could be used to transfer data from one block to another. With two envelopes, the above example might read as follows:

```
procedure CAD__PROCESS(INPUT, RESULT);
var INPUT__SCHEMA          = record.....end;
var RESULT__SCHEMA         = record.....end;
var P__ODD__TO__P__EVEN: envelope;
var P__EVEN__TO__P__ODD: envelope;
var INPUT                  : INPUT__SCHEMA;
var RESULT                 : RESULT__SCHEMA;
call P1(INPUT                ,P__ODD__TO__P__EVEN);
call P2(P__ODD__TO__P__EVEN,P__EVEN__TO__P__ODD);
.....
call PLAST(P__ODD__TO__P__EVEN,RESULT);
/*or perhaps
call PLAST(P__EVEN__TO__P__ODD,RESULT);
depending upon whether the number of processes is even or odd */
end CAD__PROCESS;
```

Let us take a look at various programming languages from this point of view. In fact, the envelope technique is used extensively in systems which are based on FORTRAN or PL/1. In FORTRAN, envelopes may be mapped onto arrays of REAL and/or INTEGER data of a COMMON block or a parameter with the EQUIVALENCE technique. In PL/1, based variables offer a great variety of options to implement envelopes. In both FORTRAN and PL/1, files may serve as implementations of an envelope. In the languages of the ALGOL and PASCAL families which are more restrictive with respect to type checking, it is much more difficult (if at all feasible) to violate the block structure principles and to implement envelopes for data of unknown schemas. We do not want to underestimate the dangers associated with the misuse of EQUIVALENCE in FORTRAN or based variables in PL/1 (or files in both); but the judicious use of these features offers possibilities which are urgently needed in practical CAD system implementation, and which do not find adequate support in more restrictive programming languages. The package concept in the programming language ADA [ADA__79], [ADA__82] provides a step towards a solution of this problem. It allows us to hide the internal interface data structures safely within the package, so that they cannot be accessed by the containing block. However, it does not resolve the problems of wasted storage space, or the problem of the as yet unknown schema for interfaces between future activities.

Let us now consider data base management systems under the same aspects. Obviously data base management systems are well suited to transport data from one program to another, since this is one of their fundamental tasks. To some extent, they offer sufficient flexibility to allow us to augment and modify the schema at later times without the loss of already existing data. Thus one might be tempted to leave all data transfer tasks to a data base management system. However, there are drawbacks to this approach. If solely used for interfacing programs, data base management systems require a considerable amount of overhead due to the necessary transformations from the external schema (used in the programs) to the internal schema (used for data storage) and in the return. It is unlikely that a user would want the stiffness matrix of a finite element program to be stored in a data base, where it is stored less efficiently than on a simple file. The advantage of being able to query or possibly modify individual data elements does not apply to stiffness matrices, which are more or less

meaningless except to the analysis program for which each matrix has been assembled.

After the general trend towards restrictive block-structure oriented programming techniques and languages in the 1970s, from the CAD standpoint it is desirable to see the development of efficient and practicable techniques for passing data between programs (modules, processes) which are safer and theoretically better founded than the constructs that are now available in FORTRAN and PL/1. A typical situation which calls for the violation of block structure is the interactive definition of graphical information at a terminal, where the information is to be moved upwards in the process hierarchy in order to be stored somewhere for future use in other programs. During the process of defining this information in an interactive way, we must be able to deal with information chunks [TOWS79], which are created and deleted in a random way and not in a block structure.

4.2.4 Algorithmic Modeling

So far we have dealt with models which could be mapped relatively easily onto a data schema. The representation of an object was always considered to exist, though it might be undefined or defined at a given time. Such a data representation of objects consumes resources; and in many cases these resources are too valuable to be spent. As an example: when graphic information is displayed at a remote terminal, the transport of the information across the connecting line consumes both transmission cost and manpower (the time of the operator who has to wait until the transport is complete). In order to save part of the resource, it would be preferable to condense the information. For graphics text, for instance, a data representation of the letter strokes consumes more storage space and transport time than a character string along with the information on how the character string should be expanded into line strokes. In general, a data model may be replaced by:

- the identification of an algorithm; and
- a set of (condensed) data which will be used by the algorithm to generate the complete data model.

We call such a condensation an "algorithmic model". The prerequisites, of course, are that the algorithm is properly implemented and that the mapping from the condensed data to the expanded form has been agreed upon by all users of this "shorthand" model. Similar algorithmic models are implemented in conventional programming languages. Arithmetic functions like SIN or COS are not supplied as tables of data, but rather as algorithms. A geometric modeling system for two- and three-dimensional objects, which is completely based on algorithmic modeling, is described in [SCHU76]. In an algorithmic model, data are not stored, but they are evaluated whenever they are needed. Two questions arise:

- When should a model be represented as data?
- When should algorithmic modeling be used?

The answer cannot be given on the basis of abstract operations which the system is to perform. A particular operation may be implemented with either approach. The

Table 4.3. The choice between data models and algorithmic models

Aspect	The preferred modeling is:	
	data model	algorithmic model
storage capacity	high	low
processing cost	high	low
usage rate of data	high	low
processing rate	low	high
time to retrieve data	low	high
time to transport data	low	high
change rate of data	low	high

answer must be derived from resource considerations, as indicated in Table 4.3. In the extreme cases the preferred solution is obvious:

- A data model is preferred when storage capacity, processing costs, and usage rates are high.
- An algorithmic model is preferred when the computer has a fast processor, when the retrieval costs or transport times for data are high, and when the model is rapidly changing.

In the wide range of practical situations, the choice is a matter of judgment based on experience more often than on objective criteria. A particular problem is posed by the fact that the "best" solution is a function of how the system is used. In the early design phase of a three-dimensional body, the rate of change of the model is usually so high that perspective views (and dependent data such as weight) are used only once before the next change is made. Thus it is preferable to use an algorithmic model for the projection of the body (and other data). When a display is requested, the projection lines may be evaluated and displayed on a plotter or a screen, line by line, with a minimum of storage requirements. At a later stage in the design process, modifications of the body become rare, but editing of the two-dimensional pictures of the body may be required. Now it is advantageous to store the projection of the body as data, in order to avoid unnecessary repetition of the projection operation.

The opportunity to choose between algorithmic modeling and data modeling of the same object, depending upon the type of work to be done, may have a significant influence on the usefulness of a CAD system in the different phases of the design process. This aspect will require much more attention in the future.

A similar line of argument may be used to compare the two essentially different types of solid modeling systems: constructive solid geometry and boundary representations. Constructive solid geometry is in fact a representation of a function in the form of data structures, while boundary representation is the instantiation of the result of such a function. In early stages of geometric design of solids, when drastic changes are still likely to occur, the functional description of shape is more advantageous, while in the later design stage detailing is done more easily by storing only the result of the operations.

4.3 The Resource Aspect

4.3.1 Software Machine Design

When we investigate software machines which are potential candidates for incorporation into larger systems, we generally note that they obey special rules for using certain resources. Examples of such standardized use of resources are: a filing system (rules for naming and structuring external data storage, most often used in program chains, see Fig. 4.1), the COMMON-block technique (probably the most widely used CAD system basis of the FORTRAN oriented world, with rules for naming and structuring sharable internal memory), and subroutine packages (based on rules for naming the procedures and structuring their argument lists).

Such systems work fine as long as they are just used by themselves. However, when put together into a bigger system, conflicts usually arise either because some of the "subsystems" do not allow the sharing of certain resources with others, or because they use shared resources according to conflicting rules. Let us illustrate this point by a few examples:

- obviously one cannot use two independent subroutine packages in one program if both together demand more memory capacity than is available;
- even if each of the packages has its own dynamic memory allocation facility (as some FORTRAN packages provide by means of Assembler extensions) they may not be usable in combination anyway. Each of them must be limited to a maximum amount of memory: the authorization to use a resource must explicitly or implicitly be passed from some higher level or organization; and
- the same is true if the resource to be used is not of a quantitative nature (such as memory space), but of a qualitative nature. Software machines cannot be used in parallel if they make independent use of certain global names such as program names, file names, common block names. This conflict becomes evident if one thinks of using two graphics packages in a single program, one for data presentation and one for geometrical design. The chances that both of them use different subroutines with identical names – like OPEN, CLOSE, or PLOT – are rather high.

It is not sufficient that the required resources are available: some of these resources, namely those which represent the state of a process, must be reserved for exclusive use by this process. Other processes must be inhibited from modifying such resources. The design of software machines for general use in a large number of systems will have to deal with the following questions:

- What is the function of the software machine?
- Which resources does it require?
- How are these resources to be supplied? How must they be initialized (set up in a proper state prior to their actual use)?
- How do we guarantee that certain parts of these resources will remain unchanged as long as they are needed?

4.3.2 Designing Against Resource Conflicts

4.3.2.1 The Abstract Machine

Let us consider the execution of a CAD application as a running process. During this process certain other processes are created, executed, and terminated (such as the looking up of design rules from a library, or the display of graphical information on a display screen, see Sect. 3.2.4.1). Some of these processes last only as long as a "call" to a subroutine, as is often the case for mathematical algorithms. Other processes live longer; for example, the looking up of design rules in a data base is usually implemented in the following way: Make the design rules available ("open"), look up as often as needed ("search"), terminate the availability ("close"). In such a case many short-lived look-up processes exist parallel to a long-living process which we may call "maintenance of the environment for the look-up table process" (Fig. 3.18). Similarly, the communication process between the application program and a graphics terminal lasts much longer than the display process for some picture.

Let us take a look at implementations of such processes in today's CAD systems (which in most cases means FORTRAN). Short-lived processes are usually found as calls to subroutines which require all information to be passed as arguments (or possibly in a COMMON block which is filled prior to the call). Long-living processes may be found in the form of a sequence of calls to a subroutine package. All the values of the variables in the COMMON block for such a package represent the actual state of the corresponding process. It is essential for correct operation that these variables will be used exclusively by this particular package. Note that there is a functional difference between the uses of COMMON in these two examples: in the first case it is merely a shorthand writing for arguments; in the second case it is the realization of the state of the process. Confusing these two functions inevitably leads to problems (and one of the drawbacks of FORTRAN is that it supports this confusion).

However, the situation is even more complicated. Quite often several processes of the same type exist parallel to each other. A typical example is the use of several graphics displays for different purposes in the same program: some are used for close-up views, diagrams, and communication, while others are used for design drawings and overall representations. It is obvious that whenever a software machine is to continue such a process, it must be made clear which one of the particular processes should now be continued. While in the first case (only one process executing) there is no apparent need to distinguish between the process and the machine, this need becomes evident whenever a machine can operate parallel processes. In the first case the resources which represent the state of the process may or may not be included in the software machine (the COMMON technique includes them in the machine; see Fig. 4.12) while in the second case the software machine must be able to manage a varying number of process state representations (Fig. 4.13). The management of resources and the management of the different processes become clearly separated tasks (see p. 49 in [SCHN78]).

The various functions of an abstract machine either change the state of the process or provide to the "parent process" information as a function of certain input parameters and the present state [PARN75]. In a symbolic way one might write:

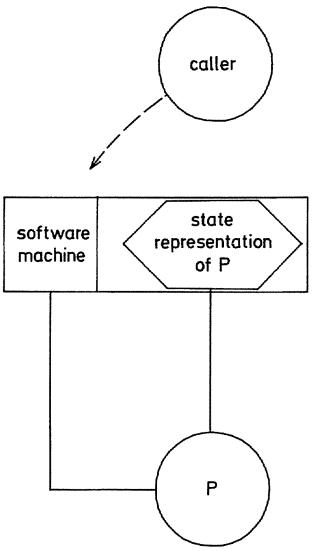


Fig. 4.12. A software machine for a single process

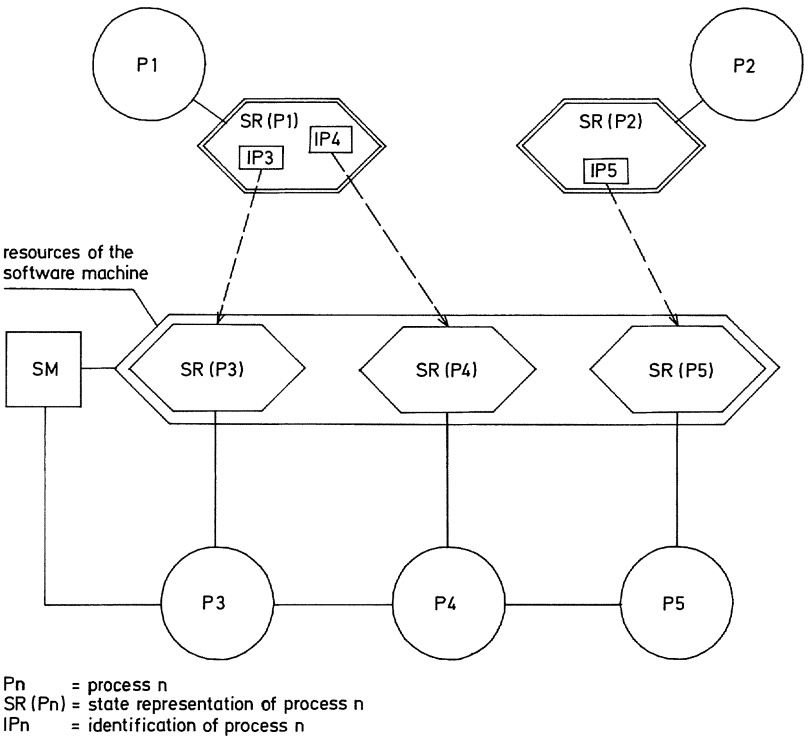


Fig. 4.13. A software machine for many parallel processes

abstract machine function = (O, V, state)

where O is a set of operating functions o,
which change the state

state = o(state, input),

while V is a set of value-delivering functions v,
such that

output = v(state, input).

As noted above, whenever the software tool is to be used for parallel processes P_i, a reference to the state of one particular process P_k must be passed as part of the input. Hence

input = (identification of process P_k, other input).

A user can use a software tool intelligently only if he has a clear understanding of its functional capabilities. For this reason we formulate the first rule

(R1): The functional documentation of a software machine must give a precise definition of the type of process which is driven by this machine. This means:
- a complete description of the state in terms of objects of the abstract object type, corresponding to the underlying conceptual schema;
- a complete description of all operating functions in terms of their effect on the state;
- a complete description of the value-delivering functions.

This rule should require no special explanation. However, it is violated quite often with respect to completeness.

The second rule applies to software machines which are to be used to drive similar processes in parallel:

(R2): Whenever a software machine may conceivably drive more than one process in parallel, caller and software machine should agree upon a unique identification of each newly created process in order to be able to communicate about this process at a later time.

There are many ways an agreement about the identification of the new process (its name) can be achieved. The caller may pass a name to the software machine, or the software machine may determine the name. In the first case, conflicts may arise if more than one caller wants to use the same name; this conflict may be removed if the name is prefixed by the caller's name. In the second case, it may happen that a process is created and later terminated and another process is then given the same name by the software machine. If the caller erroneously refers to the name of the already terminated process, a misunderstanding would result without the possibility of detection. The caller would think that operations are being performed on the old process, while the callee is performing the actions on the new process. This problem may be removed if the creation time (date and time) is made part of the process name. In either case, at least one of the partners (the caller or the software machine) must maintain a table which associates the name which was fixed in the other's environment with a private name within its own environment (a name is valid only within a given environment, see Sect. 3.3.1.3). A very efficient means of communication be-

tween caller and software machine may be achieved when both agree to use a combination of their respective private names. Such a combination of names represents a uniquely labelled key (in the literal sense): the label identifies the key to the caller (the person) while the shape of the key proper uniquely belongs to the callee (the door lock). An illustration of this technique is given in Sample Listing 4.3 and Sect. 4.3.3.2.

4.3.2.2 Process State Representation

In Sect. 3.2.4.2, we characterized a process by the fact that its state is modified only by the process itself. The process state is represented by the situation of certain resources (values of variables, position of a magnetic tape on its unit, existing connection to a certain terminal, etc.). If we have more than one parallel process, then the problem may arise that one process modifies the state representation of other processes. In order to avoid this, we formulate the rule:

(R3): A software machine should be designed and implemented so that the state representations of processes which it creates cannot be modified by other processes.

It is not always easy to implement this rule in a strict sense. If a software machine is used for one process only, the state representation of this process may be integrated with the software machine itself as shown in Fig. 4.12. In FORTRAN a common technique is to include a COMMON block which contains declarations representing the process state in all subroutines and functions which constitute the machine. This method efficiently protects the process state against modification by other processes *provided* that no other program uses the same name for a COMMON block. Safer and much more powerful techniques are provided by the package concept in the programming language ADA.

If, however, a number of independent processes are to be driven by the software machine, the machine should have the capability to maintain a variable table of process state representations as illustrated in Fig. 4.13. Here the two processes P1 and P2 use the same software machine SM. P1 has created the processes P3 and P4, P2 has created P5. The software machine (following rule (R2)) has agreed with the "callers" about the identification of the processes. The identifications of the subprocesses are stored within the state representation of the higher-level processes (IP3 for process P3, for instance). The state representations of P3 through P5 are accessible only to the software machine SM, and are thus effectively protected (an example is given in Sect. 4.3.3). When a caller calls the software machine, it supplies the identification of the process to be continued. The corresponding state representation is then bound to the software machine (the name is replaced by what it means), and the process continues.

Another approach is shown in Fig. 4.14. This solution is feasible even in FORTRAN, where the above method is difficult to implement because of the inability of FORTRAN to perform dynamic storage management for allocating and deleting the data structures which represent the process states. Here the state representations of all subprocesses (created by the software machine SM under control of a "caller" process) are embedded in the state representation of the caller itself. In a FORTRAN implementation, one would implement this as an array of REAL or INTEGER data

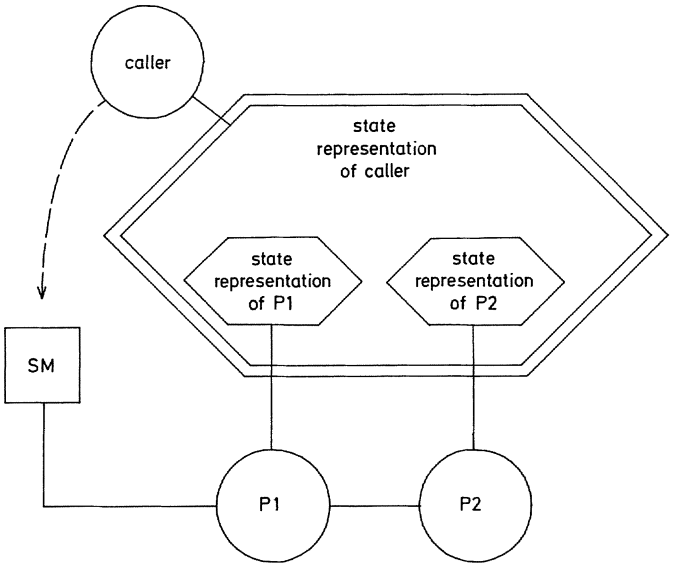


Fig. 4.14. Allocation of the resources for state representation by the caller

which are not used at all by the calling program, but passed to every routine of the software machine upon the request of a software machine function. It is a matter of discipline *not* to use this array of data for any other purpose. Some programming languages (PL/1 for instance) provide limited capabilities for the higher-level process to treat the data structure representations simply as resources (or as "envelopes" of the data structures which they contain) without being able to access the contained data structures (see Sect. 4.3.3.2).

4.3.2.3 The Concrete Machine

Software machines do not only use resources for representing the state of a process. They also use resources for operational purposes. Examples of such resources are: primary memory, working files, names of files in communication with the operating system, names of subroutines. Some of these resources are merely a certain quantity out of a larger pool (memory is an example). Other resources, however, are well identified and must be reserved for exclusive use (such as the names of the subroutines). In either case, problems may arise whenever more than one software machine is used in a CAD system.

With respect to quantitative resources, conflicts may arise if one process monopolizes a resource. (Certain systems such as ICES tend to make optimal use of primary memory by using as much as possible. This will cause failure if combined with another package which itself provides a dynamic storage management facility.) Hence, we state the rule:

(R4): If a software tool is able to obtain certain quantities of a resource for its operational purposes, the parent process should authorize it to allocate up to a certain amount of that resource. Otherwise, the necessary resources should be supplied by the parent process. In any case, the documentation must include a list of the resources from which the software tool needs a certain quantity.

With respect to qualitative resources (those which can be identified as individuals, such as all names), we state the following rule:

(R5): Qualitative resources should be obtained from the parent process. If this is not possible (as for the names of subroutines), the description of the software machine must identify which qualitative resources are used and how the software machine might be modified to permit the replacement of qualitative resources with others, if necessary.

It is worth noting that the potential of name conflicts has been recognized in the graphical kernel system GKS. This standard recommends that GKS implementations should provide a "name converter", which allows for the replacement of any global name in the package during the process of installation in a computer environment where name conflicts would otherwise arise.

4.3.2.4 Resource Management Strategies

Sometimes the number of quantitative resources needed by a process depends heavily on certain process parameters. The buffer area for a graphic display file is a familiar example. The particular requirements may also grow and shrink considerably with time. The design of software tools depends heavily on the resource management strategy followed:

Case A: During the whole process a maximum amount of resource is allocated to the process, no matter whether it is really needed (this is typical for local FORTRAN working arrays in subroutines). If many processes follow this strategy the resource may soon be exhausted.

Case B: The process obtains resources when needed and returns them when they are no longer needed. Although this strategy makes "optimum" use of the resource, it may cause considerable overhead and may also lead to the so-called "fragmentation problem" unless the sequence of allocate and free requests is issued on a last-in-first-out basis [KNUT69]. As noted in Sect. 4.2.3, block structuring of the individual processes with respect to their resource allocations is not always a satisfactory solution.

Case C: The process provides a resource estimate algorithm. The parent process usually has sufficient information available to produce good estimates of the relevant process parameters for a certain period of time. With these parameters, an estimate of the amount of resource needed may be generated so that the parent process can supply this resource to the process. Problems similar to those in Case B above may arise, but their probability is significantly reduced.

Each of these strategies has its advantages and disadvantages and we do not pretend to recommend one as being superior in all cases. It is not surprising that software machines which perform the same function may appear totally different in their implementation, depending on the resource management strategy. Hence, we formulate the rules:

(R6): The documentation of the software machine must contain a complete list of the resources which are needed for successful operation.

(R7): The documentation of a software machine must include a description of the resource management strategy of this machine in particular for
- *Case A:* the limitations on the relevant process parameters and the amount of resources needed at all times;
- *Case B:* an estimate of the amount of each resource needed as a function of relevant process parameters;
- *Case C:* same as for Case B. In addition, information should be given about the consequence of providing more or less than the estimated amount of each resource.

It is suggested that the resource estimate algorithms should not only be documented in the user's manual, but should also be provided as callable subroutines within the software machine itself.

In some cases a certain amount of one resource may be replaced by another. As an example: external storage may be used instead of primary storage. In such a case, the software machine itself is unable to determine the optimal global balance between these resources, because it does not know how an attempt to improve its own performance would influence other processes. Hence, if the software machine has the capability to adapt itself to different resource configurations, strategy C is the only one which allows for a global optimum.

In any case, the software machine should provide information about the amount of resources actually used for a particular process, so that the user is able to learn from previous applications and use the machine more efficiently in the future.

4.3.2.5 The Components of a Software Machine

Let us combine the functional aspect of a software machine with its resource aspect in the following schema, which is illustrated by Fig. 4.15:

software machine = (abstract machine function,
resource management machine,
documentation);

The abstract machine function has been described in Sect. 4.3.2.1. The resource management machine must be explained in more detail:

resource management machine = (management state,
management functions);

The resource management state includes a list of the authorizations (or limitations) obtained from a higher-level resource management process, and a list of the

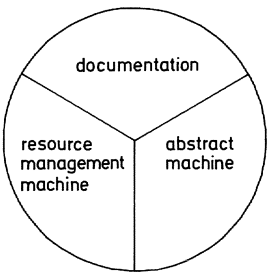


Fig. 4.15. The components of a software machine

resources which have actually been allocated by the various processes to the "abstract machine" part of this software machine. The resource management functions are operational functions which authorize or limit the use of resources by the software machine, or which deliver information about resource requirements (estimated or actual).

Note that while the abstract machine part may drive several processes in parallel, there is only one resource management process associated with a software machine. The machine is completed by its documentation:

```
documentation = (documentation of the abstract machine,
                 documentation of qualitative resource requirements,
                 documentation of resource management strategy);
```

In the previous pages (Sects. 4.3.1 and 4.3.2), we have emphasized that the design of software machines should try to avoid potential conflicts in the assembly of such software machines into larger units. This means that we have emphasized the bottom-up approach to system development, as opposed to the top-down or "stepwise refinement" approach which is often advertised, particularly in computer science oriented literature. However, top-down design may imply that the system is broken down into components at an early stage, when the knowledge about the consequences of such a decision is still too sparse [JACK82]. Furthermore, the practical problems in designing and implementing large systems for CAD and CAM are very often too great. A top-down approach might easily result in the final system being available at a time when it is no longer needed. Hence, one often has to be satisfied with partial solutions that are feasible in the required time scale. If time and the judicious design of the partial solution permit, they may later be combined into larger systems [LINC81].

4.3.3 A Sample Software Machine: The Stack Machine

4.3.3.1 The Task and a Simple Solution

Many operations in CAD require a set of objects as operand. As an example, we might mention the problem of hidden line removal from a drawing of a three-dimensional body, where all the surfaces of the body have to be considered simultaneously. This example further illustrates that the objects in such a set may have different representations (a cylinder will have a different representation from a plane). Only very few pro-

gramming languages (PASCAL and C for instance) provide a limited set facility. In general, sets are mapped onto list structures [KNUT69], which are available in a much greater number of programming languages. In order to minimize the amount of work involved for this example we will use the most simple list structure: a stack. The idea of the following paragraphs can easily be extended to more complicated structures. The basic operations of a stack are:

```
o-functions: push(item);
                 /* puts an item on top of the stack */
             pop(item);
                 /* removes the item from the top of
                    the stack and delivers it */
v-function:  empty;
                 /* true if stack is empty false otherwise */
```

The implementation of these functions in various programming languages is straightforward and may be found in a number of textbooks. Here we will use the programming language ADA [ADA__79], whose "package" concept can be interpreted as a formal way of describing software machines.

The ADA program in Sample Listing 4.1 deals with the abstract machine only. The resource aspect is completely hidden. Only the compiler has to care about the memory resources, not the programmer. The resource aspect becomes apparent, however, if we specify the task as follows:

- the software machine should be able to manage several stacks;
- the software machine should not make any assumptions whatsoever regarding the structure of the items. The ADA module of Sample Listing 4.1 is applicable for one ITEM schema only. Using a case construct, one could easily expand the applicability to a finite number of *predefined* schemas (see Sect. 4.2.2). But *all* schemas which should ever be stored in the stack would have to be bound to the package at *programming* time by including their declaration. Hence, ADA does not readily provide answers to our problem;
- the resource (storage space) allocated to each stack should be restricted. One reason for this restriction is to avoid storage overflow and uncontrolled program failure in a case of incorrect use (in an external loop of push operations, for instance). If the limit is too restrictive, the machine should be able to call for help; and
- the possibility to save whole stacks and to restore them at a later time should be provided.

Before we plan the "stack machine" in more detail, let us rewrite the above ADA module in PL/1 in order to be able to compare the purely functional program with the final software package more easily. We choose PL/1 rather than ADA for comparison purposes because in PL/1 it is much easier to implement features which are *not* readily available in the language. ADA (like PASCAL) is less permissive, and allows us only to express objects and operations which the designers of the language wanted to allow. For this reason, these languages are much more suitable for teaching purposes than PL/1 or FORTRAN. PL/1 may appear too powerful to be a "good" language, but often this extra power is quite helpful in systems programming (see Sample Listing 4.2).

Sample Listing 4.1. A stack machine for a single item type. The ADA version

```
package STACK__MANAGER is

    type ITEM is
         record
           .....record declaration....
         end record;

    NULL__ITEM : constant ITEM        := ....value of an ITEM which will be
                                           recognized as NULL__ITEM .....;
    procedure PUSH ( NEW__ITEM :  in ITEM);
    procedure POP   ( TOP__ITEM  : out ITEM);
    STACK__FULL : exception; -- may be raised by push
    end;

package body STACK__MANAGER is
    SIZE : constant INTEGER := 2000;
    subtype INDEX is INTEGER range 0..SIZE;
    type INTERNAL__ITEM is
        record
           CONTENT : ITEM;
           SUCC    : INDEX;
           PRED    : INDEX;
        end record;

    STACK : array (INDEX'FIRST..INDEX'LAST) of INTERNAL__ITEM;
    FIRST__BUSY__ITEM : INDEX := 0;
    FIRST__FREE__ITEM : INDEX := 1;

    function BUSY__LIST__EMPTY return BOOLEAN is .... end;
    function FREE__LIST__EMPTY return BOOLEAN is .... end;
    procedure EXCHANGE (FROM     : in INDEX; TO : in INDEX ) is ...end;
    procedure PUSH (NEW__ITEM : in ITEM) is
       begin
          if FREE__LIST__EMPTY then raise TABLE__FULL; end if;
            .....
            -- remaining code for PUSH
            .....
       end PUSH;
    procedure POP (TOP__ITEM : in ITEM) is .... end POP;
       begin
            .....
            -- code for initialization of stack linkages
            .....
    end STACK__MANAGER;
```

Sample Listing 4.2. A stack machine for a single item type. The PL/1 version

```
PROCEDURE STACKM;
  DECLARE 1 ITEM BASED,
              2 ....
                .....STRUCTURE declaration....;
  DECLARE 1 NULL_ITEM STATIC,
              2 .... INITIAL( ... value of NULL_ITEM ...)
                .....repeat declaration of ITEM with initial values;
      /* PUSH: ENTRY ( NEW_ITEM: IN LIKE ITEM)
         POP  : ENTRY ( TOP_ITEM: OUT LIKE ITEM)
         STACK_FULL: CONDITION , MAY BE RAISED BY PUSH */
      /* PL/1 MACRO PROCESSOR CAPABILITY IS USED TO REPLACE "SIZE" BY
         "2000" AND "INDEX" BY "BINARY FIXED(15)" IN THE FOLLOWING PRO-
         GRAM TEXT                                                    */
      % DECLARE SIZE FIXED;
      % SIZE = 2000;
      % DECLARE INDEX CHARACTER;
      % INDEX = 'BINARY FIXED(15)';
      /* WHEREVER A VARIABLE OF TYPE INDEX IS MODIFIED CHECK THAT IT
         REMAINS WITHIN 0..SIZE                                       */
      DECLARE 1 INTERNAL_ITEM BASED,
                  2 CONTENT LIKE ITEM,
                  2 SUCC    INDEX,
                  2 PRED    INDEX;
      DECLARE 1 STACK STATIC
                  2 INTERNAL_ITEM ( 0 : 2000 ) LIKE ITEM,
                  2 FIRST_BUSY_ITEM INDEX INIT(0),
                  2 FIRST_FREEITEM INDEX INIT(1);

      BUSY_LIST_EMPTY: PROCEDURE RETURNS(BIT); ..... RETURN;END;
      FREE_LIST_EMPTY: PROCEDURE RETURNS(BIT); ..... RETURN;END;
      EXCHANGE: PROCEDURE (FROM , TO ) ;
            DECLARE (FROM,TO) INDEX;
                  ......
      RETURN;END EXCHANGE;
      PUSH: ENTRY       (NEW_ITEM );
            DECLARE NEW_ITEM LIKE ITEM;
                  IF FREE_LIST_EMPTY THEN SIGNAL TABLE_FULL;
                  .....
                  /* remaining code for PUSH */
      RETURN;END PUSH;
      POP  : ENTRY      (TOP_ITEM );
            DECLARE NEW_ITEM LIKE ITEM;
                  .....
      RETURN;END PUSH;
                  .....
                  /* code for initialization of stack linkages */
      RETURN;END STACKM;
```

4.3.3.2 Planning of the Stack Machine

The fundamental operations which we want to provide are:

management functions:
initiate__stack__machine(/*no resource restrictions*/,
file for messages, help)
terminate__stack__machine(final report, help)
estimate__resource__requirement(stack characteristics, help)
create__stack(resource allowance, stack name, help)
save__stack(stack name, resource for saving, help)
restore__stack(stack name, resources for saved stack, help)
o-functions:
push(stack name, item, help)
pop(stack name, item, help)
v-functions:
empty(stack name, help).

We will now deal with the problem of mapping the abstract object types mentioned in the above functions onto PL/1. We have to make decisions on the following issues:

What are the items which we want to push on the stacks? In our ADA example (and in practically all examples which may be found in the literature), the stack machine knows the schema of the items to be handled. At best, the machine knows a small number of schemas such that objects of various structure may be handled. But there is actually no reason why the stack machine should have this knowledge. In fact, this knowledge is bad knowledge. The stack machine is not allowed to perform any action with the objects in the stack items, so why should it know the names and structures of these objects? What is worse, if at a later time we decide to utilize the stack machine for stacking items with a different schema, we will be forced to include the new schema in the stack machine and recompile. The solution to this problem is to use a very general schema onto which most items may be mapped in an efficient way: the contiguous storage space. We allow the user to push and pop all items which may be mapped onto a contiguous storage space or "envelope", as introduced in Sect. 4.2.3. This generalization requires a "mapping around the language" (see Sect. 4.2.1.3) and will be difficult (if at all possible) to perform in restrictive programming languages of the ALGOL or PASCAL type. Other languages like PL/1 and even FORTRAN are much more permissive and make it relatively easy to perform such mappings. The stack machine should actually not deal with the objects pushed on the stack, but rather with the resources (storage space) that represent these objects. In PL/1, two possibilities offer themselves for the mapping of items to a contiguous area of storage space: CHARACTER strings and AREA data. The mapping around the language level for a byte-oriented computer according to Fig. 4.9 would be as follows:

abstract level	storage space
language level	CHARACTER(*) or AREA(*)
machine level	a set of contiguous bytes

The caller of the stack machine would have to perform this mapping, and would pass to the stack machine the length of the storage area (the PL/1 built-in function CURRENTSTORAGE may be used to determine this value in terms of bytes, corresponding to CHARACTERS on the language level) and the address of the first byte of the aggregate (which may be obtained by the PL/1 built-in function ADDR).

- How do we represent saved stacks?
  The problem here is similar to the problem of representing items with an as yet unknown structure. In this case, the representation of the stack is to be handed over to a higher level process (the caller) who should not do anything to the contents of the stack. Hence, the caller should not know the schema of the stack. In a way similar to the treatment of the stack items, we map the representation of a stack onto a contiguous storage space (a PL/1 STRUCTURE) which contains the stack representation and additional information (size, identification, producer and consumer). We call this an "envelope", by analogy to the envelope in which someone may send a letter (information in a certain schema) to someone else (or to himself) without the post office having to know the contents of the letter.
- How do we represent a stack?
  The stack is the representation of a stack process. We choose to use list structure. An array would be an alternative; but for storing items with varying storage space requirements, a list structure is more economical with respect to storage usage. The question arises whether the stack implementation should take into consideration right at the outset that stacks might have to be saved in the form of an envelope. This would mean that not only would the stack have to be enveloped for saving and restoring, but every stack would be implemented within an envelope, whether saving is requested or not. The decision cannot be derived from functional aspects. A consideration of the resources (here: effort involved in planning the data structure, processing and storage requirements) must give the answer. In order to reduce the planning effort, we decide to use only one representation of stacks for both saved and non-saved stacks. In a real case, the decision might come out differently.
- How do we represent the state of the stack management machine?
  The state of the resource management machine is characterized by the actual stacks, the storage allowance for each stack, and the actual storage use. In order to keep track of its state, the machine itself needs some storage space for a stack table. We do not decide at this point how the stack table should be implemented; we leave this decision until after further refinement of the schema and the algorithm. With respect to the operating states of the stack machine (see Sect. 3.2.4.3), the distinction of three different states appears to be necessary:

  1) "existing" prior to initialization of the stack machine, and after termination;
  2) "in repair" during a call to a help procedure; and
  3) "executable" otherwise.

- How do we represent names?
  The options among which we may choose in PL/1 are: integers, character strings with fixed or variable length, pointers, and offsets. The most efficient solution with respect to execution is the use of pointers. Pointers may be used both for naming and for accessing objects. Their use as names avoids the need for an ad-

Sample Listing 4.3. The PL/1 stack machine. The declarations of the state representation

```
/* the following declarations are to be included in all modules */
 DECLARE 1 STACKB EXTERNAL,
             2 MESSAGE_FILE FILE,
             2 OPERATING_STATE CHAR(32) VARYING INITIAL( EXISTING'),
             2 STACK_MACHINE_NAME CHAR(32) VARYING
                                              INITIAL('STACK_MACHINE'),
             2 STACK_TABLE INITIAL(EMPTY);
 DECLARE ERROR_CODE BINARY FIXED(15);
 DECLARE 1 NAME_OF_STACK,
             2 EXTERNAL_NAME CHAR(32) VARYING,
             2 INTERNAL_NAME POINTER /* TO STACK_ENTRY */
             2 GENERATION_TIME CHAR(16);
 DECLARE ACTUAL_STACK_ENTRY POINTER;
 DECLARE 1 STACK_ENTRY BASED(ACTUAL_STACK_ENTRY),
             2 INTERNAL NAME POINTER,
             2 LINKAGES_IN_STACK_TABLE,
             2 ACTUAL_STACK POINTER;
 DECLARE 1 ENVELOPE BASED(ACTUAL_STACK),
             2 LENGTH BINARY FIXED(31),
             2 EXTERNAL_NAME CHAR(32) VARYING INITIAL(EXT_NAME),
             2 PRODUCER CHAR(32) VARYING INITIAL(STACK_MACHINE_NAME),
             2 CONSUMER CHAR(32) VARYING INITIAL(STACK_MACHINE_NAME),
             2 BASE OFFSET(ENVELOPE.STACK) INITIAL(NULL()),
             2 STACK AREA(ALLOWANCE REFER(ENVELOPE.LENGTH));
 DECLARE 1 STACK_HEADER BASED(BASE),
             2 GENERATION_TIME CHAR(16) VARYING INITIAL(DATE()||TIME()),
             2 TOP OFFSET(STACK);
```

dress table. The disadvantage of using pointers as names is the reduced security of data: every process that knows the pointer to data can potentially access these data. With respect to the names of the stack items this is not a point of concern, since these items belong to the calling process anyway. With respect to the stack names, we use a combination of names corresponding to the two environments involved: the caller supplies a character string (limited to 32 characters), while the stack machine adds a pointer and the generation time of the stack (see Sampling Listing 4.3).

- How do we represent the help functions?
  The possibilities available in PL/1 are:

  1) to return a record (a PL/1 STRUCTURE) which is to be interpreted by the calling program. The information in this record (error code) will tell the caller whether the stack machine has detected an abnormal situation, and which of the predefined possible actions it has taken. The caller may then react appropriately;
  2) to call a help procedure which was passed to the stack machine as an argument, either in the present or in an earlier call. The help procedure itself is supplied by the caller.

Referring to our example, we decide to use the first option in most cases. The stack machine will then return an integer error code, which is set to 0 for a normal case and will be set to specific integer values if abnormal situations are detected. In such cases the state of the stack machine will not be changed. A full-stack exception, however, will cause a help procedure to be called. The stack machine will supply to this procedure the stack name, the present storage size of the stack, and the size of the item to be pushed on top of the stack; and it will expect from the help procedure the information of whether the storage allowance for the stack is to be raised and to what value.

4.3.3.3 Implementation of the Stack Machine

First we will summarize all declarations which are needed for representing the stack machine and the stacks. Part of the schema is still to be refined and mapped onto the possibilities of the PL/1 language. The declarations shown in Sample Listing 4.3 must be included in all modules of the stack machine. Communication between the modules is achieved by declaring the basis of the data structure (STACKB) as EXTERNAL; this attribute assures that all modules of the stack machine operate on the same data structure (provided that they are bound into a single load module before execution).

Two tests which should be performed in almost all entries to the stack machine are represented as macros in Sample Listing 4.4. These lines of codes have to be inserted whereever they are referenced.

We must choose a module structure for the different operations of the stack machine. In the sense of Parnas [PARN72] or in the programming language ADA, all the operations together would form a "module". The term module, however, is associated with different meanings. A module is sometimes used for the portion of code which is treated by the compiler as one unit. Sometimes the term module is used for the loadable and executable unit of machine code that is produced by the linkage editor (or binder) from various separately compiled programs and additional subprograms retrieved from a library. At this point, we are associating "module" with the program unit to be submitted to the compiler. Such modules would also usually represent the units which would be documented and listed separately.

Sample Listing 4.4. The PL/1 stack machine. The test macros.

```
/* the following macros are to be included where they are referenced */

/* test whether stack machine is initialized */
    IF OPERATING__STATE ¬= 'EXECUTABLE' THEN DO;
                        /* set ERROR__CODE */ RETURN;END;

/* test for existence of stack with NAME__OF__STACK in stack table */
   ACTUAL__STACK__ENTRY = NAME__OF__STACK.INTERNAL__NAME;
   IF NAME__OF__STACK.EXTERNAL__NAME ¬= ENVELOPE.EXTERNAL__NAME |
       NAME__OF__STACK.GENERATION__TIME ¬= STACK__HEADER.GENERATION__TIME
          THEN DO; /* set ERROR__CODE */ RETURN;END;
```

Sample Listing 4.5. The PL/1 stack machine. The module for initialization and termination of the machine itself.

```
INITST: PROC(FILE__FOR__MESSAGES,ERROR__CODE);
/* INITIATE STACK MACHINE */
MESSAGE__FILE = FILE__FOR__MESSAGES;
OPERATING__STATE = 'EXECUTABLE';
/* set ERROR__CODE */ RETURN;

TERMST: ENTRY(NUMBER__OF__ACTIVE__STACK__PROCESSES,ERROR__CODE);
/* TERMINATE STACK MACHINE */
/* test whether stack machine is initialized */
/* delete all remaining envelopes and stack */
/* write message on MESSAGE__FILE; */
OPERATING__STATE = 'EXISTING';
/* set ERROR__CODE */ RETURN;
END INITST;
```

Sample Listing 4.6. The PL/1 stack machine. The module for creating, deleting, and estimating stacks.

```
CREAST: PROC(ALLOWANCE,EXT__NAME,NAME__OF__STACK);
/* CREATE STACK */
/* test whether stack machine is initialized                              */
    DECLARE EXT__NAME CHAR(32) VARYING;
    ALLOCATE STACK__ENTRY;ALLOCATE ENVELOPE;
/*  insert stack entry into stack table;                                  */
    STACK__ENTRY.INTERNAL__NAME = ACTUAL__STACK__ENTRY;
    NAME__OF__STACK.EXTERNAL__NAME  = STACK.EXTERNAL__NAME;
    NAME__OF__STACK.INTERNAL__NAME  = STACK__ENTRY.INTERNAL__NAME;
    NAME__OF__STACK.GENERATION__TIME = STACK__HEADER.GENERATION__TIME;
/* set ERROR__CODE */ RETURN;

DELEST: ENTRY(NAME__OF__STACK,NORMAL__END);
/* DELETE STACK */
/* test whether stack machine is initialized                              */
/* test for existence of stack with NAME__OF__STACK in stack table        */
/* delete envelope and remove stack entry from stack table                */
/* set ERROR__CODE */ RETURN;

ESTIST: ENTRY(ESTIMATED__SIZE,ESTIMATED__NUMBER,GUESS,ERROR__CODE);
/* ESTIMATE STORAGE REQUIREMENT */
  DECLARE (GUESS,ESTIMATED__SIZE,ESTIMATED__NUMBER)BINARY FIXED(31);
      GUESS = (ESTIMATED__SIZE+4)*ESTIMATED__NUMBER+20;
/* set ERROR__CODE */ RETURN;
END CREAST;
```

Sample Listing 4.7. The PL/1 stack machine. The module for saving and restoring stacks.

```
SAVEST: PROC(NAME_OF_STACK,ENVELOPE_ADDRESS,ERROR_CODE);
/* SAVE STACK */
/* test whether stack machine is initialized                          */
/* test for existence of stack with NAME_OF_STACK in stack table      */
          ENVELOPE_ADDRESS = ACTUAL_STACK;
/* remove stack entry from stack table and delete it;                 */
/* set ERROR_CODE */ RETURN;

RESTST: PROC(NAME_OF_STACK,ENVELOPE_ADDRESS,ERROR_CODE);
/* RESTORE STACK */
/* test whether stack machine is initialized                          */
   ACTUAL_STACK = ENVELOPE_ADDRESS;
   IF ACTUAL_STACK→OWNER≠STACK_MACHINE_NAME THEN DO;
      /* set ERROR CODE */ RETURN;END;
/* test for non-existence of stack with NAME_OF_STACK                 */
/* allocate stack entry and insert stack with stack entry             */
/* into stack table;                                                  */
   STACK_ENTRY.INTERNAL_NAME = ACTUAL_STACK_ENTRY;
   NAME_OF_STACK.EXTERNAL_NAME  = STACK.EXTERNAL_NAME;
   NAME_OF_STACK.INTERNAL_NAME  = STACK_ENTRY.INTERNAL_NAME;
   NAME_OF_STACK.GENERATION_TIME = STACK_HEADER.GENERATION_TIME;
  /* set ERROR_CODE */ RETURN;
END SAVEST;
```

The first module represents the operations which are related to stack machine management. Initialization and termination make the stack machine available as an operable resource in the environment. These two functions would bracket any other utilization of the stack machine. The second module is related to the lifetime of the stacks (see Sample Listing 4.5). Creation and deletion of a stack (Sample Listing 4.6) would bracket all other operations with the stack. The third module is related to the saving and restoring of stacks (Sample Listing 4.7). Note that although a stack is no longer accessible to the stack machine after saving (until it is restored), the stack process is considered to continue. The saved stack carries with itself the identification given by the program that created it. In a hidden way, the saved stack also contains within itself the unique identification given to it by the stack machine (the creation date and time). Thus saved, stacks maintain their identity on whatever storage medium they may reside, and they can be deleted legally only by the stack machine. Due to the method of implementation, however, we cannot guarantee that a saved stack will not be "killed" illegally by another process which has access to its representation (the program which issued the save call, or a human operator using utility programs of the operating system to delete a data set which represents a saved stack). The last module (Sample Listing 4.8) is related to the proper stack functions of an abstract stack machine: push, pop and empty.

Sample Listing 4.8. The PL/1 stack machine. The module for the abstract stack functions.

```
PUSHST: PROC(NAME_OF_STACK,OBJECT_SIZE,OBJECT_ADDRESS,
                                                HELP,ERROR_CODE);
/* test whether stack machine is initialized                              */
/* test for existence of stack with NAME_OF_STACK in stack table          */
   /* PUSH ITEM ON STACK */
     DECLARE 1 OBJECT BASED(OBJECT_ADDRESS),
                  2 SIZE BINARY FIXED(31),
                  2 CONTENT CHARACTER(OBJECT_SIZE REFER(OBJECT.SIZE));
     DECLARE ITEM_SIZE BINARY FIXED(31);
     DECLARE 1 ITEM OFFSET(TOP),
                  2 IS_BELOW OFFSET(STACK),
                  2 SIZE BINARY FIXED(31),
                  2 CONTENT CHARACTER(ITEM_SIZE REFER(ITEM.SIZE));
     DECLARE NUMBER_OF_ATTEMPTS BINARY FIXED(15) INIT(0);
/* test whether stack machine is initialized                              */
/* test for existence of stack with NAME_OF_STACK in stack table          */
        ON AREA(STACK) BEGIN;
            IF NUMBER_OF_ATTEMPTS = 0 THEN DO;
                NUMBER_OF_ATTEMPTS = 1;
           /*try to compress stack by allocating a new envelope          */
           /*with the same storage size, copy the old envelope           */
           /*into the new envelope, update the stack entry in            */
           /*stack table and delete the old envelope that caused         */
           /*the area overflow*/ END;
            ELSE BEGIN;
                  DECLARE ALLOWANCE BINARY FIXED(31);
                  OPERATING_STATE = 'IN REPAIR';
                  CALL HELP(STACK.SIZE,OBJECT_SIZE,ALLOWANCE);
                  OPERATING_STATE = 'EXECUTABLE';
                  IF STACK.SIZE≧ALLOWANCE THEN DO;
           /*set ERROR_CODE and RETURN*/ END;
                ELSE DO;
           /*try to resolve problem by allocating a new envelope         */
           /*with new ALLOWANCE; copy the old envelope into the          */
           /*new envelope; update the stack entry in the                 */
           /*stack table and delete the old envelope that caused         */
           /*the area overflow*/ END;
        END /* of AREA exception handling */;
        ALLOCATE ITEM IN(STACK);
   /* set ERROR_CODE */ RETURN;
POPST: ENTRY(NAME_OF_STACK,OBJECT_SIZE,OBJECT_ADDRESS,HELP);
/* POP ITEM FROM STACK */
/* test whether stack machine is initialized                              */
/* test for existence of stack with NAME_OF_STACK in stack table          */
   DECLARE NULL_OBJECT_SIZE BINARY FIXED(31) STATIC INIT(0);
/* if stack is empty set OBJECT_SIZE = NULL_OBJECT_SIZE;                  */
/* otherwise set OBJECT_SIZE = ITEM.SIZE, allocate object, copy           */
/* ITEM.CONTENT into OBJECT.CONTENT, delete ITEM and readjust TOP         */,
/* set ERROR_CODE */ RETURN;
EMPTST: ENTRY(NAME_OF_STACK,YES,ERROR_CODE);
/* IS STACK EMPTY ? */
   DCL YES BIT;
/* test whether stack machine is initialized                              */
/* test for existence of stack with NAME_OF_STACK in stack table          */
/* YES = true if found; otherwise YES = false                             */
/* set ERROR_CODE */ RETURN;
END PUSHST;
```

4.3.4 Distributed Systems

Much attention has been devoted recently to the concepts of distributed CAD systems. The basic idea is simple. We have a small local computer (in the design office) for doing that part of the work which has to be done fast and can be done with the limited resources of the local computer. (Note that we use the terms "local" and "remote" with respect to the user – the local computer – being in the design office, and the remote computer at some other place. In other literature these terms are often used with respect to the central computer, which is then considered as local while the attached satellites are remote.) If we need more computer power, we connect to the big computer in the computer center and submit the task for processing on "big brother". This central computer may in turn be backed up by a network of computers. However, the realization of this concept turns out not to be quite so simple. Even if there were no problems in connecting two computers (generally of different manufacturers) to each other in such a way that they can exchange messages, essential problems on the user level have to be resolved:

- Which tasks should be executed on the local computer? Which ones remotely?
- Should there be a choice of executing a specific task either locally or remotely? If so, the corresponding functions would have to be available on both machines (redundancy of functions).
- How do we split the database? Which parts of the data base should be kept as copies on both computers (redundancy of data)?

In most cases the dominant criterion is the response time. In order to decide the above questions, an estimate of the response time for the individual functions (as a function of the characteristic data supplied to the functions) would be required. Even for a dedicated local computer (with no other users competing) this is difficult, and even more so in the time-sharing environment of a computer center. The processing time (for rotation of a three-dimensional object or for searches in a data base) depends very much on the amount and complexity of the data. On the remote central computer, the work load coming from many users is a highly varying function and is hardly predictable in a response time estimate. Thus, a suitable split of the tasks between local and remote computers is often based on practical experience rather than objective criteria. Cullman [CULL80] has proposed an interface architecture between a host and a satellite computer, which principally allows for run-time adjustment of the work. Cullman proposes to implement the respective functions on both the satellite and the host and to call upon the particular version of the function which may be expected to give better performance. Some parameters which characterize the actual work load on both processors could be used by the interface system to decide autonomously (that is without being helped by the operator) whether the host side or the satellite side version of a function should be used.

The design of distributed systems requires much preplanning of the distribution of processes among the participating computers [BRIN73], [YAU__81]. Hence, it is not surprising that the architectures of distributed CAD systems show more variety than CAD systems based on a single computer. However, the three types shown in Fig. 4.16 through Fig. 4.18 may serve as a reference.

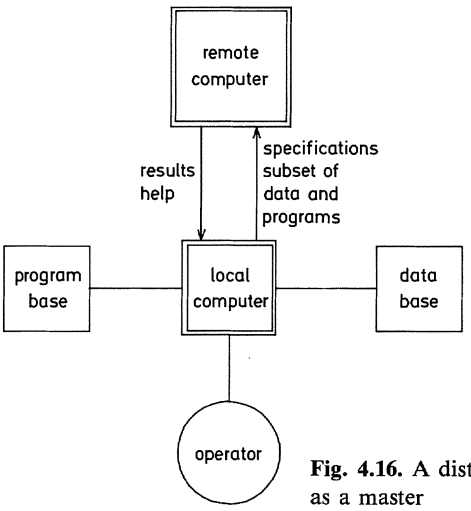


Fig. 4.16. A distributed CAD system with the local computer as a master

Figure 4.16 represents a local computer with remote backup. The local computer is the master, the remote computer a slave. Data base and program base are concentrated on the local computer. For the activities which require large computational resources, programs and data are sent to the remote computer (as an alternative, the programs may have previously been implemented in the remote computer's library). A typical application of this sort is the finite element analysis of a structure which has been designed interactively on the local computer. The remote computer is operated in batch mode. Large amounts of information are exchanged between the two computers before the start and after the termination of the remote task. The response time of the programs ("number crunching" like finite element or finite difference analyses, or operations on large data bases) generally forbids interactive operation. The advantages of this arrangement are simplicity and fast response to the user for many activities. The problems which are often encountered are:

- the methods for maintaining a growing data and program base may turn out to be less powerful on the small local computer than on the large remote one. Either the user approaches the limits of his small computer before he can satisfy his needs, or (if possible) the small computer soon grows and becomes a big one itself. In this latter case the "small" dedicated local computer loses its desired simplicity, and requires more money and computer expertise for its operation and maintenance than had originally been anticipated.
- If more than one user at different locations should want to work with the same program and data base, this arrangement is inadequate.

The concept shown in Fig. 4.17 does not produce the problem of limited computing power and storage capacity. Similar to time-shared usage of a CAD system on a big central computer directly from a terminal (without an intermediate local computer), this concept is well suited for a greater number of users at different locations. Compared to the centralized approach, however, it has the advantage that prior to a

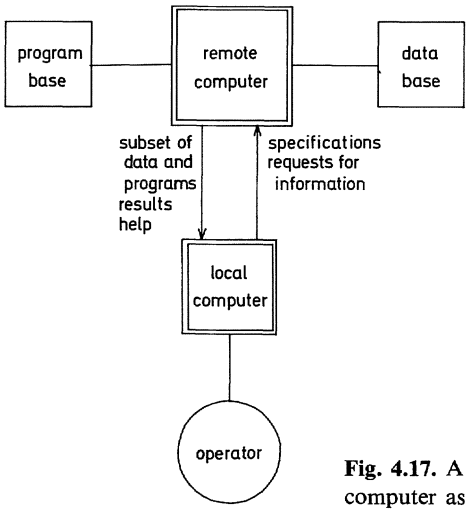


Fig. 4.17. A distributed CAD system with the remote computer as a master

sequence of activities all (or most) of the related programs and data may be transmitted to the local computer. Thus we avoid the need for exchanges of information across the connection line for even the most primitive operations. The gain in response can be significant. This benefit has to be paid for:

- Since the transmitted part of the data base is no longer under the control of the central data base management system, the problem of consistency arises. Different users may work on different copies of the same part of the data base and make changes which are not consistent with each other. Features (possibly of an organizational nature) must be added to the system to deal with this problem, thus increasing the level of complexity.
- More preplanning is required. The user cannot freely follow his intuition when working at the local computer. He can work (efficiently) only with the subset which has previously been copied from the remote computer. Thus the possibility of switching between activities (such as information retrieval, drafting, and calculation), which is typical for design work, is reduced.

Figure 4.18 shows what appears to be the solution of the dilemma. The logical program and data bases have been split into a local and a remote part according to the resource requirements (response time, processing, and storage capabilities). The problem with this concept, however, is that it does not find adequate support in reality. It is still a subject of much research work going on [SCHI79]. If despite the undeveloped methodology an attempt is made to implement such a system, one will likely be confronted with the fact that data base support, programming languages, and other tools are not compatible on the two computers. Methods that have been developed in the area of process control systems may possibly be borrowed to the advantage of distributed CAD system development [RAMA81]. In this case, a separation once defined is likely to be fixed forever because otherwise much programming would have to be redone (consider two different database management systems on the two computers, PASCAL on the big computer, FORTRAN on the small one). Por-

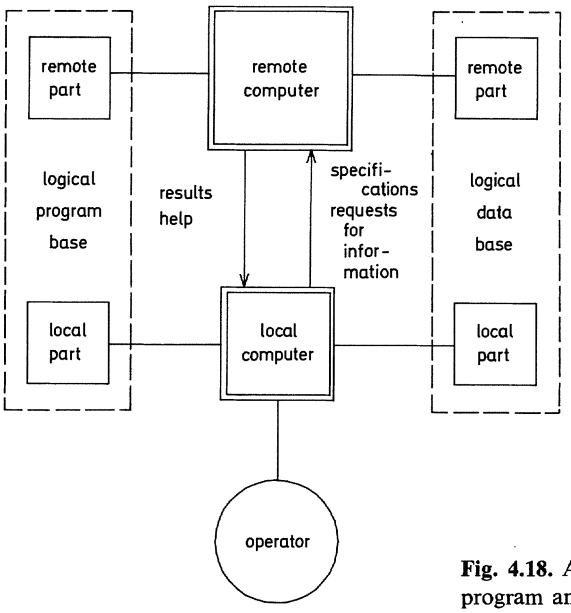


Fig. 4.18. A distributed CAD system with split program and data bases

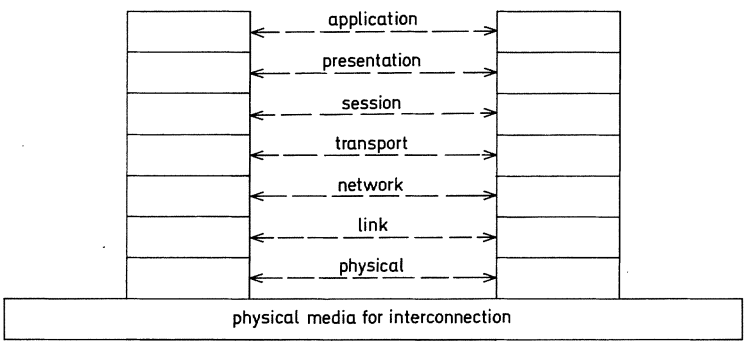


Fig. 4.19. The seven-layer model for connecting two systems

tability of system functions from local to remote and vice versa is important because, as experience with the implemented system grows, the need to adapt the response behavior to changing user attitudes and changing computer loads will arise. A system which is split between two computers is probably much less portable to other installations than a system based on one computer only (at least in the general case of two different computer manufacturers).

Nevertheless, the potential benefits of a distributed CAD system require more consideration and research. The architecture developed in [OSI_78] (see Fig. 4.19) is promising. Seven levels (or layers) of processes have been identified. The highest layer would correspond to one or more CAD processes; the lowest level represents the hardware which connects the computers. The need for subschema transformations

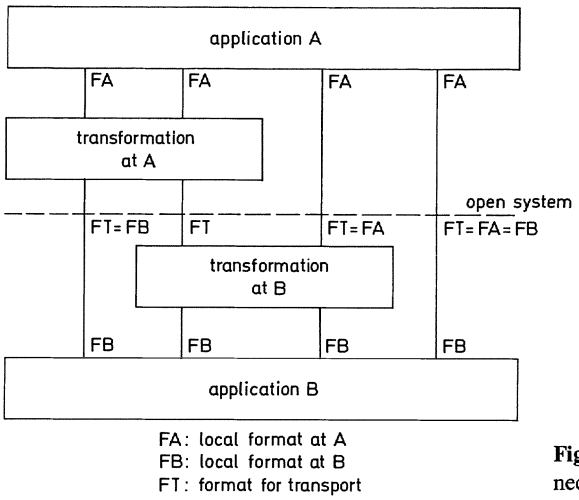


Fig. 4.20. Transformations in connected systems

(see Sect. 3.3.3) is recognized in the concept. Figure 4.20 indicates that subschema transformations may take place:

- not at all, if the same subschema is used on both computers;
- on one computer or the other; or
- on both computers, if the interconnection between the components requires a different subschema for the transport itself.

There is a remarkable agreement between the Open Systems concept and the CAD process model as derived in Chapter 3 (Figs. 3.7 and 3.8) with respect to the importance assigned to management functions. A layer in the Open Systems concept is characterized by a varying number of "application-process entities" (say, abstract processes) and a single "management entity" (say, the environment process). The "application-process entities" have direct relations to corresponding entities on higher and lower levels; the "management entity" is linked to a "system-management entity" (Fig. 4.21). The similarity between the ISO Open Systems concept and the concepts of CAD processes and CAD software machines (see Chap. 4.3.1) suggests that this proposal will suitably support distributed CAD processes (once it becomes available). At present, however, designers of distributed CAD systems have to deal with incompatibilities of computer systems on all levels (even down to the hardware), and either must refrain from utilizing the benefits of distributed processes or must themselves provide interfaces at low levels instead of concentrating on their actual problem: the optimal split of the process functions on the user level.

4.3.5 The Graphical Kernel System GKS as a Software Machine

4.3.5.1 The Process Aspect in GKS

The design of the Graphical Kernel System GKS has been influenced considerably by the ideas described in Chap. 3 with respect to processes, and by the concepts

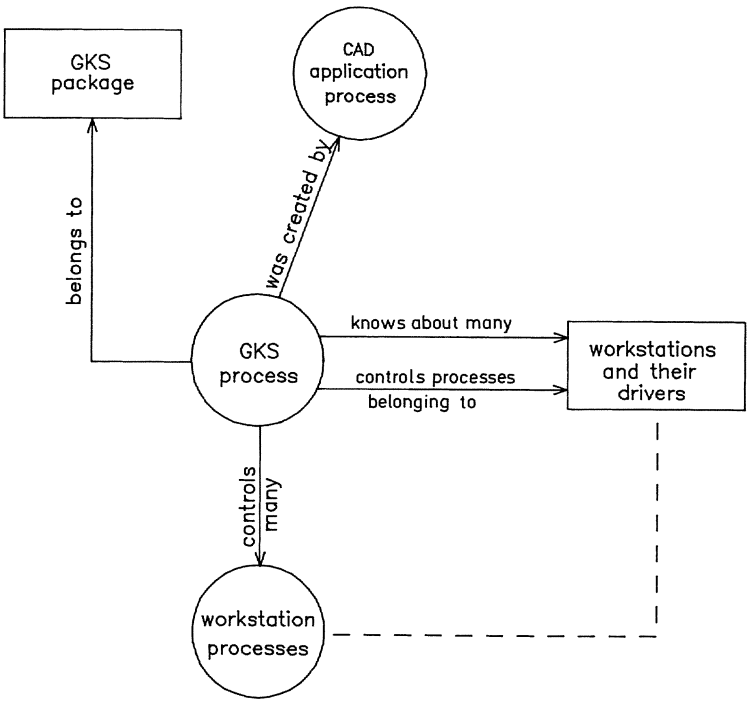


Fig. 4.21. The main processes and environments in GKS applications

developed in this chapter regarding software machine design. GKS is indeed a typical software machine. It has been specified functionally in the GKS standard for a wide range of applications (not only CAD) on an abstract level, independent from any particular programming language – though the GKS designers (more precisely, the designers of the specification) always kept in mind that it should be implementable in a FORTRAN environment. During its development, nobody could foresee the possible resource conflicts which might arise in future application programs, when GKS would start to compete with other software components for resources. Hence, it was necessary to consider the resource aspect as an essential part of the GKS functional specification.

The application program using GKS obviously has to know about the existence of the GKS package in its environment. This condition is achieved by appropriate programming (like the use of the correct names of GKS functions) on the programming level, and by binding the corresponding GKS subroutines into the executable application module. In the sense of Chap. 3 (see Fig. 3.8) the GKS package is an environment that is known to the application process. The application program may request the creation of a GKS process from the package (called OPEN GKS in GKS terminology); it may modify the state of this process by calling upon different GKS functions, and it may finally terminate the process by CLOSE GKS. As part of the GKS process state, several subprocesses may be created, each one corresponding to an individual workstation. These subprocesses have names (the workstation identifiers), while the

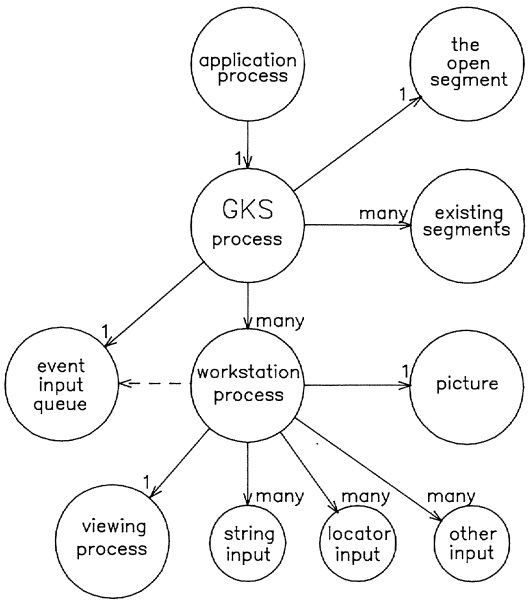


Fig. 4.22. The subprocesses under GKS control

GKS process itself does not need an explicit name: only one GKS process may exist in an application program. Via GKS and a workstation process, the application program may address the drivers of graphical hardware. Both the graphical hardware and the associated drivers constitute part of the computer environment in which the application is executed. The knowledge about the capabilities of the various types of workstations in a computer environment is passed to the application program via GKS as the "workstation description table". Figure 4.21 illustrates this situation; it is a special case of the more general schema shown in Fig. 3.8.

In Sect. 3.2.4.3 (Fig. 3.19) we stated that each process has a certain state representation. We will now refer to Fig. 4.22 for the discussion of the subprocesses in GKS. The environment aspect has not been included in this figure in order not to make it overly complicated. In the GKS standard, the state representation of the *GKS process* is defined as the "GKS state list", and each workstation process has a "workstation state list". Among other information, the workstation state list contains the definitions for visualization of the graphical primitives.

A subtask of the GKS process is the management of segments. A particular segment is the *open segment.* The lifetime of the *open segment process* is bracketed by OPEN SEGMENT and CLOSE SEGMENT. This latter statement transfers the newly created segment process into the set of the *existing segments*, where it continues to exist until it is deleted. A second way to create segment processes under GKS control is to read segments which were created in previous GKS applications from a GKS metafile.

Another task of GKS is the maintenance of the "list of associated workstations" for each segment. This list may be interpreted as an inversion of a *picture* definition. The notion of a picture is not available in GKS. We regard a *picture* as a collection of segments, and represent the correspondence between a picture and a workstation

by considering the picture as a subprocess to a workstation process. It would be desirable to have the concept of a picture in a Graphical Kernel System in order to be able to treat such pictures as entities (for copying, deleting, etc.). However, very early in the GKS specification work it was decided to use only a single level of naming for identifying graphical entities. The introduction of named "pictures" would have created a second such level. The state of a picture is modified by creating new segments or by reading segments from a metafile while the corresponding workstation is "active", or by copying existing segments from one picture into another.

Each workstation process has (potentially) a number of *input subprocesses*. The input processes are associated with input devices of various types. Each one of the input processes may be uniquely identified by a qualified name consisting of workstation identification, device class, and device number. The state of each input process is characterized by an input class mode (REQUEST, SAMPLE, or EVENT). The GKS standard defines "measure" processes and "trigger" processes in order to describe the state of the input processes in more detail. For the REQUEST and SAMPLE input modes, the input processes are synchronized by GKS. For EVENT input, the input processes operate asynchronously and communicate their event reports through the workstation processes to the *input queue* (see dashed line in Fig. 4.22), from which these reports may be retrieved by the application process via GKS.

One *viewing process* is associated with each workstation. It presents an image of the workstation picture (that is, of the set of segments associated with the workstation) to the operator. The viewing process lifetime equals the lifetime of the workstation process. The state of the viewing process is what the operator can visually perceive on a display screen or a sheet of paper. This image is not necessarily identical with the visual representation of the corresponding picture according to the actual state of the workstation state list. Due to deferred updating there may be a delay, which we interpret as a difference between the states of the workstation process and its viewing subprocess.

4.3.5.2 The Resource Aspect in GKS

As is common in functional specifications, the GKS standard concentrates on abstract functions and does not elaborate on the resource aspect in great detail. Nevertheless, the necessity of dealing with this aspect has been realized to the extent that a whole annex has been devoted to it under the title "interfaces". A number of rules, roughly corresponding to rules (R1) through (R7) of Sect. 4.3.2, are documented as a means of minimizing the potential for resource conflicts in future applications. For instance, every GKS implementation is requested to supply a renaming facility. This feature should eliminate conflicts between GKS names, and names used by other parts of the application program. It concerns both programming time conflicts (names of GKS functions that are visible to the application program) and binding time conflicts (global names of SUBROUTINEs or COMMONs in GKS, which are not visible to the application program). Another rule calls for the description of all additional files (for buffer overflow or GKS module libraries, for instance) as resources that have to be provided by the application process in order to make GKS operable.

But beyond that, it should be noted that the abstract concept of workstations [ENCA__80] has resulted from resource considerations. It was found to be essential to provide both GKS itself and the application process with knowledge about the capabilities of the various types of graphics workstations that are available in their computer environment. The quantification of the graphical resources was intentionally not hidden in the drivers, but made apparent so as to allow the higher-level processes (GKS and application) to adjust their behavior in line with whatever the drivers and the hardware can or cannot perform. The workstation description table is a special case of a quantified resource description, and should be considered a powerful tool in the implementation of portable software machines.

One point where more consideration of resource aspects would have been appropriate in GKS is the need for storage capacity for the graphical information. It would be a significant help for designers of CAD systems using GKS for their graphics part if GKS provided a storage estimate inquiry function, which would pass back the estimated storage capacity needed for a segment of given complexity (number and type of graphical primitives in the segment). As it is, GKS will respond with an error message like "storage overflow" if the resources are insufficient; the application program may then take corrective actions. Instead of correcting and redoing previous work, one would probably prefer to have an estimate of the resource requirements in advance in order to prevent the error condition from occurring at all.

4.4 Summary

In the first paragraphs of this chapter, we concentrated on the components and interfaces of CAD systems. The components were considered under various aspects (hardware, software, functional). The functional aspect was related to the main design activities:

- specification,
- synthesis,
- analysis,
- transformation,
- presentation, and
- evaluation.

The interface between the components has been identified as an important characteristic of CAD system architectures. Different interfaces exist during the development, invocation and application of a CAD system. The development and use of a CAD system is greatly improved if suitable CAD tools (or CAD machines) are available. CAD tools are "black box" components which perform certain tasks in whatever environment they are used. Thus, CAD systems can conceptually be built by assembling a number of suitable tools.

While in conventional design the conceptual model of real objects may be described informally, in CAD the conceptual model must be mapped in a formal way onto the computer hardware. This mapping is done in (at least) two steps:

1) conceptual model → language level, and
2) language level → hardware.

Many languages may be used for this mapping process (programming languages as well as data definition languages of data base management systems). Different languages may offer different capacities for expressing important properties of objects relating to the conceptual model. Not all languages will be equally suitable. Often it is necessary to introduce intermediate mappings (such as when calendar dates are mapped onto integers). In extreme cases, the desired mapping can be achieved only by "mapping around the language", and the programming language becomes more of an obstacle than a help.

Binding was identified as an important step between the definition of an object and its use. The binding process replaces a reference to a name by the reference to the object denoted by the name. Binding may occur at different times: programming, module binding, job preparation, or execution. Flexibility and efficiency are competing criteria which influence the decision of whether early or late binding is preferable.

Besides the representation of objects by data, objects may be represented in algorithmic form. Data modeling and algorithmic modeling both have their pros and cons. The "best" model depends very much on how the model is to be used in practice.

As a key issue of CAD system architectures the resource aspect was described in some detail. The abstract function of a CAD system is essential, but the question of resource management may equally well determine the usefulness of a system in a given environment. Questions like

- Which resources are needed? and
- How does the system try to optimize resource usage?

should not be hidden from the user, but should instead be made evident. CAD tools (or CAD machines) can be freely combined into new CAD systems only when the design of the CAD machines avoids potential conflicts. A set of rules, significantly reducing the conflict problem, has been formulated. A sample problem (a stack machine) illustrates the difference in the amount of work required for the writing of an algorithm in a programming language and for the realization of a powerful CAD machine for the same task.

Distributed CAD systems offer a promising compromise between the use of a dedicated (usually small) CAD computer and the time-shared use of a large central computer system. The benefits of a distributed system (fast response to small problems, back-up by a powerful computer for big problems) must be paid for: distributed systems are much more complex, and portable solution concepts are still in the development stage.

Finally, the Graphical Kernel System (GKS) was discussed under the aspects of processes and software machine design, which had been developed in Chaps. 3 and 4.

4.5 Bibliography

[ADA__79] J. Ichbiah et al.: Preliminary ADA Reference Manual. SIGPLAN Notices 14, 6 (1979).

[ADA__82] J. Ichbiah: Reference Manual for the ADA Programming Language. United States Department of Defense, July 1980, Proposed Standard Document. LNCS 106. New York, Springer-Verlag (1982).

[ALLA78] J. J. Allan III, K. Bø: A Survey of Commercial turnkey CAD/CAM Systems. Dallas, Productivity Int. Corp. (1978).

[BALZ81] H. Balzert: Methoden, Sprachen und Werkzeuge zur Definition, Dokumentation und Analyse von Anforderungen an Software-Produkte. Part I. Informatik-Spektrum 4, 3 (1981), pp. 145–163. Teil II. Informatik-Spektrum 4, 4 (1981), pp. 246–260.

[BART78] W. Bartussek, D. L. Parnas: Using Assertions about Traces to Write Abstract Specifications for Software Modules. In: G. G. Bracchi, P. C. Lockemann: Proc. Information Systems Methodology. Heidelberg, Springer-Verlag (1978).

[BONO82] P. Bono, J. Encarnação, F. Hopgood, P. ten Hagen: GKS The First Graphics Standard. IEEE Computer Graphics and Applications 2, 5 (1982), pp. 9–23.

[BRIN73] P. Brinch Hansen: Distributed Processes: A Concurrent Programming Concept. Computing Surveys 5, 4 (1973), pp. 223–245.

[CULL80] N. Cullmann: Optimized Software Distribution in Satellite Graphics Systems. In: C. E. Vandoni, Proc. Eurographics '80, Geneva. Amsterdam, North Holland (1980).

[ENCA__80] J. L. Encarnação, G. Enderle, K. Kansy, G. Nees, E. G. Schlechtendahl, J. Weiß, P. Wißkirchen: The Workstation Concept of GKS and the Resulting Conceptual Differences to the GSPC Proposal. Proc. SIGGRAPH '80, Computer Graphics 14 (1980).

[GKS__82] ISO/TC97/SC5/WG2 N117; Draft International Standard; Information Processing, GRAPHICAL KERNEL SYSTEM (GKS), 1982.

[GUTT77] J. V. Guttag: Abstract Data Types and the Development of Data Structures. CACM 20 (1977), pp. 396–404.

[HATV77] J. Hatvany: Trends and Developments in Computer-Aided Design. In: B. Gilchrist (ed.), Information Processing 1977. Amsterdam, North-Holland (1977), p. 267–271.

[HESS81] W. Hesse: Methoden und Werkzeuge zur Software-Entwicklung. Informatik-Spektrum 4, 4 (1981), pp. 229–245.

[HOUG80] R. C. Houghten Jr., K. A. Oakley: NBS Software Tools Database, NBSIR 80-2159, Washington, National Bureau of Standards (1980).

[HUEN80] K. Hünke: Software Engineering Environments. Amsterdam, North-Holland (1980).

[INTS75] Integrierte Programmsysteme. Report KfK-CAD 2, Kernforschungszentrum Karlsruhe (1975).

[JACK82] M. A. Jackson: Software Development as an Engineering Problem. Angewandte Informatik 2 (1982), pp. 96–103.

[JENS78] K. Jensen, N. Wirth: PASCAL User Manual and Report. Second Corrected Reprint of the Second Edition. New York, Springer-Verlag (1978).

[KNUT69] D. E. Knuth: The Art of Computer Programming. Vol. 1: Fundamental Algorithms (2nd ed.). Reading, Mass., Addison-Wesley Publ. (1969).

[LATO78] J.-C. Latombe: Artificial Intelligence and Pattern Recognition in Computer Aided Design. Amsterdam, North-Holland (1978).

[LEIN78] K. Leinemann, E. G. Schlechtendahl: The REGENT System for CAD. In: J. J. Allan III, CAD Systems. Amsterdam, North-Holland (1977), pp. 143–168.

[LILL81] F. Lillehagen: CAD/CAM Workstations for Man/Model Communication. IEEE Computer Graphics 1, 3 (1981), pp. 17–27.

[LINC81] W. Lincke: Zukünftige CAD-Anwendungen; Forderungen und Perspektiven. VDI-Berichte 413 (1981), pp. 137–142.

[LISK75] B. H. Liskov, S. N. Zilles: Specification Techniques for Data Abstractions. IEEE Trans. on Softw. Eng. 1 (1975), pp. 7–19.

[LUDE78] J. Ludewig, W. Streng: Überblick und Vergleich verschiedener Mittel für die Spezifikation und den Entwurf von Software. Report KfK 2509, Kernforschungszentrum Karlsruhe (1978).

[NOPP77] R. Noppen: Technische Datenverarbeitung bei der Planung und Fertigung industrieller Erzeugnisse. Informatik Fachberichte 11, Heidelberg, Springer-Verlag (1977), pp. 1–19.

[OLLE78] T. W. Olle: The CODASYL Approach to Data Base Management. Chichester, John Wiley (1978).

[OSI_78] Reference Model of Open Systems Interconnection (Version 4 as of June 1979). Report ISO/TC97/SC16 N227 Paris, Association Française de Normalisation (1978).

[PARN72] D. L. Parnas: A Technique for Software Module Specification with Examples. CACM 15, 5 (1972), pp. 330–336.

[PARN75] D. L. Parnas: On the Need for Fewer Restrictions in Changing Compile-Time Environments. SIGPLAN Notices 10 (1975), pp. 29–36.

[RAMA81] C. V. Ramamoorthy, Y. R. Mok, F. B. Bastani, G. H. Chin, K. Suzuki: Application of a Methodology for the Development and Validation of Reliable Process Control Software. IEEE Trans. Softw. Eng. SE-7, 6 (1981), pp. 537–555.

[ROSS76] D. T. Ross, K. E. Schoman: Structured Analysis for Requirements Definition. Proc. IEEE/ACM 2nd Int. Conf. on Softw. Eng., San Francisco (1976).

[SALT78] J. H. Saltzer: Naming and Binding of Objects. In: G. Goos, J. Hartmanis (eds.), Lecture Notes in Computer Science 60: Operating Systems. Heidelberg, Springer-Verlag (1978), pp. 99–208.

[SCHI79] S. Schindler, J. C. W. Schröder (eds.), Kommunikation in verteilten Systemen. Informatik Fachberichte 22, Heidelberg, Springer-Verlag (1979).

[SCHL78] E. G. Schlechtendahl, K. H. Bechler, G. Enderle, K. Leinemann, W. Olbrich: REGENT-Handbuch. Report KfK 2666 (KfK-CAD 71), Kernforschungszentrum Karlsruhe (1978).

[SCHN78] P. Schnupp: Rechnernetze; Entwurf und Realisierung. Berlin, de Gruyter (1978).

[SCHU76] R. Schuster: System und Sprache zur Behandlung graphischer Information im rechnergestützten Entwurf. Report KfK 2305, Kernforschungszentrum Karlsruhe (1976).

[TEIC77] D. Teichrov, E. A. Hershey III: PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. IEEE Trans. on Soft. Eng. SE-3, 1 (1977), pp. 41–48.

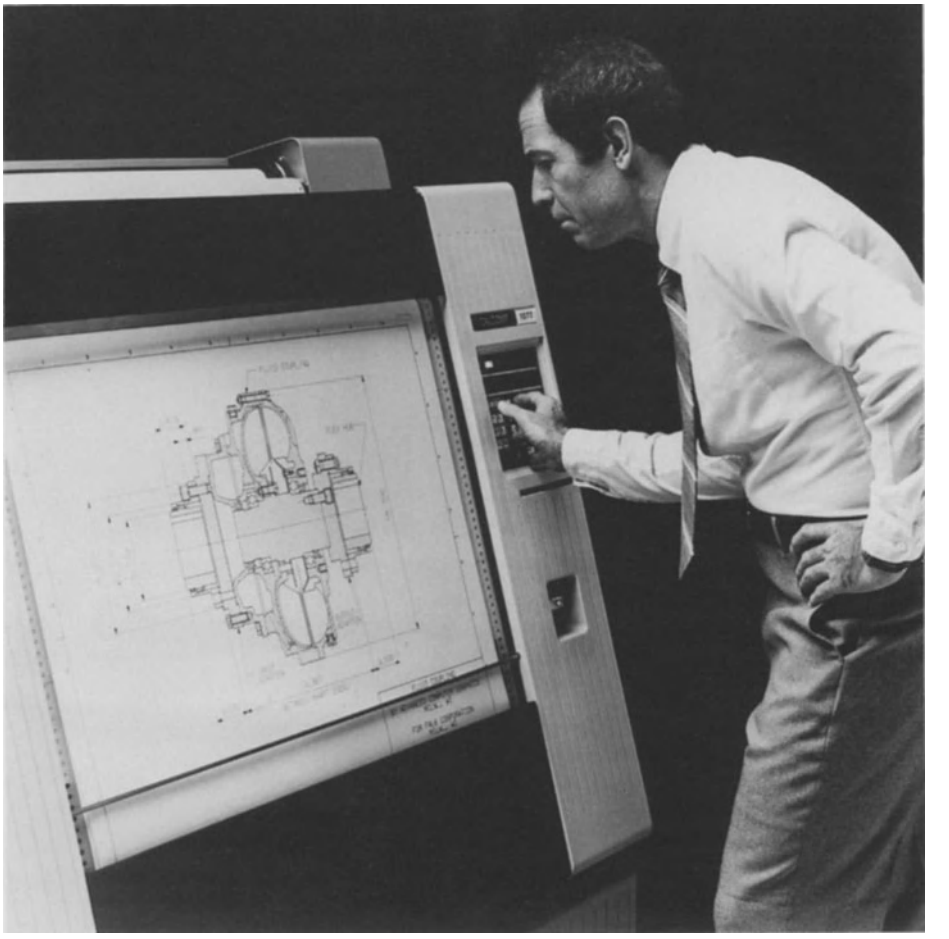
[TOWS79] E. Towster: A Convention for Explicit Declaration of Environments and Top-Down Refinement of Data. IEEE Trans. on Soft. Eng. SE-5, 4 (1979), pp. 374–386.

[VOGE80] U. Voges, L. Gmeiner, A. Amschler von Mayrhauser: SADAT An Automated Testing Tool. IEEE Trans. on Softw. Eng. SE-5, 6 (1980), pp. 286–290.

[YAU_81] S. S. Yau, C.-C. Yang, S. M. Shatz: An Approach to Distributed Computing System Software Design. IEEE Trans. Softw. Eng. SE-7, 4 (1981), pp. 427–447.

[WULF73] W. Wulf, M. Shaw: Global Variables Considered Harmful. SIGPLAN Notices 8 (1973), pp. 18–29, and 3, pp. 226–230.

5 Implementation Methodology



Operating a drum plotter
(courtesy of Calcomp, Anaheim, USA)

5.1 Introduction

Computer graphics hardware and system architectures have been subject to dramatic changes during the past years. At the same time, there has been a great breakthrough in graphics standards, with the aim to obtain a stable interface between expensive graphics software and continuously changing graphics hardware.

In this chapter, computer graphics hardware, system architecture aspects, and (being the archetype of computer graphics standards) the international graphics standard GKS are presented.

It was the intention of the authors to make evident how quickly the world of computer graphics hardware changes; therefore, devices are also presented which have no practical importance any longer. Devices in common use today may be obsolete within only a few years.

The topic of system architecture is treated by describing today's workstations and networks. In the field of workstations, the amazingly rapid development of faster and faster VLSI processors has resulted in an almost uniform architecture for most current workstations. Really dedicated hardware and systems solutions have been superseded by standard firmware implementations profiting from general-purpose CPUs and from high-speed, specialized VLSI processors. Today, networks tend to become the standard solution for flexible architectures of workstations and resources.

In the field of graphics standards, the classical two-dimensional international standard GKS has proved the effectiveness of standards in this field. Today, standards are under development with much higher capabilities. The description of these activities, however, is beyond the scope of this book.

5.2 Computer Graphics Hardware

5.2.1 Introduction

Graphical I/O devices cannot be treated absolutely separate from each other. They are not just pieces of hardware giving system information to the user and user information to the system. They must be observed in their role as one link each in the closed chain of interaction. This chain contains more than the devices themselves. Let us describe it, starting with a system's input to the user. We find the following links in the interaction loop:

- user's senses (mainly eyes, but possibly also ears, touch, etc.);
- user's brain;
- user's means of action (mainly hands, but possibly also feet, voice, eye and head movements, etc.);
- the hardware of the graphical input device (possibly containing firmware and software);
- the input-oriented hardware of the workstation's processing unit;
- the workstation's operating system including input drivers and input sections of the basic graphics software;

- typically, the application software;
- the output sections of the basic graphics software and output drivers interfacing with the workstation's operating system again;
- the output-oriented hardware of the workstation's processing unit; and, finally,
- the hardware of the graphical output device (typically containing software and firmware) with its means of visualization (possibly supported by sound, movement, etc.).

Each link above gets its input from its predecessor (the first link from the last one) and gives its output to its successor (the last link to the first one). Obviously, the loop required for interactive graphics is quite lengthy. It is a chain of links, a pipeline partitionable into a user-specific and a workstation-specific section. The suitability of a certain input or output device for a certain application is therefore determined by its behavior in the pipeline. The pipeline itself is characterized by its slowest link which may be found in the user-specific or in the workstation-specific section of the interaction loop. Any well chosen device will not be a bottleneck in the interaction loop.

Actually, the interaction mentioned above is not as simple as was previously described. In fact, there are also lower loops (e.g., echoing, dragging, etc.) and higher loops (e.g., actions using remote facilities). On the other hand, speed cannot be defined in milliseconds, but is also related to demands like accuracy for geometry and identification.

The preliminary remarks make it obvious that any valuation of interactive graphical devices has to consider certain assumptions concerning the parameters of the hardware/firmware/software in the CAD system, the capabilities of the user, and the target application.

5.2.2 Graphical Output Devices

The term "graphical output" is used in two different senses: as a presentation of a picture and as a pictorial reflection of an application situation. These different meanings result in varying demands on the output devices. We will therefore distinguish between interactive (graphical) output devices for interactive applications and graphical archival devices for those applications where the production of graphical results is not a link in an interaction loop. Of course, applications range continuously from highly interactive tasks (e.g., simulation) to clearly non-interactive ones (e.g., archiving), and thus our distinction is arbitrary to a certain degree.

The suitability of graphical output devices is not as inherently dependent on the performance of the interaction loop as graphical input is. Graphical output has its value in itself, while graphical input has its value in its effect on graphical output. However, certain applications (and CAD is clearly one of them) may require a very fast interaction loop and therefore a very fast graphical output as one link in this loop. Thus, a lot of features must be observed in order to select an appropriate interactive graphical output device:

- Features of the presentation area:
  - resolution (coordinate grid and line width or pixel size),
  - presentation mode (line drawing or raster),

  - size and format of the presentation area,
  - geometrical accuracy (absolute and relative),
  - color capabilities, brightness, contrast,
  - write, refresh and update characteristics;
- Features of the display processor (the display processor may be hardware, firmware, or software or a mixture of all three):
  - virtual coordinate range,
  - refresh manipulation capabilities (support for zooming, panning, and windowing),
  - supported attributes (highlight, blink, line styles, textures, pick enable, etc.),
  - output generation support (line generation, polygon filling, pixel block transfers, z-buffer operations, etc.),
  - input and echoing support (cursors, pick support, etc.);
- Features of the software, driving the output device:
  - specifically, features of the basic software (operating system, device drivers, graphical package) have to be considered, but also the demands of the application; and
- Economic Features:
  - suitability for the application type,
  - availability and costs of device drivers,
  - price and availability of device,
  - expected maintenance efforts (time and costs), and
  - expected lifetime and innovation cycle time.

For the moment, we will focus on the features of the presentation area and on some of the economic features. The features of the display processor will be discussed later, whereas features of the driving software will be mentioned in Sect. 5.5 (The Graphical Kernel System).

In the discussion of interactive graphical output devices it is useful to divide them up into classes, using typical features for the distinction between devices of different classes. In our presentation, the following distinguishing features are used in three levels of hierarchy (Fig. 5.1):

- the principle used for writing (refreshing) the information on the display surface: calligraphic (continuous-line writing on vector-type displays) or scanning (pixel setting on raster-type displays);
- the kind of display used: periodical refresh of the information on the screen from a non-visible storage or visible storage of the information on the screen;
- the technological principle used for making the picture information visible.

In the presented schema of visual displays (Fig. 5.1), the recently still competing technologies (calligraphic storage tube displays and raster-type CRT-displays which dominate in CAD applications) are considered together with the retiring classical calligraphic CRT-vector display, the stillborn hybrid calligraphic laser-vector display, the raster-type plasma panel about to be kicked out, the raster-type liquid crystal panel with its uncertain promises for the future, and "other technologies", the open field of surprises. Within the latter, "electroluminescent powder layer panels" are an example, but these do not have much importance.

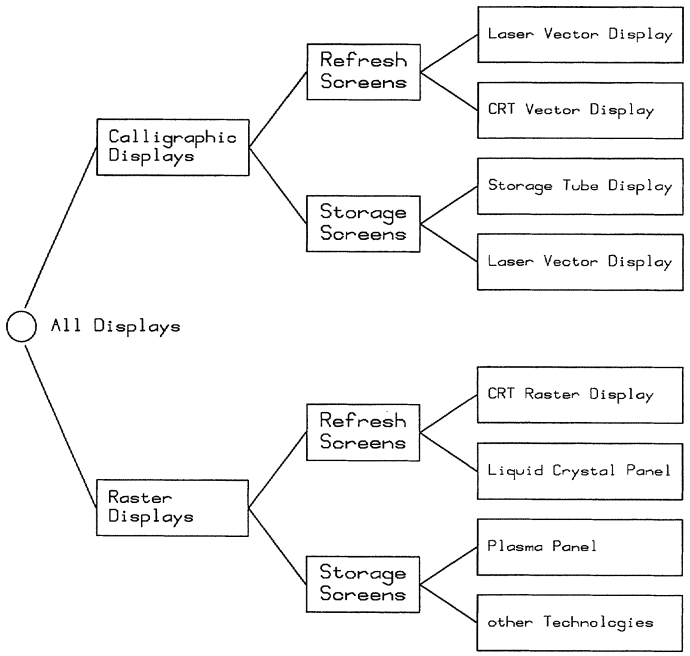


Fig. 5.1. Hierarchy of visual displays

The limited size of this chapter does not allow to describe all technologies for displays in detail. Thus, we shall focus on the advantages of CRT raster technology. Because of its properties this technology is the most promising candidate in the field of interactive graphical displays. In comparing it with its competitors, these will also be discussed to some degree.

5.2.2.1 Refreshing Vector and Raster Devices

In the early days of computer graphics, only calligraphic devices were available (vector plotters and vector screens). When raster devices appeared on the market, for quite a long time they were not considered to be suitable for high quality applications. The low resolution of the screens resulted in such badly jagged lines and edges that no users seriously accepted this low picture quality. Moreover, in the beginning of computer graphics nearly all applications made use of line presentations – partly because of the type of application, and partly because only line-drawing output devices were available.

Today, raster screens are applied in cases where color is used as an essential element of communication information and where filled areas are used. The presentation of "realistic pictures", in particular, is the domain of raster graphics. But also most applications using abstract presentation on line basis (wire frames, production drawings, etc.) have changed over to raster displays now as a result of the dramatically increasing quality of raster displays. It is a fact that raster displays have superseded

vector displays completely within the last few years, making use of CRT raster technology which itself is heavily supported by the high-definition television market expected in the near future.

For quite a long time, raster technology and vector technology had to compete against each other. Both had advantages and drawbacks and both used the same type of output medium, the CRT (cathode ray tube). In CRTs, an electron beam hitting the screen produces a bright spot on the thin phosphor layer inside the tube. The brightness of this spot holds for a short time (some ten milliseconds) even if the electron beam is switched off or moved aside. Thus, by controlling the position of the spot and the intensity of the electron beam, it is possible to keep many positions and even the entire screen bright. The picture on the screen needs to be refreshed in order to avoid visible flicker. As a "fast" phosphor is required for fast changes on the screen, the refresh rate must be 50 cycles per second or even more. Arbitrary intensity control of the electron beam can be performed quite simply at a very high speed while even medium speed arbitrary deflection is much more expensive if it has to be very accurate: deflection is done magnetically, requiring the control of quite large currents through deflection coils. Therefore, deflection is the physical bottleneck for the presentation of pictures on the CRT screen. If done periodically regular, deflection of the beam can be much faster, especially if absolute accuracy is not very important (while relative accuracy is still very high). Raster and vector CRT displays differ by the way in which they use electron beam positioning and intensity control. While CRT vector displays (Fig. 5.2) move the intensified electron beam along the lines to be shown or (with intensity switched off) to the starting point of the next line, in CRT raster displays (Fig. 5.3) the beam is moved in parallel lines over the entire screen regardless the picture to be shown and is switched on at all positions on the screen which are to appear bright. The clock and synchronization generator controls the deflection system of the CRT and – corresponding to the position of the electron beam – the address generator of the pixel memory. The pixel memory must contain the intensity and color information of the picture for every pixel on the screen. This information is generated by the scan converter which breaks down graphics primitives and their attributes to single pixel presentations. The pixel information is read for

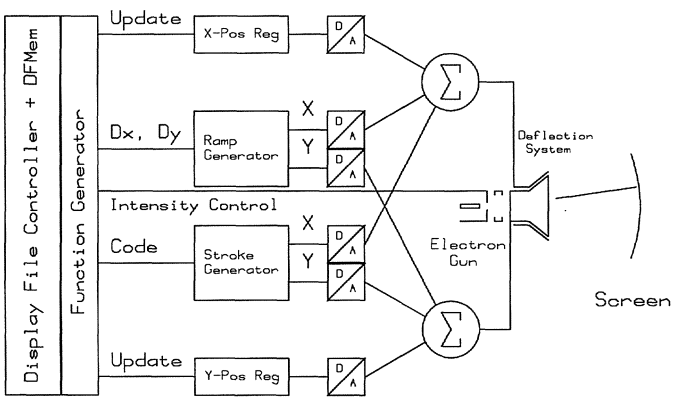


Fig. 5.2. Block diagram of a CRT vector display

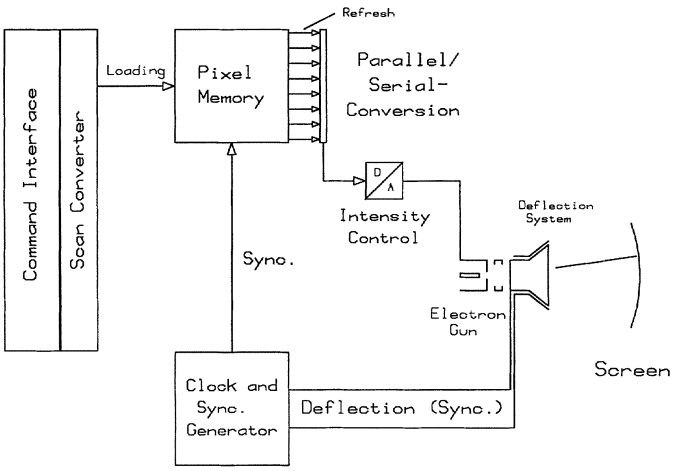


Fig. 5.3. Block diagram of a CRT raster display

refresh several pixels per access and then converted from parallel to serial format. This conversion makes it possible to avoid high-speed demands on memory.

The demand in a CRT vector display for exact random positioning of the beam at high speed and for a medium data rate (start and end point coordinates of the vectors) compares with the demand for a very high data rate (intensity control) and a cyclic and periodically regular beam deflection at a very high speed in a CRT raster display. The problems of vector display technology become evident from the appearance of two different vector generators for very short vectors as required for text (the stroke generator) and for longer vectors (the ramp generator). The quasi-static current position and the increments from the vector generators are converted from digital to analog presentation and fed to the sum points of operational amplifiers which control the deflection amplifiers. The intensity of the electron beam is controlled separately by the function generator.

The very high data rate required in the raster display results in large-scale calculation load for data preprocessing (the so-called scan conversion). Raster technology comes out to be more advantageous than vector technology for the following reasons.

The hardware of raster displays is essentially digital. Raster displays require much less maintenance than vector displays. The deflection amplifiers of the latter need periodical adjustment as they are essentially analog components.

Raster displays benefit from television technology that has produced abundant experience in the field of CRT and associated raster electronics design. The high-definition television standard now under development will support high-resolution raster displays even further. Thus, raster displays are much less costly than vector displays for comparatively similar quality.

Raster displays offer simultaneous access to the entire screen as the electron beam passes every point of it periodically at the refresh rate. Therefore, only the resolution of the pixel grid (number of raster lines on the screen and video bandwidth, i.e., pixel spacing within the raster line) restricts the complexity of the information that can be displayed. On the other hand, vector displays suffer from their limited deflection

bandwidth and are typically not able to fill the entire screen with information without flicker. Moreover, vector displays are not able to modify intensity or color along a vector drawn. Thus, as soon as polygons are to be filled, vector displays are no longer suitable output devices. On the other hand, different colors and intensities are not really effective unless filled areas are utilized.

Especially, since vector displays typically do not offer color and more than about three different intensities, the field of realistic pictorial representation is clearly the domain of raster displays.

The previous list of benefits from raster technology has a complementary list of drawbacks.

The extremely high data bandwidth and the absolutely strict timing required for the intensity control of the electron beam results in a huge amount of data processing. The conversion of application primitives (vectors, circles, polygons, etc.) to picture elements (pixels) for the raster representation is called scan conversion. There are only few and very expensive display systems (simulators) that offer scan conversion in real time (i.e., scan conversion of the complete picture at refresh rate). In the field of CAD, scan conversion is done by preprocessing, resulting in a pixelwise storage of the results in main storage or in a special pixel memory. Refresh of the screen is done from this pixel buffer. Special memory devices offer high-speed pixel output for refresh in parallel to random access for pixel update in a dual port manner and thus lessen the problems of pixel memory management. However, large-scale data processing is required for the scan conversion necessary for any update of the picture. Special firmware and hardware was developed to reduce these severe problems of raster displays (see Sect. 5.3). Vector displays do not need scan conversion and therefore do not suffer from these problems.

Raster displays present the picture after sampling it (scan conversion) previously by giving intensity and color values to the pixels on the screen. This procedure means sampling and is subject to sampling theory which makes certain demands on it. Spatial filtering prior to sampling the scene and after introducing the sample values for regenerating it is absolutely essential for proper display. Actually, in most cases neither the first nor the second filtering is performed, as avoiding the computational expense is given higher priority than guaranteeing proper display in all instances. This results in many annoying effects which are classed under the general term aliasing. Antialiasing is the name given to all algorithmic efforts to reduce aliasing effects. Aliasing effects can be reduced by increasing the pixel density. This is not antialiasing, but just allows for higher spatial frequencies in the original scene. The second filtering mentioned above is performed to a certain degree by the spatial profile of the electron beam's spot on the screen. However, aliasing effects are still a severe drawback for raster displays. They will be nearly overcome by further increases in the screen resolution and by some hardware postprocessing during refresh. There are also some simple and fast algorithmic antialiasing methods which may be applied to scan conversion. Vector displays do not show any aliasing effects like jagged lines and edges or Moiré patterns in dense line structures. The reason why no severe aliasing effects are observed in television technology is that the TV camera performs the first filtering to some extent (CCD cameras do not!).

Updating the picture for a raster display means accessing a lot of separate storage entities (pixels) constituting the changed primitives (vectors, polygons, etc.) from the

application. Additionally, and especially for $2\frac{1}{2}$D and 3D object structures, the whole display file has to be scanned for primitives which may have been hidden by the pixels of a changing primitive. One line being dragged over the screen might otherwise erase a substantial part of the entire picture. On the other hand, updating line drawings is quite simple for vector displays where the refresh is done from the display file directly (and at intersections the intensities of the overlapping lines just add up unnoticed). In any case, the task of picture update requires much more computational power for the raster display than for the vector display.

For identifying objects on the screen, raster displays do not offer a comfortable pick as vector displays do by using the lightpen device. Vector displays can use the hit-signal of the lightpen to interrupt the display processor and thus provide the application with the display file address of the primitive just being refreshed. As raster displays use the intermediate storage of the picture in pixel memory for refresh, this technique is not available there. Instead, only the position of the lightpen can be obtained, and a scan process is required to identify the primitive associated with the pixel at this position. Software techniques (e.g., box lists) give some help in this task. As a consequence of these drawbacks, the lightpen is not used with raster displays. As the complexity of pictures on the screen increases with growing resolution, the lightpen becomes generally inefficient because of its inaccuracy. Therefore, the drawback of raster displays mentioned above is no longer very important.

5.2.2.2 The Storage Tube

While with increasing quality and decreasing cost the CRT raster refresh displays superseded the CRT vector refresh display, another CRT device, the storage tube (Fig. 5.4), has for a long time been able to keep up with both. Different from its competitors, it has no refresh problems. The picture information is stored on a so-called storage grid (non-conducting material on a conducting grid) which is positioned inside the tube, parallel to the phosphor of the screen and very near to it, with the conducting grid towards the phosphor. The storage tube has a special cathode for unfocused electrons besides the conventional one, the source of the electron beam. The unfocused electrons from the special cathode are drawn by the conducting part of the storage grid.

The device makes use of electrons from the two different sources at four different energy levels (the lowest energy level 1 up to level 4, levels 1 and 3 from the special cathode and levels 2 and 4 using the electron beam; see Table 5.1). The energy of the

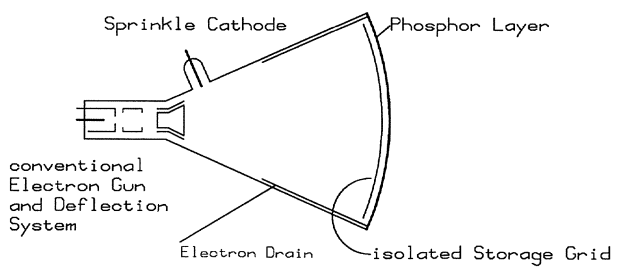


Fig. 5.4. The interior of a storage tube

Table 5.1. The usage of electrons in a storage tube

El.-Energy	El.-Source	Effect on St. Grid	Effect on Screen
low	sprinkle Cathode	None. Electrons are reflected	Stored picture is displayed
medium	normal Cathode	None. Electrons pass grid everywhere	Display in refresh mode
high	sprinkle Cathode	Grid is saturated with electrons	Short flash, then dark screen
very high	normal Cathode	Diminuation of negative load by 2nd el. emission	Writing is brightly visible

electrons is mainly determined by controlling the voltage of the emitting cathode. For acceleration control of electrons from the special cathode, the conducting material of the storage grid may be set to different voltages. Different from the normal CRT, the storage tube does not use the electron beam, but non-focused electrons from the special cathode (lowest energy level 1) for the presentation of the picture. The non-conducting material of the storage grid may be completely loaded negatively by sprinkling it with electrons (energy level 3) from the special cathode. Thereby, any differences of electrical load are cleared away and thus any stored picture information is erased. Energy level 3 is set by raising the voltage of the conducting material of the storage grid. Once the non-conducting material on the storage grid is negatively loaded (erased), electrons at presentation energy level 1 from the special cathode cannot pass the storage grid and the screen remains dark. The electron beam of the storage tube is used for removing electrons from the storage grid by bombarding the grid with high-energy electrons (energy level 4) at those places where the screen is to appear bright. Use is made of the secondary electron emission effect, where any high-energy electron entering the grid gives its energy to more than one electron of the grid, thus enabling them to leave the grid and move to the anode-voltage coating of the tube. As the negative load of the storage grid is diminished at those places where the grid was hit by the writing electron beam, electrons of energy level 1 may pass through the meshes of the grid and make the opposite phosphor luminous. These electrons are not able to enter the non-conducting material and to alter the electrical load on it, though they are able to pass the grid. The energy level 2 of electrons (beam cathode) is used in the so-called "write-through mode", where electrons are able to pass the storage grid even on erased places without being able to enter it. In this mode, the storage tube may be used like the classical vector CRT.

The advantage of the storage tube, in comparison with normal CRT devices, is its ability to present pictures of nearly any geometrical complexity. On the other hand, there are severe disadvantages of this device. Despite the absence of the special maintenance problems and the high costs of the vector CRT refresh devices, the high resolution of the device (a coordinate grid of typically 4000 by 4000 pixels on the screen) is only possible at a relatively low writing speed. Therefore, the maximum amount of information presented in the write-through mode is low, and writing a

complex picture is a lengthy procedure. Any change that removes part of the stored picture must be preceded by a complete erase of the storage grid information and requires redrawing the whole picture. These facts are a severe limitation for the storage tube in interactive mode.

Moreover, the storage tube does not allow different intensities as the writing process has to be done to saturation. Storage tubes do not offer color presentations and really have only poor facilities for structuring the picture or even just highlighting. For these reasons, they have practically disappeared from CAD applications.

5.2.2.3 The Plasma Panel

Another storage-type device used at present is the plasma panel display (Fig. 5.5). It combines the advantage of visible storage with the advantage of partial erase. It offers very good brightness and contrast and inherently does not show any flicker. It is flat and may be transparent, allowing the addition of background information from another source. Color is a severe problem, though not impossible if several layers with different plasma are used. On the other hand, it does not offer intensity control besides on-off. Furthermore, it is of raster type and has most of the disadvantages

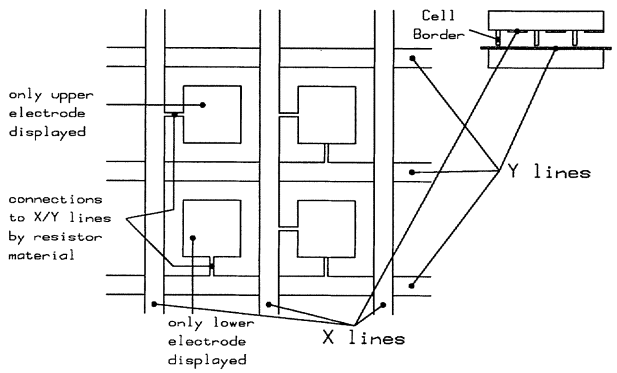


Fig. 5.5. The interior of a plasma panel

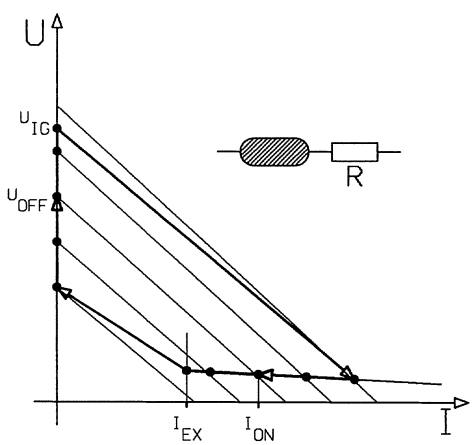


Fig. 5.6. Characteristics of a plasma cell

typical for these devices. Even if its limitations concerning resolution are solved, their update speed problems will remain because of the large relaxation time of the plasma which does not allow fast erase. Also, it has quite a high power consumption. Compared with liquid crystal devices, it does not seem to be very promising for CAD applications. In Fig. 5.6, U_{OFF} is the idle voltage at an extinguished cell, I_{ON} is the idle current at an ignited cell. The two points above respectively below U_{OFF} delimit the voltage range at an extinguished cell, the two points right respectively left of I_{ON} delimit the current range at an ignited cell while other cells in the same row or column of the display are ignited respectively extinguished. U_{IG} is the ignition voltage, I_{EX} is the extinguish current of a cell. R represents the resistance of the two connections of a cell to its X and Y lines.

5.2.2.4 Liquid Crystal Devices

Liquid crystal devices (Fig. 5.7) are still an unknown factor for the future. Though they are of refresh type, their refresh demands are much less critical than those of CRT devices. They are able to present a fairly good scale of intensities and color. They offer a flat screen and a very low power consumption. As they are also candidates for television, they may replace the CRT tube. They share most of the advantages and disadvantages of CRT raster displays. There are still technological problems with resolution and especially with display size. Their practical observation angle is quite narrow, as they operate using polarization. At the moment, they are used in those cases where the low power consumption is an essential advantage, as it is in portable, battery-run personal computers.

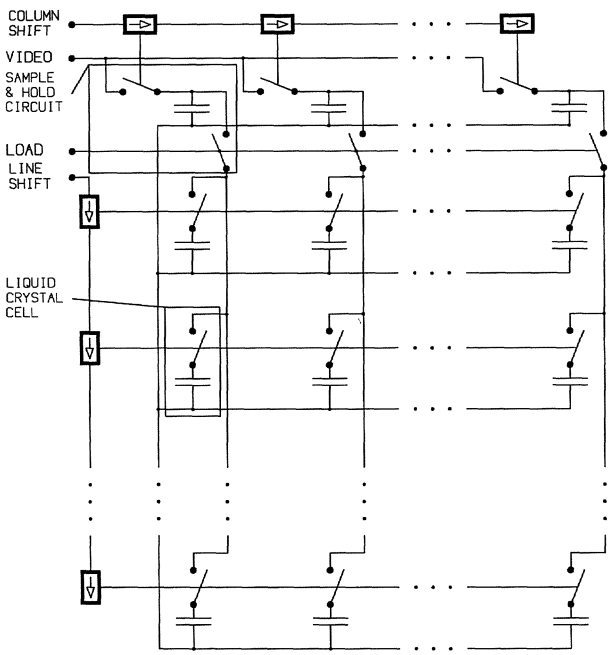


Fig. 5.7. Functionality of a liquid crystal display

5.2.2.5 Graphical Storage

After having described interactive graphical output devices, the function and features of the most important hardcopy and archival devices will be outlined.

These devices cover the range of hardcopy, in a general sense, and inherently graphical storage. Hardcopy is a means of storing pictures (e.g., as a plotter output) in such a way that they are visible without previous processing. Inherently graphical storage (e.g., a video recording on magnetic tape) requires processing to get a viewable result (use of a video recorder to get video output on a screen). The property "inherently graphical" is not strictly defined. An obvious example of non-inherently graphical storage is a graphics metafile written to a magnetic tape: though the stored information describes graphical information, it is just conventional digital data and thus not inherently graphical. On the other hand, it seems sufficiently clear that a video tape, containing in analogous storage mode a sequence of video frames, is inherently graphical storage. Digital optical disks, however, will most probably be fast enough in the near future to store and output video frames at refresh rate. Will they then be inherently graphical or only very fast digital mass storage devices? Subsequently, we will focus on graphical hardcopy devices and keep in mind that in the near future there will exist storage media with random access and capacity to write and read thousands of high-resolution video frames digitally.

Among plotters, those devices producing hardcopy on paper are the most common. The development of these devices reflects the advancements in graphical display devices. While, in the past, plotters were line-drawing devices, today the trend is to raster devices with their inherent capability to fill areas efficiently and to produce shaded color output.

5.2.2.6 Pen Plotters

The classical line-drawing devices operate mechanically and can be distinguished from each other by the movement of the pen and the surface: while flatbed plotters (Fig. 5.8) move the pen along two axes over a flat surface, drum plotters (Fig. 5.9) move the pen along one, the surface on the other axis; a drum functions as the support for the paper and moves it back and forth. While the drum plotter is faster, it is less precise than the flatbed plotter because of the limited adhesion between paper and drum surface, causing slippage. The flatbed plotter additionally allows working on non-flexible material, but is more expensive, heavier, larger, and has a size limit on both axes while the drum plotter may use paper from rolls. Line-drawing plotters are

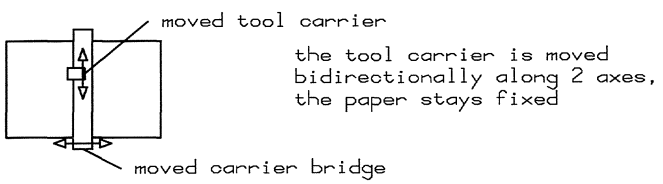


Fig. 5.8. Functionality of a (calligraphic) flatbed plotter

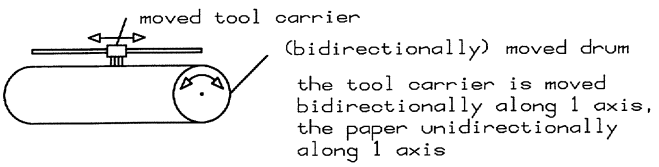


Fig. 5.9. Functionality of a (calligraphic) drum plotter

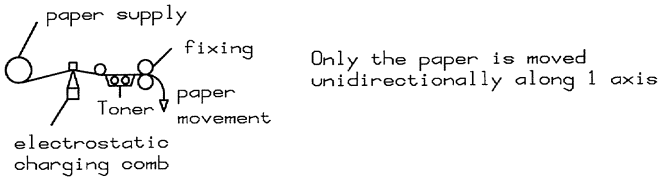


Fig. 5.10. Functionality of an electrostatic raster plotter

not designed to plot filled areas, though they are able to do it by tight hatching. They use different colors and line width by changing pens and thus are limited. Shaded areas cannot be plotted.

5.2.2.7 Raster Plotters

Raster plotters for output on paper may be distinguished from each other by their principle of writing and by their method of scanning. Mainly, there are ink jet and electrostatic plotters. Within electrostatic plotters there are those with electrodes and those using laser beams for influencing electrical load on the surface. Ink jet plotters (Fig. 5.11) and laser plotters (Fig. 5.12) perform line scanning in a serial manner, while electrostatic plotters using electrodes assemble each scan line electronically and output it simultaneously. While ink jet plotters can produce color hardcopy in one run using four ink jets in parallel (yellow, magenta, cyan, and black), electrostatic plotters usually need 4 runs for the four components (the black component is required because it is not possible to produce a pure black by superposing the three basic colors). This time-consuming procedure may soon be overcome by using 4 systems in a pipeline. The problem to be solved is to fix the toner on the paper before it arrives in the next system for processing the next color.

Given powerful processors providing for the pixel data, raster plotters need a fix time for a plot of a given size regardless the picture contents. This property distinguishes raster plotters from line-drawing plotters which produce plots in an amount of time which is directly proportional to the size of the picture contents. Typically, raster plotters are much faster than line-drawing ones, especially if raster plotters with parallel output of the scan lines are used.

The same problems that plague raster displays exist for raster plotters, and in the beginning the quality of raster plots was very poor. Nowadays, with resolutions of

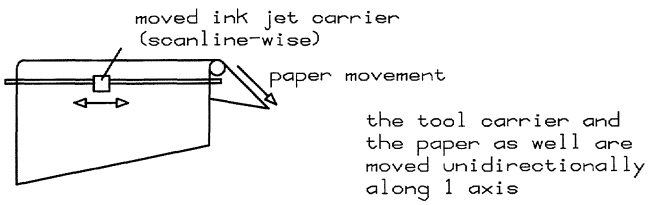


Fig. 5.11. Functionality of an ink jet plotter

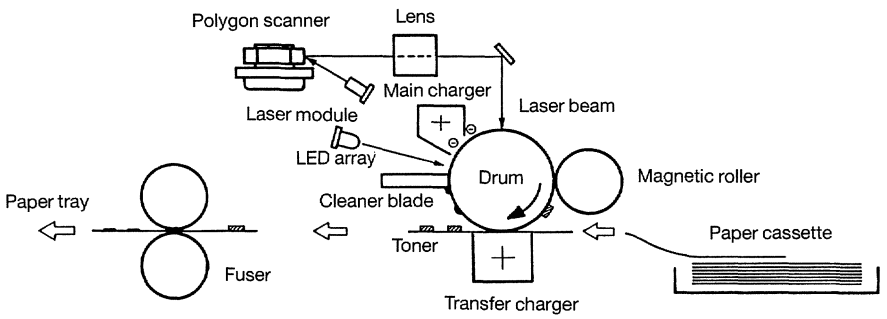


Fig. 5.12. Functionality of a laser printer plotter (courtesy of Kyocera, Tokyo, Japan)

300 pixels per inch and more, the quality is excellent. Ink jet plotters are the low cost version of raster plotters, offering less resolution and much less speed. Higher quality than on paper is possible on film material, where laser plotters offer 250 pixels per inch with a pixel substructure of 24 by 24 subpixels for high-quality typography.

5.2.3 Graphical Input Devices

The term "Graphical Input Device" is mainly used for those peripherals designed for interaction between user and graphical workstation. Later, we will refer to these devices as Interactive (Graphical) Input Devices and thus distinguish them from those used for capturing graphical data. The latter devices will be referred to as (Graphical) Acquisition Devices.

Interactive graphical input devices can be distinguished by the way they operate. Class 1 devices interact with the display itself (e.g., lightpen). Class 2 devices work on a direct mapping of the display (e.g., tablet, digitizer, touch panel), and class 3 devices on an indirect one (e.g., mouse). Class 4 devices update the position of a cursor in increments (e.g., trackball, thumbwheels, cursor control keys), and class 5 devices by controlling its movement (e.g., joystick, 3D mouse, joyswitch). They may even work in a two-step procedure as in the case of a cursor controlling menus. As interactive graphical input is inherently combined with graphical output as a feedback, the importance and suitability of interactive graphical input devices has been heavily influenced by the development of graphical output devices.

5.2.3.1 The Lightpen

The classical interactive graphical input device is the lightpen (Fig. 5.13), the only device of class 1. This device is a good example for describing the influence of graphical output devices on the suitability of graphical input devices. In the early days of computer graphics, high-performance interactive graphics output was exclusively done by vector screens, CRT devices writing vectors randomly. The maximum number of flicker-free displayable vectors was around a few thousand on a large screen. The lightpen, as a light-sensitive device, initiated an interrupt for the display processor when the trace of the refreshing electron beam came sufficiently near to it. Therefore this interrupt was directly time-correlated with the generation of the vector, and the display processor was able to make the appropriate display file address available to the CPU running the application.

Two prerequisites for efficient input operation were met in those early times: very fast identification of a display file element (because of the operating principle of the

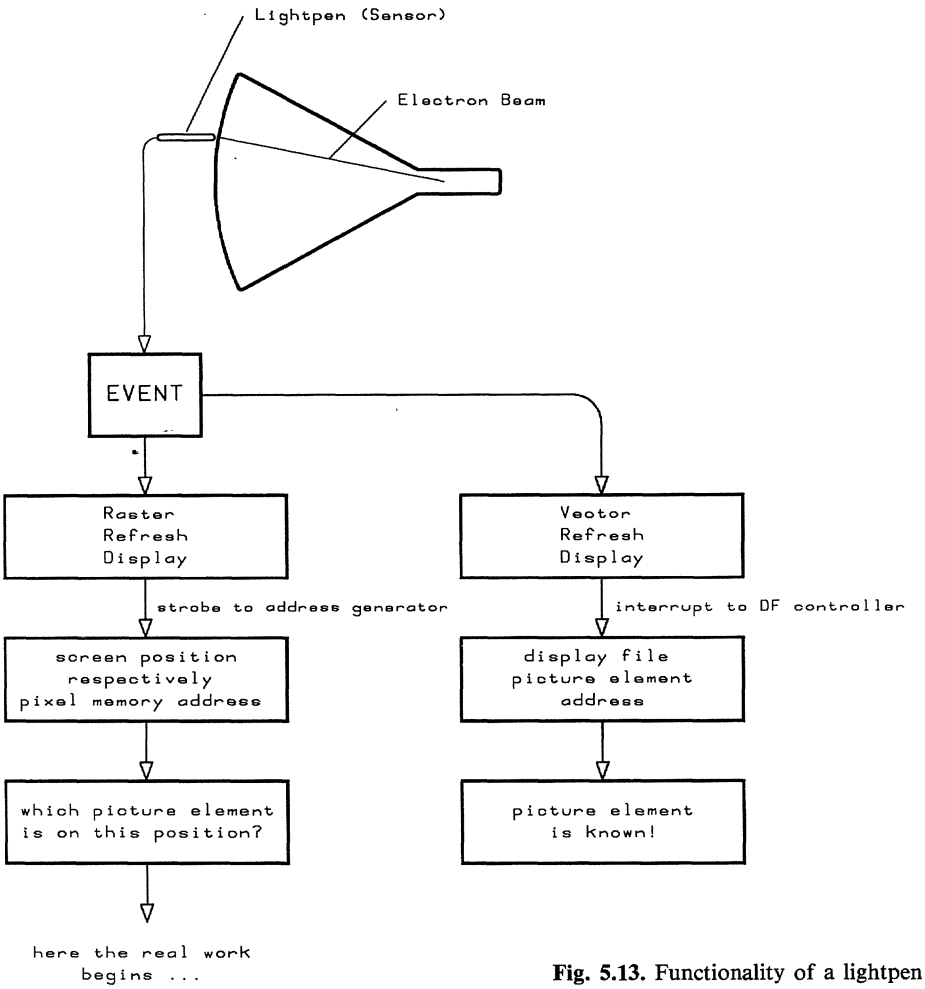


Fig. 5.13. Functionality of a lightpen

lightpen), and low probability of identification conflicts on the screen (because of the low density of picture elements on the screen). Precise positioning with the lightpen has always been a problem. Adjustable coordinate grids on the screen and automatic, program-controlled matching of vertices near to each other were two techniques used to reduce these problems.

When CRT raster scan output devices appeared, attempts were made to keep the lightpen. But the prerequisites mentioned above were not met any longer: the pixels making up picture elements from the display file are not refreshed coherently on the raster screen, and the only information obtained from the lightpen is the position of a pixel on the screen. To be more precise, typically one pixel from a set of adjacent bright pixels is identified: the old positioning problem of the lightpen still exists. Thus, the identification capability of the lightpen is lost and must be simulated by (slow) matching tests in the software. This is the reason why the lightpen has been unused since CRT raster scan output devices were introduced.

5.2.3.2 Tablet, Digitizer and Touch Panel

With regard to the order of directness of correspondence between output and input, the tablet (digitizer and touch panel as well) as a device of class 2 is the next interactive graphical input device to be discussed (Fig. 5.15, Fig. 5.16). The operating principle is positioning: the position of a pen (or a reticule, Fig. 5.14) on a surface is input to the workstation processor. There are many different principles of operation: magnetic

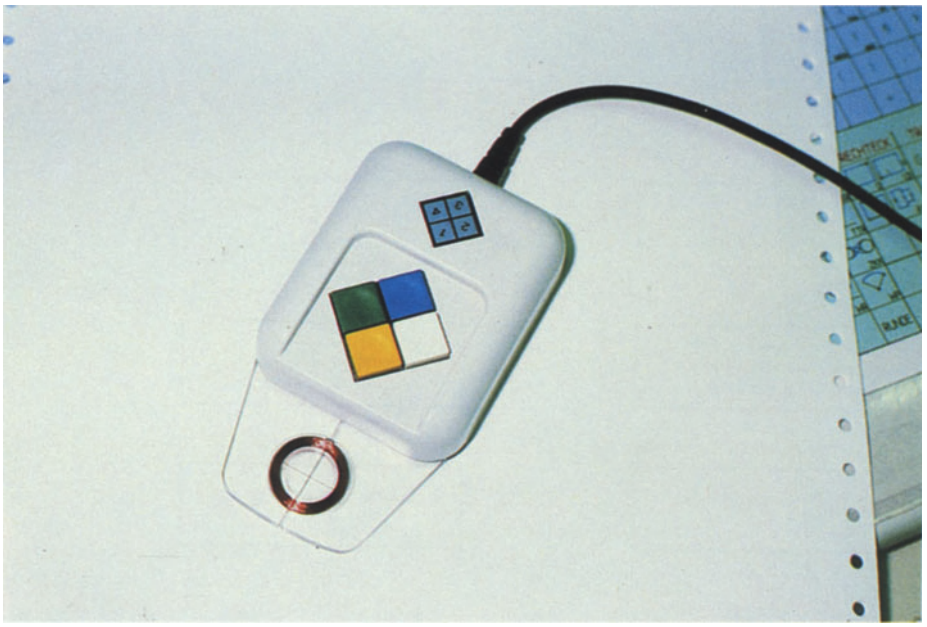


Fig. 5.14. Reticule of an electromagnetic tablet

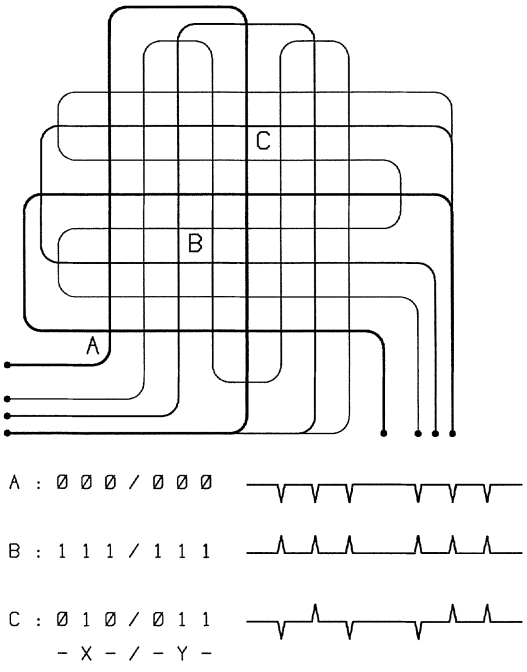


Fig. 5.15. Functionality of an electromagnetic tablet

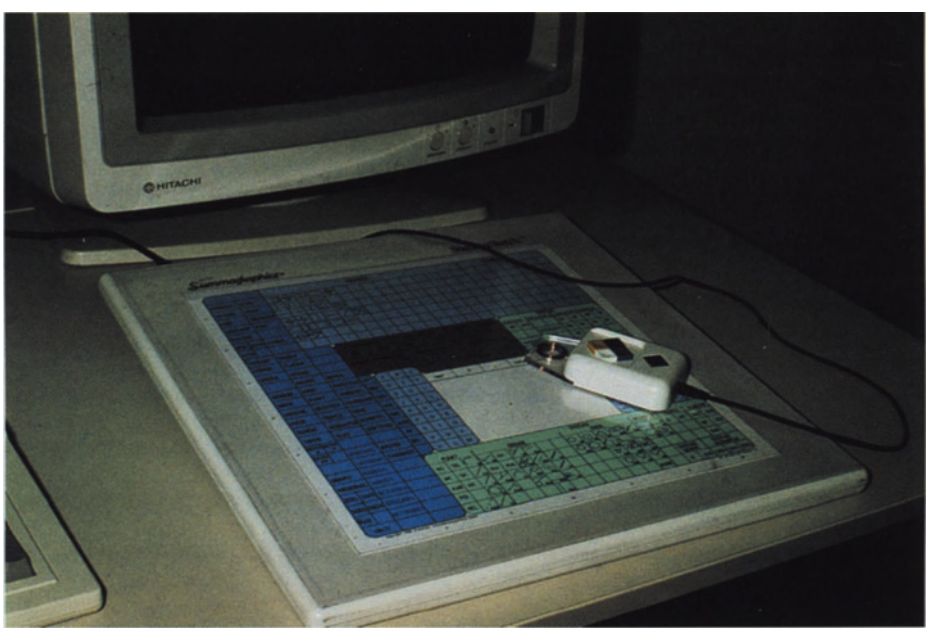


Fig. 5.16. An electromagnetic tablet

operation is the most important one today. Ultrasonic devices and laser devices are already available but are not commonly used. They are not bound to plane surfaces and therefore allow three-dimensional positioning.

One variation on the tablet could have been discussed prior to the classical tablet if it were more important: the touch panel. The touch panel is a tablet device directly on the display surface. The operating principle is optical: two orthogonal rows of light barriers detect the position of the user's finger on the screen and make it available for the workstation processor. The resolution of this device is very low, and therefore it is mainly used for menu selection purposes. For picture element identification or even positioning, this input device is hardly appropriate. It is of no significance to CAD.

The suitability of tablets and digitizers is rather dependent on the application. Their most severe disadvantage is the fact that the user has to concentrate on the device in all cases where he wants to make use of the direct mapping between display and input device surface. As a consequence of this fact, the input loop is broken. The user will tolerate this effect, however, in those cases where he is working on something like digitizing a drawing. In those cases where the user concentrates on the display surface, the digitizer is not suitable because of its bulkiness, and the tablet is used just for cursor control, comparable with the next classes of interactive graphical input devices, those of indirect mapping.

5.2.3.3 Mechanical and Optical Mouse

The classical indirect mapping, interactive graphical input device is the so-called mouse (Fig. 5.17, Fig. 5.18, and Fig. 5.19), a device of class 3. The mouse provides motion increments as it is moved on a surface. The mapping to the display surface is indirect because the mouse may be lifted from the surface and then moved over the surface without issuing motion increments. This fact makes it possible to get along with a quite small working surface for the mouse, even if working on a large display surface with high resolution. Inherently, the mouse is a cursor-positioning device, and therefore the user is not interrupted in his concentration on the display surface. Of

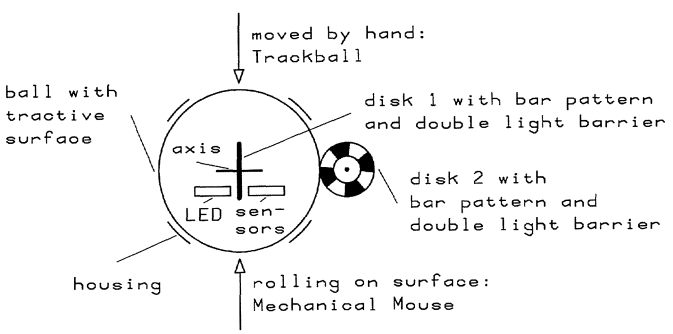


Fig. 5.17. Functionality of a mechanical mouse

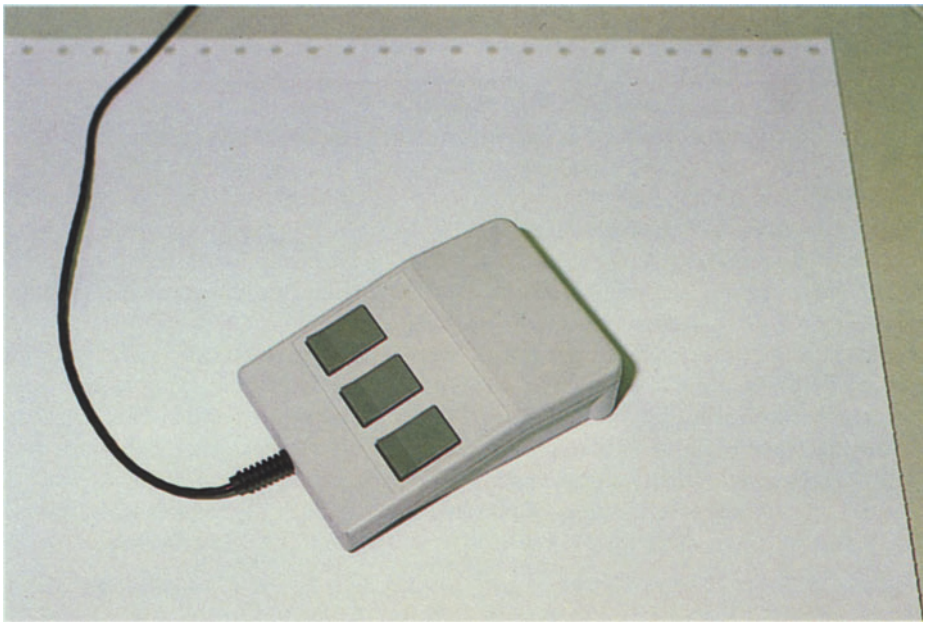


Fig. 5.18. A mechanical mouse (top view)

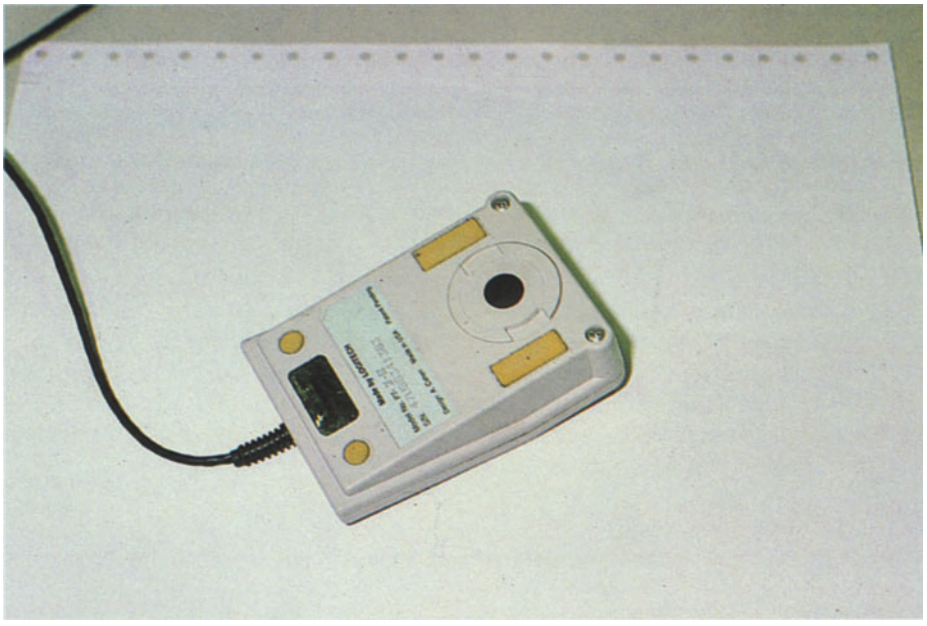


Fig. 5.19. A mechanical mouse (bottom view)

course, for dragging picture elements or for panning, the whole coordinate system may even be connected to the cursor and moved with it.

The classical operating principle of the mouse is mechanical with all side effects inherent in this operation. In the beginning, two little wheels mounted on two orthogonal shafts were turned by moving the mouse over the surface. Connected to the wheels on the same shafts, transparent disks with bar codes printed on them gave signals to the electronic part of the device using the light barrier principle. In the next stage of development, the two wheels were replaced by a metal ball, transferring the motion of the mouse to the bar-coded disks. This replacement diminishes the slippage of the mouse on the working surface, but still, many mechanical components make the device sensitive to dirt and rough treatment.

Alternatively, the optical mouse is available. This device requires a special working surface which contains the bar code, while a simple optical system and sensor electronics is located inside the movable mouse device. The two orthogonal directions can be distinguished by using two colors for the grid lines and correlated colors for two light emitting diodes (LEDs) which are alternately switched on and off. Four sensors forming the vertices of a square are monitored to distinguish the motion of the mouse on the surface. The mechanical problems were overcome by this principle.

5.2.3.4 Trackball, Thumbwheels, Dials and Positioning Keys

Class 4 of interactive graphical input devices is based on incremental cursor-position control combined with a stationary device. Devices of this class are trackball, thumbwheels, and cursor positioning keys.

The trackball has been an alternative choice for the mouse and in fact is a mouse turned upsidedown, using the surface of the user's hand as a working surface. As the user's hand does not embody bar codes, the operating principle of the optical mouse cannot be used for the trackball.

Thumbwheels or dials (Fig. 5.20) may be looked upon as a simplification of the trackball, where the measuring wheels are moved by the user separately and directly. Though the slippage problem can be neglected here, the lack of random motion control is usually a severe disadvantage. On the other hand, the lack of random motion control allows separate motion of the cursor in the direction of either coordinate axis. This ability may be an advantage for several applications (e.g., for 3D operation).

Cursor control keys lack any element of motion, but are extremely simple, and allow any desired accuracy at reciprocal operation speed. They are physically available on any workstation, but frequently this simple input device is not implemented because of the presence of more efficient ones.

5.2.3.5 Joystick, Joyswitch, 3D Mouse and Menus

Class 5 of graphical input devices comprises those devices which allow the cursor to be given a speed in a chosen direction. The classical device of this class is the joystick (Fig. 5.21). A handle rises above the cabinet holding the electronic circuitry; kept in resting position by springs, any displacement of the handle from this position results

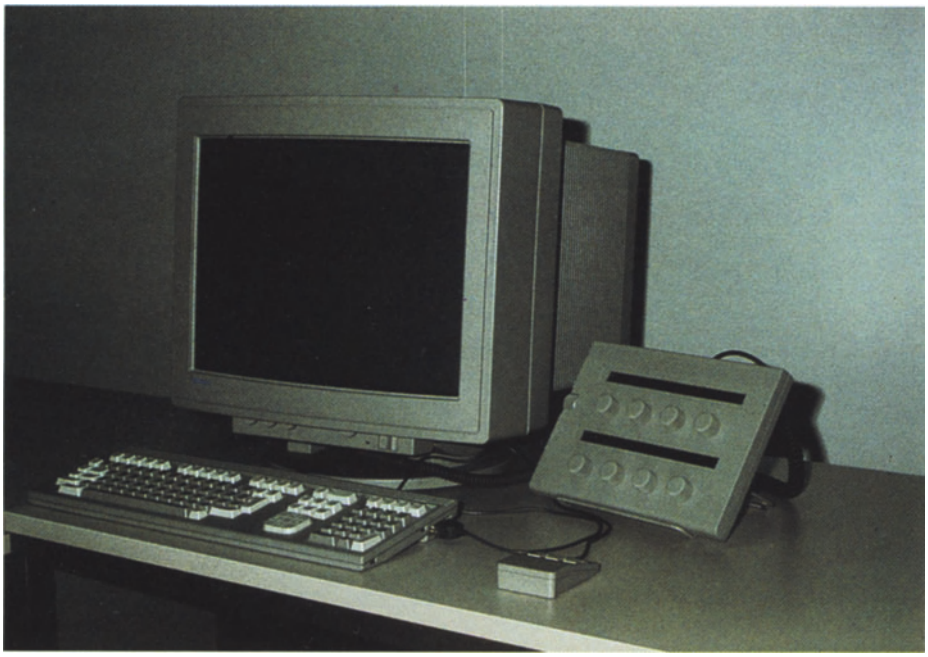


Fig. 5.20. Graphics workstation with dials

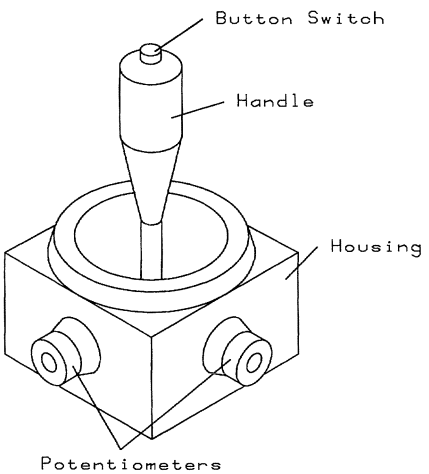


Fig. 5.21. A joystick

in cursor movement in the direction determined by the direction of handle movement, and at a speed determined by the amount of the displacement. Typical for this device is the tradeoff between speed and accuracy of cursor control. With increasing resolution of the output devices and thus growing accuracy demands, the suitability of the joystick is diminished because of the resulting decrease of speed. Nowadays, it is mainly used in computer games where there are low accuracy requirements.

Fig. 5.22. A joyswitch

The so-called joyswitch (Fig. 5.22) has the functionality of the joystick in a very much simplified manner. It is an octagonal key which may be pressed down on any of its eight edges, and combines the function of a field of cursor-positioning keys in repeat mode. It allows cursor movement in 8 directions.

A very new device that looks like a trackball device with the ball completely outside the electronics box, but completely different in operation, controls six dimensions of motion: three-dimensional translation and three-dimensional rotation at the same time. The device reacts with coordinate translation on thrust or traction, and with coordinate rotation on torque. Providing excellent three-dimensional visual feedback, this one-hand device may be very powerful.

Menu fields for cursor control are the latest fashion for user interface managers. They are inherently slow and have even more disadvantages than cursor control keys.

5.2.3.6 Scanners

Interactive graphical input devices are mainly used for control of program flow (e.g., by identifying objects). They are thus general purpose, while graphical acquisition devices are specialized for the capture of data from graphical representations. In the beginning of graphical acquisition, these graphical representations have typically been documents from hand-made archives (drawings, microfilms), and the task was to make these hardcopies accessible to CAD packages and computer-supported archiving. The main problem with this type of graphical acquisition is the extraction of the structural information from the documents. This task starts with feature extraction

and ends with a knowledge-based system containing the rules of the respective picture types, and produces data suitable for further information processing. For standard drawings with strict rules, almost full automation can be obtained today. Nowadays, however, pictures true to nature are the graphical representations to be processed. Wide application fields are found in nuclear physics experimental work, terrestrial observation, production quality verification, and robot control. The principle of processing is the same as outlined above, but because of the much higher complexity of the pictures, feature extraction and especially the rules in the knowledge-based system are much more complicated. Up to now, this task has required massive human-expert interaction and is far from full automation.

We shall restrict ourselves to the hardware section of the hardware/software task of graphical acquisition. This decision leads us to the description of devices supporting raw data input.

Today, devices for supporting raw picture data are of the scanner type. For low and medium resolution, video cameras are appropriate. For high resolution, parallel sampling of scan lines with dedicated sensors is used, and for very high resolution, expensive mechanical scanners are required, making use of laser technology for extremely high performance.

Of course, there is a trade-off between the resolution and speed of these devices. Additionally, mechanical operation in one or two dimensions slows down the scan procedure dramatically. While frames with a size of about 500 by 500 sample points can be scanned in 10 milliseconds (which means 25 megapixels per second) using a CCD camera, high resolution scanning with 2000 by 2000 sample points can be done in 10 seconds (line-parallel sampling with a rate of 400 kilopixels per second) by a telefax device; very high resolution scanning with about 30000 by 30000 sample points requires 15000 seconds (about 4 hours with about 64 kilopixels per second) on a laser scanner. This technique, however, has little significance to CAD at this time.

5.3 Graphics Workstations

CAD workstations in the context of hardware equipment are no longer simply sets of modules specialized for CAD requirements. Since graphical representation is commonly used at the user interface of general purpose applications, and since the quality of graphics output on raster screens has been increased dramatically, the same general purpose machinery is used for CAD as well as for many other tasks. The typical appearance of past CAD workstations with their two different screens (alphanumeric and graphics) and their large peripherals (digitizers, plotters) is now history. Today's raster screens, in combination with window techniques, are capable of taking the roles of both of the former dual screens in one. Today's users are accustomed to working with the screen, and usually do not require separate plotter output. Today's data are stored in databases and not on drawing paper. Thus, digitizers and plotters have lost their importance in the design process.

5.3.1 The Interdependence of Hardware and Software

The effectiveness of a CAD workstation is determined by the quality of the CAD software, the capabilities of the workstation hardware and, last but not least, by the interface between the two, in this order of importance. Today, software has the leading role in the design of CAD systems. Graphics standards define the interface between device-independent software and hardware-specific code. Hardware has had some influence on the definition of the standard interfaces. However, since there have been radical changes in graphics hardware since these first definitions, matching hardware and software is still full of problems.

The typical problems to be solved in the future are the difficulties of eliminating the computational bottlenecks in the software by hardware support, or to make hardware capabilities effective for the software. Problems of this kind are window support, hidden-surface elimination, patch rendering, and many others.

For example, typical window problems arise in software when there is no hardware support for windows. In this case, the pixel generation capabilities of the graphics hardware are no longer useful as they directly put their results into the pixel frame buffer without checking window limits. Consequently, in those cases the task of pixel generation is pulled up to the software and is thus dramatically slowed down. Hardware window support, however, may consist of memory access control in accordance with window borders or it may even allow for several virtual screens being mapped to the physical screen by address handling under window-parameter control, and always being kept in an updated state. It will require severe changes in the actual software to make this hardware accessible.

Hidden-surface elimination can be done in software algorithmically by calculating the appearance of a scene after perspective transformation. This bothersome procedure may in many data cases ($2\frac{1}{2}$D) be replaced by a sorting procedure and entry of the primitives into the pixel store in the order of visibility. A general solution to the hidden-surface problem is the z-buffer technique, which has no limitations concerning the graphical data. To make use of this technique, it must be addressable at the graphics standard interface.

At the moment, the modeling sections of CAD systems use much more complex primitives than the rendering system of graphics machines do. Thus, one step on the way towards visualization is the substitution of complex primitives by simple ones. Spline curves are substituted by polylines, patches are substituted by planar facets. These substitutions are required by the low complexity of the interpolation processors in the pixel generation hardware. It is probably only a matter of time until this situation changes and hardware pixel generators are able to render splines and patches directly. The development of the appropriate hardware will have to take the situation in the modeling area into account.

The three examples above make it evident that software algorithms and strategies, graphics standard specifications, and hardware processors and architectures must not be treated in an isolated manner. Satisfying results will be obtained only if there is tight cooperation between the experts in the three respective fields. Today, the software experts often do not know the full capabilities of their hardware and some do not even know the bottlenecks of their implementations. On the other hand, the hardware experts often do not know the true requirements of the software and some do

not even know the effectiveness of their hardware features. So, though both expert groups do their best in their own fields, this does not always result in the best decisions in a global sense.

Graphics workstations of today are able to calculate and present complex schematic pictures in a very small amount of time. They are also not far from the point of presenting true-to-nature scenes in real time. Since CAD graphics output is typically simple (schematics, wire frames, etc.), the output processing capability of workstations is no longer a severe problem. As a pixel frame format of more than 1000 by 1000 pixels is now standard, the quality of the graphics output is also sufficiently high. Even in those cases where the software does not make use of all the support that graphics hardware offers, the bottleneck should not be attributed to the hardware of the machines.

5.3.2 Graphics Workstation Architecture

Graphics workstations (Fig. 5.23) always contain a powerful general purpose processor (the so-called CPU Section of the workstation) which is typically not severely involved in graphics input/output processing. It is responsible for running the CAD software, while the tasks of graphics I/O are typically done by specially designed hardware (the so-called GRAPHICS Section of the workstation).

Typically, graphical workstations are offered as so-called families. The members of one family may differ in the power of the general purpose processing system and they typically offer different graphics hardware. Members of the same workstation

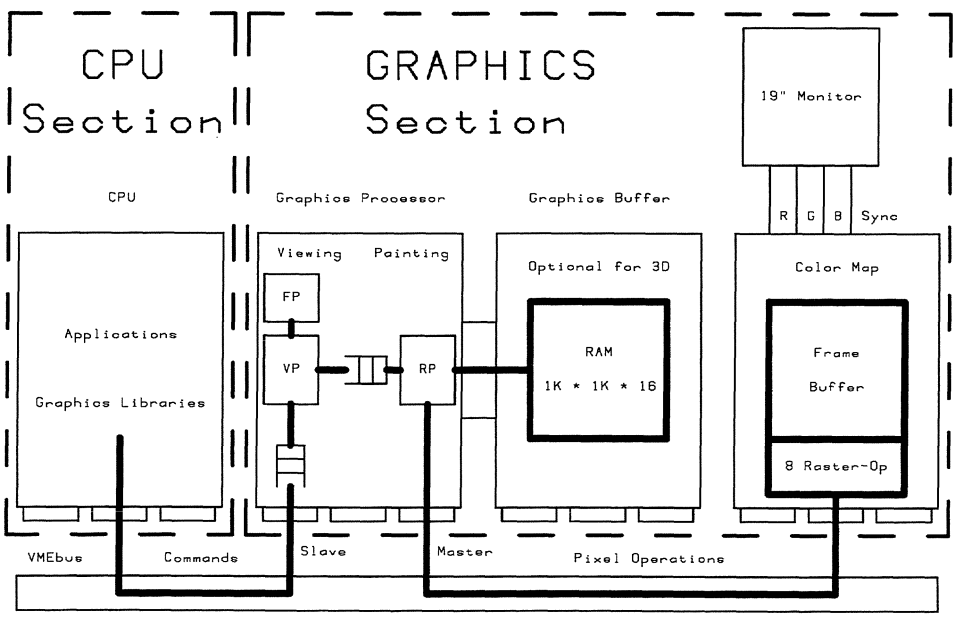


Fig. 5.23. Sections of a graphics workstation
(courtesy of SUN Microsystems, Mountain View, USA)

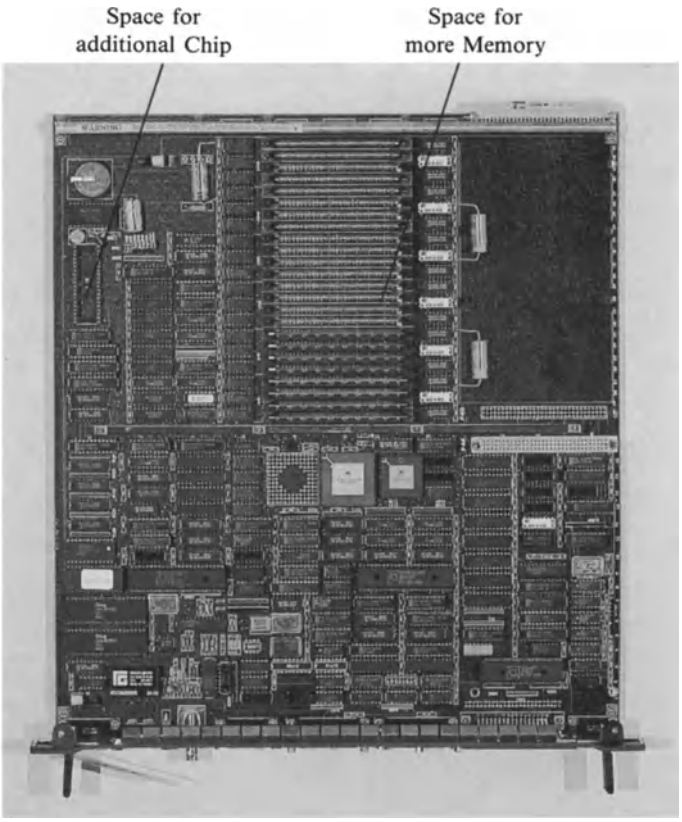


Fig. 5.24. Board of a single-board graphics workstation

family generally follow a conformable basic concept, use the same software, but are different in graphics speed and quality.

Concerning mechanical structure, there are two types of workstations: single-board machines, and bus-oriented machines. Single-board machines are cheaper than bus-oriented machines with the same capabilities. Bus orientation at the same cost always means loss of speed. As single-board machines are naturally the low-end members of a workstation family, they are likely to be sold in large numbers, which brings the price down further. Thus, the price gap between a single-board machine and a bus-oriented machine in the same workstation family is typically rather larger than the performance gap. However, the single-board machines have the disadvantage of very low flexibility for upgrading the hardware. Typically, only chips (e.g., floating point processor) or so-called piggy-packs (e.g., additional main memory) may be added on these boards (Fig. 5.24). The performance of these machines cannot be increased dramatically by upgrades. More flexible, but at the same time more expensive, are the bus-oriented machines. In these workstations, a general purpose bus connects the general purpose processor of the workstation with the special graphics hardware. Both the CPU Section and the GRAPHICS Section can be configured from a set of

boards each. By choosing an appropriate configuration, many different members of the same workstation family can be composed to take a certain place in a wide range of performance. It is practical to select a family in such a way that a medium-equipped member of it offers the required performance.

5.3.3 Personal Computers and Graphics Workstations

Today it is not possible to distinguish clearly between personal computers and workstations. Since microprocessors became fast and able to address huge main memories, since memory became cheap, since cheap, small-sized hard disks with large capacity became available, since high-resolution monitors became inexpensive and, most importantly, since high-quality software began to be sold at a comparably low price, the whole range between $1000 PCs and $100000 workstations is covered by machines capable of running CAD software more or less professionally.

A possible criterion for classifying a workstation is the power of its two sections. The more powerful the workstation is, the more specialized and numerous are the modules found in either of its two sections.

PCs suitable for CAD are frequently of bus-oriented type. The reasons for this decision, however, are not the better conditions for upgrading. The bus orientation, instead, results from the fact that bus-oriented PC boards without graphics capabilities (the former alphanumerical PCs) are available at very low cost, and graphics boards are also (e.g., Hercules, EGA, VGA, etc.). Many PCs are still sold without graphics but may be upgraded easily. The tendency clearly is towards graphics capabilities, pushed by the window-based user interfaces of recent software (e.g., GEM, X-Windows, etc.). In bus-oriented PCs there is at least one board for the CPU Section and another for the GRAPHICS Section. The bus itself is not a bottleneck in these systems. The selection of the bus is therefore mainly dependent on the product line of the manufacturer and not due to the properties of the bus itself. Thus, many different buses coexist in different PCs, the two existing microprocessor bus standards (the so-called MULTIBUS and the so-called VME bus) are found in addition to manufacturer-specific buses (e.g., the quasi-standard buses of IBM). Though quite powerful due to very fast microprocessors, large main memory and effective graphics boards, PCs are clearly the low end in the range of CAD-capable machines. One of the reasons for this fact is the responsibility of the CPU Section for all graphics calculations except for the last pixel-level ones in the calculation pipeline.

It is useful to keep in mind the pipeline of tasks which has to be passed through by the data to obtain the picture on the raster screen. This pipeline may be subdivided into two sections: the coordinate-oriented one and the pixel-oriented one. The pixel-oriented tasks are the calculation of the pixels (determining which pixels are different from the background and how they appear on the screen), their entry into the pixel memory (frame buffer), their read-out for screen refresh and their presentation on the screen. The calculation of the pixels is in a very general sense called rendering. The entry of the pixel values into the pixel memory is typically done pixel-wise and is just data entry. The read-out of the pixels can be done in larger packages (as the sequence of pixels required is regular) and may perform a lot of functions (panning, zooming, window functions, etc.). The presentation on the screen is the generation of the video

signal from the pixel value information and may contain straight-forward functions (color table, antialiasing, etc.).

The highest speed demands are found in the read-out and presentation tasks of the GRAPHICS Section. However, no severe problems arise from these tasks, as the architecture of the data stream is simple and not data sensitive. Special functions generally require special hardware as there is no flexibility left in the straightforward functionality. The bottlenecks of the GRAPHICS Section are found in the task of pixel entry (a problem of memory bandwidth) and especially in the task of rendering. While the task of pixel entry is clearly defined and can be solved by special hardware architectures, the rendering task is a completely open field and requires special discussion.

Rendering means to convert the picture information from the application to pixel presentation on the raster screen. Generally, there is a wide range of formats for defining a picture. First, the geometry may be defined in many ways (vectors, curves, patches, solids). Second, the appearance may be defined differently (color, surface parameters like reflection, diffusion, texture, refractive index, etc., material parameters like transparency index, absorption spectrum, strewing factor, etc.). Third, the light model may contain a lot of information (number, place, and characteristics of light sources) and typically defines compromises for effects like shadows, reflections, and highlights. In CAD, however, the main work is done using abstractions of the reality like wire frames. These types of user model presentation can be done with a quite simple rendering task compared with presentations true to nature. The latter are used in CAD at least for final judgement, and therefore lie outside the interactive process of design. Thus, for interactive CAD the task of rendering is merely restricted to geometrical computation, hidden-surface elimination (which is also required for wire frames), and possibly depth cueing in order to help the user of 3D models.

The first steps of the geometry pipeline are generally done in the CPU Section of the machine. Depending on the facilities in the GRAPHICS Section, the last of these steps are transformations and clipping (if the GRAPHICS Section does not support these operations) or the conversion of application-specific primitives to machine-specific ones (if transformation and clipping processors are integrated in the GRAPHICS Section). The task of primitive conversion is always done by the CPU Section of a workstation as it is application-dependent and therefore does not admit of special treatment in the GRAPHICS Section, which is only able to handle machine-specific primitives. Whether the GRAPHICS Section performs its tasks in software, firmware, or hardware is dependent on the compromise chosen between speed and cost. Changing from software to firmware and from firmware to hardware means speeding up the respective tasks but, on the other hand, loss of flexibility and increased cost.

Today, PCs offer CPU Sections with typically one megabyte of main memory and a processor performing about one million instructions per second. They speed up floating point operations by a special coprocessor and typically have a Winchester hard disk of some tens of megabytes capacity.

In single-board PCs the GRAPHICS Section typically shares the main memory with the CPU (the pixel memory is part of the main memory) and is just responsible for the screen refresh. Rendering of pixel data and entry into the pixel memory is done entirely by the CPU, unless the GRAPHICS Section contains a processor able to enter

primitives into the pixel memory directly, to some certain degree. So-called CRT controller chips have pixel generation and entry capabilities besides their screen refresh activities.

In bus-oriented PCs the GRAPHICS Section is typically a complete subsystem with a microprocessor of its own, a CRT controller, and a separate pixel memory built from video RAMs. Here the availability of the CPU Section for the application program is much better and the graphics output does not slow down the CPU substantially as in the case of the single-board solution. On the other hand, the fixed interface specification of the GRAPHICS Section may require adaptation of the application programs.

Workstations may be looked upon as upgraded PCs today. In their CPU Section, especially low-end workstations frequently make use of the same CPU chip types as PCs do, but typically they run them at a higher clock rate and use the cache technique to allow the memory to keep up with the fast processor. In bus-oriented workstations, special floating point accelerators are available, much faster than the floating point coprocessors which are standard provision. Very powerful graphics workstations use multi-stage instruction pipelining in their CPU Section, and RISC processors. Workstations do not use main memory for pixel buffers. They have a special frame buffer allowing very extensive pixel specification (up to three times 8 bits color plus 16 bits z-buffer) without speed limitations. This frame buffer is part of the GRAPHICS Section.

The GRAPHICS Section of a graphics workstation typically is a high-speed processor (comparable with the CPU processor) assisted by further processors specialized

Fig. 5.25. An example of a graphics workstation

Fig. 5.26. Another sample of a graphics workstation (courtesy of Siemens, Munich/Berlin, Germany)

for certain tasks. Typical tasks appropriate for special processors are transformation, clipping, pixel generation including interpolation for geometry and appearance, and picture processing on a pixel basis. While in the CPU Section there are no hints that conventional architectures will be overtaken soon (it will only be a matter of a few years before, at the same price, CPU Sections will be speed up by a factor of 5), in the GRAPHICS Section parallel processing and extensive pipelining seems to be taking the place of expensive ultra-speed hardware in those cases where a very high complexity-quality-speed product is to be obtained.

The physical appearance of modern graphics workstations is quite uniform (Fig. 5.25, Fig. 5.26). Low-end machines are of desk-top type with a PC-sized cabinet, a 15- to 19-inch screen (black/white, grey-scale, or color), a keyboard, and a mouse. More powerful workstations have a larger cabinet, but still small enough to find its place under the working table. All workstations offer network capability. Plotting and scanning is typically done in the network, as is extensive data storage.

5.4 Graphics in Networks

5.4.1 Introduction

Because of technological advances in computer hardware, it has become advantageous to use a great number of computers and to join them to a system of different

computers. Such a system, consisting of personal computers, workstations, minicomputers, and mainframes, allows decentralized data processing based on networking.

Single computing systems or devices, called nodes in general, are connected via serial buses, building a network. There are the following advantages:

- A node's program and datafiles can be used by other nodes. This can be done by filetransfers, that means by copying one node's files into the directory of another node. Also, a given node (destination) can be operated via another node. Although connected to a remote node, networking gives the user the impression that his terminal is connected to the destination. This technique is called a virtual terminal.
- Rare (and often expensive) resources can be used as shared resources by network nodes, so these resources are used more economically. Such resources are, for example, high performance computers (so-called computing servers), graphic plotters, laser printers, and also database servers.
- Filesaving can be done efficiently via the network by transferring the contents of a computer's secondary memory to the mass memory of a special, highly reliable computer (fileserver) using file transfers. This filesaving can be initiated centrally by any node.
- The network will make the implementation of system-wide fault tolerance possible, if in case of an error redundant resources are available.
- The increase of computing power can be obtained using net-wide computing resources in parallel.

5.4.2 CAD's Requirements on Computer Networks

Today most CAD is based on workstations [SALM87]. Engineering projects are in general performed by a team of engineers. The team concept applied to CAD means the use of multiple workstations within one project; that means decentralization of tasks, so it is necessary to link CAD workstations [EIGN85]:

- It must be possible to access external files from every workstation according to the requirements of tasks. These files must be updated according to the tasks' progress.
- A project communication between project members must take place by messages or via commonly usable databases.
- As CAD systems are typically used in office environments, networking need not exceed the capabilities of local networks (LANs).
- To be able to react to project requirements in an optimal manner, it is necessary to adapt the number of workstations by simply changing the network's installations.
- Today's LANs meet the requirements outlined above.
- Workstations of different manufacturers must be able to form a heterogeneous system. There may be severe problems in building such a system.

5.4.3 Basics of CAD Nets

CAD networks are typically based on LAN general concepts [FRAN86], [HASL87], [KELL86], [GOER85], [SCHI86], [CHYL87]. Local networks have an extent of some

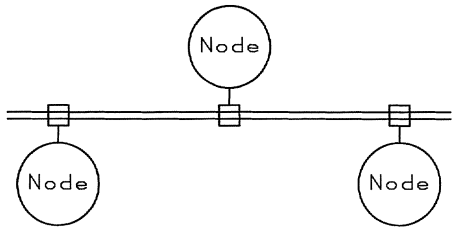


Fig. 5.27. Bus topology

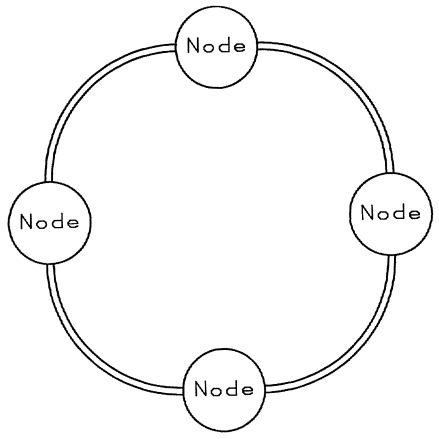


Fig. 5.28. Ring topology

100 to 1000 meters. The number of nodes is unlimited in theory, and can be increased easily (open networks). There is no central switching system, so connections are established decentrally by communications software. This software is based on network controllers represented by the nodes' interface boards, which connect a node to the physical medium by a cable adapter. In CAD systems coaxial cables are mainly used; these allow a transmission rate of about 10 Mbit/s. An increase to 400 Mbit/s is possible by using fiber optic cables (e.g., [KAUF87]). The way single nodes are connected is called network topology, and may be represented by graphs. CAD LAN systems typically form the topologies "bus" and "ring" (Fig. 5.27 and Fig. 5.28).

Bus means to connect all nodes to a single medium (Fig. 5.27). Ethernet [CHYL87] is a typical, commonly used bus-LAN. Its medium consists of a coaxial cable, which allows the connection of nodes to it without interrupting the bus, by using a special tool.

The ring topology means a ring made of coaxial cables (Fig. 5.28). Here the access method "Token Ring" is used. That means the right to access the ring (= Token) is passed from one node to the neighbour node. This method has a great performance. Its disadvantage however is that communication may be interrupted if a node fails, and is interrupted during the installation of additional nodes [SCHI85].

In general, computer networks use bit-serial transfers [FAER87]; several bits are united into a data word (e.g., an octet), several data words form a block, and several blocks form a packet called a datagram. Such a packet is the smallest transferred unit. One task of communications software is to keep the time context of all datagrams.

For this purpose, each datagram contains appropriate information. As already shown in Sect. 4.3.4, data communication can be described by using the ISO-7-layer-model. Communication means communication between the same layers of this model. Each layer has interfaces to its neighbouring layers. – Actually, today's manufacturers of communications systems base their systems only approximately on this model. The tasks of the single layers are briefly described in the context of LANs as follows:

- Layer 1 = Physical layer
  - transfer of unstructured bit-streams by transmission codes that ensure synchronization (e.g., Manchester code with coaxial cable Ethernet, NRZ with fiberoptic networks);
  - Modulation/Demodulation: in general not necessary in LANs, as baseband is sufficient for transmission;
  - Connection of nodes to transfer medium by appropriate hardware modules (typical standards, e.g.: RS 232, special transceivers for Ethernet connection).
- Layer 2 = Link Layer
  - access method;
  - integration of unstructured word streams to blocks (frames);
  - guaranteeing sufficiently safe transmission of each block by methods of fault detection and correction. (Typical LAN standard protocol: HDLC, [SCHI85].)

  Remarks on access methods:

  As only one node is allowed to access the transmission medium at a time, access conflicts caused by simultaneous access by more than one node must be solved by an access method. One appropriate method is the already mentioned token ring method, where the token defines the node that will work next as a transmitter. This method is a deterministic one, and is usable in real time processing [CHYL87], [I3EB85].

  CSMA-CD (Carrier-Sense Multiple-Access Collision Detection) [I3EA85] is a non-deterministic method. Any node is allowed to access the bus only when it does not recognize the carrier (or a signal, which takes the place of the carrier if baseband is used). It interrupts its data transmission and transmits a jam-signal to all nodes when it recognizes any disturbance of its signal on the bus while transmitting. After a time, defined by a random-number generator, it repeats its transmission. As any node involved in a collision tries another transmission in the described manner after a individual random time, most probably one node will be the first and will occupy the bus before the second node will check the state of the carrier. This access method needs no modification in case of system extension. Bus request time, however, increases with the bus load, because the number of bus collisions increases. CSMA-CD has been developed in the context of the Ethernet.

Above these layers today's protocol software is offered, strongly dependent on its manufacturer.

- Layer 3 = Network Layer
  - Fragmentation and defragmentation of datagrams' blocks;
  - Establishing of connections, connecting several LANs.
- Layer 4 = Transport Layer

  - Reconstruction of some datagrams' context;
  - Addressing of certain user processes on end nodes;
  - error treatment by repeating transmission of datagrams.
- Layer 5 = Session Layer
  - Establishing monitoring, closing of connections during a user's session;
  - Conversion between logical names and network addresses.
- Layer 6 = Presentation Layer
  - Converting data intended to be transmitted to a system-neutral format in order to be compatible with different systems.
- Layer 7 = Application Layer
  - Synchronization of application, calling network services;
  - Net access monitoring.

Some widely distributed protocol families:

- TCP/IP covers layers 3 and 4.
  TCP (Transmission Control Protocol) addresses communication partners definitely, and reconstructs the correct sequence of datagrams.
  IP (Internet Protocol) transmits datagrams between several different networks.
- FTP (File Transfer Program) covers layers 5, 6, and 7 and is based on TCP/IP, performing a filetransfer between two different systems.
- TELNET covers layers 5, 6, and 7 and is also based on TCP/IP. It builds a virtual terminal.
- NFS[1] (Network File System) covers layers 5, 6, and 7 and is also based on TCP/IP. It makes possible network-wide directory-oriented access to different nodes' files, network-wide remote procedure calls, and translations of file formats.

5.4.4 Decentral Computing Centers/Graphic Computing Centers

Low-cost computing hardware and local networks have propagated decentral data processing, also in CAD. This kind of data processing, however, requires system-wide management to guarantee computing operation. Such management is commonly the task of a computing center. Decentral data processing results in a distributed system; administration of system components, however, must be managed by a centrally working facility which is not in contradiction to a decentral computing center [REMM85].

Graphic data processing, especially in CAD, requires such facility as well. Here, this facility is called a graphic computing center, because it specifically takes the needs of graphic data processing into consideration. Its tasks are mainly to maintain:

- resource administration, that contains an information system having regard to system configuration according to the particular properties of CAD, and available to all users;
- a project-related, network-wide reservation system;
- system-wide process and data saving;
- a maintenance system taking into account the special needs of graphic input/output devices.

[1] Trademark of SUN Microsystems, Inc.

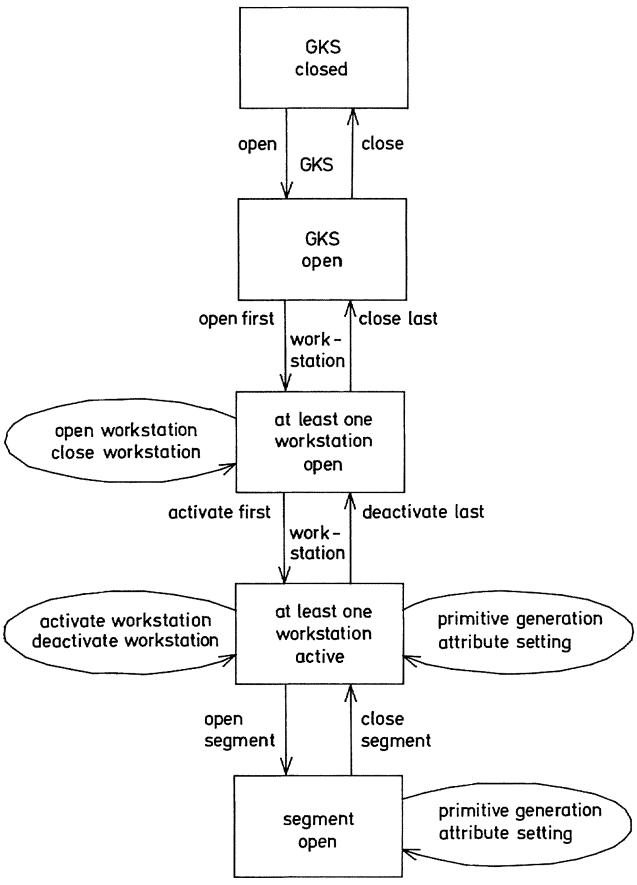


Fig. 5.29. Operating states and state transitions in GKS

5.5 The Graphical Kernel System

5.5.1 System Description

The international standard GKS has already been described very briefly in Sect. 2.3.2. As it is a representative example of the graphics software many system programmers and nearly all application programmers employ to make use of graphics hardware, additional details, and especially examples, are given here.

GKS is structured according to a strictly defined set of operating states. In each of these states, only certain actions are allowed (Fig. 5.29).

Our description of GKS is based on [ISOA85], and we take examples from [STRA81]. In particular, we will discuss some essential design decisions, which are typical for the process of developing a full functional specification of a software tool.

GKS is also an example of the overwhelming amount of effort that has to be put into a specification when one really labors to achieve completeness, consistency, orthogonality, and other worthwhile goals. In the development of CAD systems, a comparable amount of time and man power can be allotted to the specification task only in very rare cases. In general, CAD system development will have to cope with specifications that are much less complete, partially inconsistent, or otherwise flawed.

From the start of the standardization activities in 1974, the main objective was to allow easy portability of graphics systems between different installations. Although GKS was to be capable of being used in small stand-alone graphics programs, it was also essential that large suites of CAD programs could be written and moved from one installation to another, possibly with quite different hardware, without the need for significant reprogramming that might involve changes not only to the syntax, but also to the program structure.

One of the major problems of GKS design was that the characteristics of graphics hardware exhibit significant differences, and will continue to do so in the future. It is obviously difficult to represent a wide range of facilities by a single abstraction which would satisfactorily approximate a flat bed plotter, a high-performance vector display, a plasma panel, and a color raster display. It was felt that the provision of implementation-dependent defaults for unavailable facilities (say, color on a liquid crystal display) would not always be appropriate. There was a need for the application programmer to have some control over the mapping of graphics primitives to a particular device. Hence, a rather sophisticated parametric abstraction was conceived: the workstation concept. A graphics workstation is equipped with a single display area and a number of input devices. It may have a certain amount of intelligence, either locally, connected to the display itself, or in the form of a workstation driver running under GKS control as part of the application program. A workstation is described as belonging to a standard type (one of a set of types) available at a given installation (plotter, storage screen, refresh display, etc.). The application programmer has the ability to modify its overall behavior so as to make optimal use of its features in that environment.

An operator can have a number of GKS workstations under his control at the same time. For example, he may output a large CAD drawing on a plotter while getting a quick-look view on a separate raster screen. Or he may be interactively changing his model on a refresh display, while making occasional hardcopies on a plotter.

The application programmer has considerable flexibility in how he uses each workstation. Different workstations may be set to view different parts of the whole graphics picture. The frequency of update may be different on different devices.

The CAD application programmer may insert statements into his program inquiring which capabilities are implemented in the graphics hardware of its run-time environment. He may design and implement the program so that it will adjust its behavior to the available facilities. For example, with a refresh display one program will show any modification of the display information immediately, while with a storage tube, it will collect a number of changes until the whole picture is redrawn. The lack of a tablet may necessitate a different method of entering locator positions. The type of echoing may depend on the line speed between display and computer. Such adaptive behavior may be programmed into the CAD system to improve its portability in terms of the range of graphics hardware installations on which it can run.

Besides the workstation concept, the treatment of the visual appearance of graphical information on the display surface is peculiar to GKS. Graphics primitives such as lines can have attributes associated with them, such as color, thickness, linestyle, etc. The approach chosen for GKS was to provide one major attribute per primitive, called the primitive index; it is a number between 1 and some implementation maximum. A GKS program like

```
SET POLYLINE INDEX(1)
POLYLINE
SET POLYLINE INDEX(2)
POLYLINE
```

also requires a definition of how indices 1 and 2 should be represented on one device or another. Upon display, the indices are used to look up corresponding entries in a table associated with the workstation, and the table defines a whole "bundle" of appearance attributes for each bundle index. These entries may be preset by calls from the application program to GKS. They are modal attributes and thus remain in effect until they are redefined. They are associated with one workstation each and not with GKS as a whole. The application programmer may set the representation of index 1 as red, thick, and solid, while index 2 could mean green, thick, and dashed. The advantage of making the pen specification workstation-dependent is that the representation may be quite different on another workstation. For example, a designer using a flat bed plotter for his final drawing output and a plasma panel during his interactive work could recognize different colors as indicating different linestyles on his display (red = solid, green = dashed, for instance) – provided that the application programmer has decided to distinguish the different bundle attributes in this way.

The elementary graphical items treated in GKS are called primitives. GKS has defined six output primitives (see Sect. 2.3.2):

- POLYLINE,
- POLYMARKER,
- FILL AREA,
- TEXT,
- CELL ARRAY, and
- GENERALIZED DRAWING PRIMITIVE (GDP).

For line drawing, a polyline – which generates a sequence of lines connecting an ordered set of points (given as a parameter) – is the fundamental line drawing primitive. The motivation for having a polyline as a primitive, rather than a single line, is that in most applications a set of lines is needed to form some shape. Since a polyline rather than a line is the basic primitive, attributes such as linestyle apply to the complete polyline instead of a single line segment. Thus, a red dotted or green dashed curve is drawn as a single entity.

The polymarker is an obvious primitive, once polyline has been defined. Text similarly produces a string of characters, rather than a single character, so that there is some similarity of level among the three main output primitives.

The three remaining primitives demonstrate the spreading influence of raster graphics and the need to allow access to the hardware features of certain output devices. For example, raster displays support cell arrays on the hardware level. Conse-

quently, this primitive has been made available in GKS in a generalized way. The GKS primitive "fill area" defines a boundary whose interior can be filled in some pattern and color, – or simply hatched if the hardware can only produce lines. Some plotters provide such features as circles, arcs, or interpolation curves. The "generalized drawing primitive" (GDP) provides a standard way of addressing such non-standard capabilities (circles, arcs, etc.).

Polyline and polymarker have a single attribute, the index; at a given workstation this index determines a bundle containing

for POLYLINE: LINETYPE, LINEWIDTH SCALE FACTOR, COLOR,

and for POLYMARKER: MARKERTYPE, MARKERSIZE SCALE FACTOR, COLOR.

Text, on the other hand, has two sets of attributes, some of which are set by the text bundle table:

TEXT: FONT, PRECISION, COLOR

while the remaining attributes are set modally on GKS level; that is, they are bound directly to a primitive when it is generated. The motivation for this split is that the overall form and shape of the text is considered more as a geometric property which should remain invariant on whatever output device is used, while the appearance (form and quality) of the individual characters may vary.

The geometric text attributes are:

- HEIGHT: defines the required height of the character in the user's coordinate system.
- EXPANSION FACTOR: defines the actual width on the basis of the nominal width expressed in the height/width ratio of a given font.
- CHARUP VECTOR: defines the degree of rotation for every character in the text line. Characters can therefore be drawn at any orientation.
- PATH: defines the direction in which characters are drawn. The normal setting is RIGHT while LEFT draws them from right to left. Similarly, UP and DOWN have obvious meanings.
- SPACING: controls the amount of space between characters, beyond that nominally provided by the font.

It is recognized that some devices may have difficulty specifying characters with this degree of sophistication. Consequently, the PRECISION attribute (with values STRING, CHARACTER, STROKE) in the text bundle table defines the closeness of the output to the specified requirements:

- STRING: the position of the first character is all that is guaranteed to be correct. Thus, a device's hardware character generator can be used. If a different orientation or size is requested, it can be ignored.
- CHARACTER: the position of each individual character box must be correct. The form of the character within each box is workstation-dependent. Again, hardware characters could be used but, in that case, they would probably have to be output one at a time.
- STROKE: all the text attributes have to be implemented correctly. This will almost certainly require the hardware to have a very flexible character generator, or else

the text output must be simulated in software using polylines or fill area primitives.

This method of defining text in GKS permits the use of sophisticated hardware character generators if available. Otherwise, the high-precision text has to be produced in software. In order to avoid the processing required for such a software simulation, the application programmer may choose a simple and cheap text representation of precision STRING or CHARACTER for all but the final drawing.

Similar to text, the fill area primitive also has two sets of attributes: the first one regarded as part of the geometry, independent of the workstation, the other one selected by index and defined in a workstation table. The geometric properties are relevant when the interior is filled with a certain pattern:

- PATTERN SIZE: defines what size will be assigned to the pattern. The pattern is replicated in both x and y directions, until the complete area is covered.
- PATTERN REFERENCE POINT: specifies the origin for the replicating process.

These attributes are mainly oriented towards raster devices, but simulations for vector devices are also defined. The following attributes are associated with each workstation individually:

- INTERIOR STYLE: defines the mode of filling as hollow (not filled), solid, patterned, or hatched.
- STYLE INDEX: specifies for pattern an entry in a pattern table, to be used for filling. If the interior style is hatched, the index is used to determine which of a number of predefined hatch styles is used.
- COLOR INDEX: is used for both hollow and solid, and refers to the color table.

The next essential topic to discuss is the segment facility in GKS. A single graphical primitive will not generally correspond to an information chunk in the communication between the designer and the CAD system. GKS provides the means to group a number of primitives together into a meaningful information package called a segment, which may be handled as a whole. Segments can be individually deleted as a unit, or highlighted to stand out from other information. They may be moved around, scaled, or rotated. This is achieved by having a transformation matrix associated with each segment, which may be altered after the segment is defined.

On workstations which provide storage capability of their own, segments may be stored and made visible or invisible by appropriate requests from the application program to GKS. Sometimes the need arises to display a segment on a workstation that was not active when the segment was created. A typical example is a plotter which is not activated until a complete and correct picture has been composed on an interactive display. For such purposes, GKS provides a device-independent segment storage which can keep copies of segments as they are formed. All transformations performed on the segment will also apply to its device-independent copy. From the device-independent storage, the segments can be copied to other workstations. Facilities are also provided for inserting a segment into another segment. The difference is that a copy operation – as the name suggests – produces a duplicate of the segment, while insertion copies all the primitives contained in the segment into a newly created segment. These facilities are supported only in higher levels of GKS implementations.

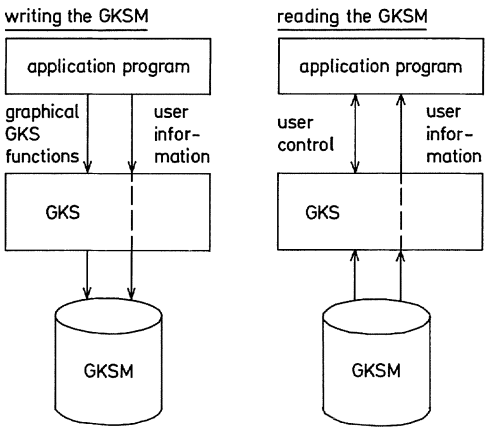


Fig. 5.30. Communication between an applications program and the GKS Metafile

For archiving graphical information, and for transport to and from other installations or systems, GKS provides the GKS metafile facility (GKSM, see also Sect. 2.3.4). The application program may direct graphical output to a GKS metafile workstation, which will write it onto a sequential file. The application program can also transmit its own records (containing non-graphical information) to the metafile via GKS. In a subsequent job, the same or another application program may have GKS read and interpret the metafile, effectively regenerating whatever graphics information is on the file. User records may either be ignored or passed to the application program for further interpretation (Fig. 5.30).

Obviously, the segmentation decision made for GKS is by no means the only one possible. One could envisage systems that treat individual primitives as entire units for handling purposes. One might also consider higher levels of structuring, such as segments containing references to other segments. The decision made for GKS in this respect – as in all others – is a compromise, based on the maximum international agreement that could be achieved as to what was required by the application community and could be provided without overloading the implementation with requirements that might be shifted to another layer in the application program.

A common need of interactive graphics users is the ability to switch between overall views of their model and some kind of detail view. The window/viewport concept involves a standard computer graphics technique of mapping some rectangle of the application space (called the window) onto a rectangle of the display surface (the viewport). In GKS, where several workstations may be active at the same time, it makes sense to allow different views of the same picture to be seen on different workstations. This flexibility is achieved by having three different coordinate systems and two distinct window/viewport mappings (see Sect. 2.3.1). The applications programmer defines his output in terms of a world coordinate (WC) system, which is mapped onto some part of the normalized device coordinate (NDC) plane. The active workstations can then present separate views of the NDC space when the application program chooses a separate mapping from that space to the device coordinates for each workstation.

Multiple windows are a useful facility in GKS. It is quite common for the application programmer to want to display several distinct parts which are most appropriately defined in different coordinate systems. A conventional way to do this would be to redefine the window each time as required. For example:

```
SET WINDOW(XMIN,XMAX,YMIN,YMAX)
DRAW PICTURE A
SET WINDOW(X2MIN,X2MAX,Y2MIN,Y2MAX)
DRAW PICTURE B
```

Here PICTURE A is drawn when the first coordinate system is defined, while PICTURE B is drawn with the second coordinate system. The user effectively sees a display made up if two parts with different coordinate systems. The application program would have to memorize the corresponding windows associated with pictures A and B, and reset them whenever required. However, on input the problem becomes more severe, since both pictures coexist on the same surface and it is not immediately obvious whether a point on the surface refers to the space of picture A or B. Hence, it is not possible for GKS to transform the coordinates of a point on the device back into the user world in a unique manner. The problem has been solved by giving the different world coordinate systems distinct names (in fact, numbers), and by making them known to the GKS process. Thus, the application program and GKS can inform each other which mapping is to be applied in each instance. The equivalent form of the above program in GKS would look like:

```
DEF WINDOW (1,XMIN,XMAX,YMIN,YMAX)
DEF WINDOW (2,X2MIN,X2MAX,Y2MIN,Y2MAX)
SELECT (1)
DRAW PICTURE A
SELECT (2)
DRAW PICTURE B
```

Note that due to this technique, the application programmer will have a tendency to define all the potentially needed coordinate systems collectively at the start of execution and then select a particular transformation whenever required. The more conventional technique would have transformation definitions scattered throughout the program. This is another example of how strongly the layout of a software machine influences the structure of the programs that are built on top of it.

Input in GKS is defined in terms of a set of logical devices which may be implemented on a workstation in a number of ways. The different types of input are:

- LOCATOR: provides a position in world coordinates. The position indicated on the display will be within one of the window/viewport transformations defined. This will be used to give the correct world coordinate position.
- VALUATOR: provides a real number.
- CHOICE: provides an integer defining one of a set of possible choices.
- PICK: provides a segment name and a pick identifier associated with a particular primitive.
- STRING: provides a character string.

The implementation of the logical device on a workstation may be done in a variety of ways. For example, while it may be natural to input a STRING using a keyboard, it could also be done by free-hand drawing on a tablet, or by hitting a set of light buttons indicating particular characters on a display. The exact form of the implementation depends on the individual workstation in terms of hardware and software.

Input can be obtained in three distinct ways:

- REQUEST: this is like a FORTRAN READ. The system waits until the input event has taken place, and then returns the appropriate value. Only one input request is valid at a time.
- SAMPLE: the current value of a GKS input device is examined. This is most frequently used for devices which provide a continuous read-out of their value. For example, the current position of a potentiometer or of the stylus on a digitizer can be sampled.
- EVENT: this mode is used for devices which would normally cause interrupts on the workstation. For example, a lightpen hit or a touch of the tip switch on a tablet would normally generate an event. Upon occurrence of the interrupt, a record containing the input information and indicating its source is stored in a queue. The queue is ordered according to the time when the interrupt occurred. Functions are available to retrieve these records from the list for interpretation by the application program.

Earlier versions of GKS had a much more complex input system, with non-sequential listing of input events. It was decided that such functions should be built on top of GKS, rather than being a part of the kernel system.

5.5.2 GKS Examples

In this section, some important features of GKS are demonstrated using sample programs taken from [STRA81].

In the first example (Sample Listing 5.1), we demonstrate the window/viewport transformation and the associated clipping behavior. The result is shown in Fig. 5.31.

The second example (Sample Listing 5.2) demonstrates the transform segment function when moving the hands of a clock. Depending on the particular device capabilities connected to GKS, different actions are performed within the system so that the visual effect will remain nearly the same. Either the segment transformation is performed by hardware functions of the device, or a device-dependent segment mechanism is used that allows for the deletion of a selected segment and a redisplay of just the transformed segment, or the display surface has to be erased completely and all segments have to be redrawn. In the third case, no real-time movement of the second hand is normally possible. The result is shown in Fig. 5.32.

The third example (Sample Listing 5.3) is a simple example of using text and pen attributes in a static way; that is, each time a new pen and text representation is set, the subsequent output is drawn using the new attributes, without changing output already created. Within this program, a circle and eight surrounding letters are drawn. Application-defined pens are used to generate lines. Characters of different color are drawn at different angles around a circle. Precision "STROKE" is defined for all characters. The result of this GKS example is shown in Fig. 5.33.

Sample Listing 5.1. The GKS program for Fig. 5.31.

```
C WINDOW/VIEWPORT TRANSFORMATION
      REAL      WLINEX(5), WLINEY(5), BX(5), BY(5)
      REAL      TRIAX(4), TRIAY(4)
      INTEGER   NOCLIP/0/,CLIP/1/
      DATA      BX     /  0.,   1.,   1.,   0., 0./
      DATA      BY     /  0.,   0.,   1.,   1., 0./
      DATA      TRIAX  / -7.,   2.,  14., -7./
      DATA      TRIAY  / -7.,  11., -7., -7./
      DATA      WLINEX / -10., -10.,  10.,  10., -10./
      DATA      WLINEY / -10.,  10.,  10., -10., -10./
C OPEN GKS (ERROR FILE IS ON DEVICE 22)
      CALL GOPKS (22)
      CALL GOPWK (1,1,6)
C DEFINE NORMALIZATION TRANSFORMATION
      CALL GSW (1,-10., 10.,-10., 10.)
      CALL GSW (2,-10., 10.,-10., 10.)
      CALL GSW (3,-10., 10.,-10., 10.)
      CALL GSVW (1,0.05,0.45,0.55,0.95)
      CALL GSVW (2,0.5 ,0.9 ,0.55,0.95)
      CALL GSVW (3,0.55,0.95,0.05,0.45)
      CALL GACWK (1)
C SET AND DRAW WINDOW BOUNDARIES
C IT SHOWS THE BOUNDARIES OF THE (0,1)-NDC COORDINATE SPACE
      CALL GSELNT (0)
      CALL GPL (5,BX,BY)
C SELECT MARKER TYPE 3
      CALL GSPMI (3)
C 1. IMAGE
C SET NEW VIEWPORT AND DRAW THE WINDOW BOUNDARIES
      CALL GSELNT (1)
      CALL GPL (5,WLINEX,WLINEY)
C PICTURE NUMBER IN THE UPPER LEFT CORNER
      CALL GTX (-8.,8.,1,1H1)
C FIRST IMAGE IS TO BE CLIPPED AT THE WINDOW
      CALL GSCLIN (CLIP)
C POLYMARKER AT TRIANGLE CORNER POINTS
      CALL GPM (3,TRIAX,TRIAY)
C POLYLINE (TRIANGLE)
      CALL GPL (4,TRIAX,TRIAY)
C 2. IMAGE
C SET NEW VIEWPORT AND DRAW WINDOW BOUNDARIES
      CALL GSELNT (2)
      CALL GPL (5,WLINEX,WLINEY)
      CALL GTX (-8.,8.,1,1H2)
C SECOND IMAGE IS NOT CLIPPED AT THE WINDOW
      CALL GSCLIN (NOCLIP)
C POLYMARKER AT TRIANGLE CORNER POINTS
      CALL GPM (3,TRIAX,TRIAY)
C POLYLINE (TRIANGLE)
      CALL GPL (4,TRIAX,TRIAY)
```

Sample Listing 5.1 (continued)

```
C 3. IMAGE
C SET NEW VIEWPORT AND DRAW THE LIMITATION OF THE WINDOW
      CALL GSELNT (3)
      CALL GPL (5,WLINEX,WLINEY)
      CALL GTX (-8.,8.,1,1H3)
C THIRD IMAGE IS NOT CLIPPED AT THE WINDOW (BUT AT THE NDC-SPACE)
      CALL GSCLIN (NOCLIP)
      CALL GPM (3,TRIAX,TRIAY)
      CALL GPL (4,TRIAX,TRIAY)
      CALL GDAWK (1)
      CALL GCLWK (1)
      CALL GCLKS
```

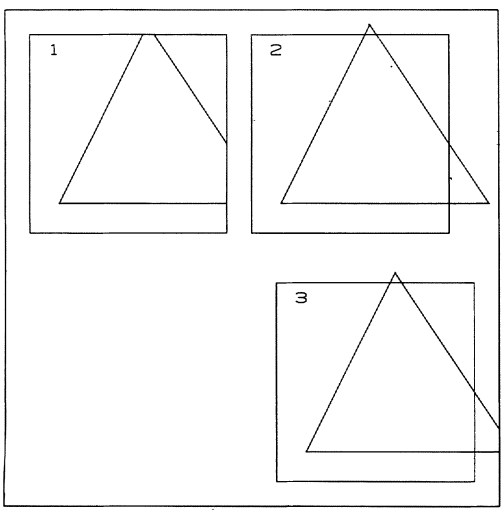


Fig. 5.31. GKS example 1 output

Sample Listing 5.2. The GKS program for Fig. 5.32.

```
C     REAL TIME CLOCK
        REAL        X(3),Y(3),PHI,MHX(6),MHY(6),SHX(5),SHY(5),HHX(10)
      1             ,HHY(10),MH(6),MM(6),MS(6),PI
        INTEGER     I,DIGIT(24),J,K,ICH
        DATA        DIGIT /24H3 2 1 1211109 8 7 6 5 4 /,PI/3.14159/
C     DEFINITION OF THE HANDS
        DATA        SHX/ -0.02 ,  0.  , -0.02,  0.  , -0.02/
        DATA        SHY/  0.  , -0.04,  0.  ,  0.7,  -0.  /
        DATA        MHX/ -0.06, -0.02,  0.02,  0.06,  0.  , -0.06/
        DATA        MHY/  0.  , -0.1 , -0.1 ,  0.  ,  0.65,  0.  /
        DATA        HHX/ -0.015, -0.1, -0.02,  0.02,  0.1,  0.015, 0.1,0. , -0.1, -0.015/
        DATA        HHY/  0.  , -0.1, -0.2 , -0.2 , -0.1,  0.  , 0.2,0.5,  0.2,  0.  /
        DATA        MH/1.,0.,0.,0.,1.,0./,MM/1.,0.,0.,0.,1.,0./
        DATA        MS/1.,0.,0.,0.,1.,0./
        INTEGER WORLDC/0/
C       INTEGER EMPTI/??????/ IS IMPLEMENTATION DEPENDENT
        CALL GOPKS (22)
```

Sample Listing 5.2 (continued)

```
      CALL GOPWK (1,2,6)
      CALL GACWK (1)
      CALL GSW (1, -1., 1., -1., 1.)
      CALL GSELNT (1)
      CALL GCRSG (10)
C     DRAW FRAME AND HANDS AXES BY THREE CIRCLES
      X(1) = 0.
      Y(1) = 0.
      X(2) = 1.
      Y(2) = 0.
      CALL GGDP (2,X,Y,1,1,EMPTI)
      X(2) = 0.95
      CALL GGDP (2,X,Y,1,1,EMPTI)
      X(2) = 0.01
      CALL GGDP (2,X,Y,1,1,EMPTI)
C     DRAW MARKERS AND DIGITS
C     SET CHARACTER HEIGHT
      CALL GSCHH (0.08)
      CALL GSCHSP (-0.1)
      DO 50 I=1,48,4
      X(1) = COS ((I-1)*PI/24.)*0.9
      Y(1) = SIN  ((I-1)*PI/24.)*0.9
      X(2) = X(1) * 0.9
      Y(2) = Y(1) * 0.9
      CALL GPL (2,X,Y)
      X(3) = X(2) * 0.8
      Y(3) = Y(2) * 0.8 - 0.05
      IF (I.GT.9.AND.I.LT.22) X(3) = X(3) - 0.07
      ICH = (I+3)/2-1
      CALL GTX (X(3),Y(3),2,DIGIT(ICH))
   50 CONTINUE
      CALL GCLSG
C     DRAW HANDS
      CALL GCRSG (1)
      CALL GPL (5,SHX,SHY)
      CALL GCLSG
      CALL GCRSG (2)
      CALL GPL (6,MHX,MHY)
      CALL GCLSG
      CALL GCRSG (3)
      CALL GPL (10,HHX,HHY)
      CALL GCLSG
C     MOVE HANDS FOR 10000 MINUTES
C     FOR TERMINATION BY AN OPERATOR SAMPLE OR EVENT INPUT IS
      NEEDED
      PHI = -PI/30.
      DO 200 J=1,10000
      DO 100 K=1,60
      CALL GACTM (MS,0.,0.,0.,0.,PHI,1.,1.,WORLDC,MS)
      CALL GSSGT (1,MS)
```

Sample Listing 5.2 (continued)

```
C       DELAY FOR A VECTOR DEVICE WITH SEGMENTATION FACILITY BUT WITH-
        OUT HARDWARE TRANSFORMATION
        CALL DELAY (985)
  100 CONTINUE
        CALL GACTM (MM,0.,0.,0.,0.,PHI    ,1.,1.,WORLDC,MM)
        CALL GACTM (MH,0.,0.,0.,0.,PHI/12.,1.,1.,WORLDC,MH)
        CALL GSSGT (2,MM)
        CALL GSSGT (3,MH)
        CALL GUPDWK (1,1)
  200 CONTINUE
        CALL GDAWK (1)
        CALL GCLWK (1)
        CALL GCLKS
```

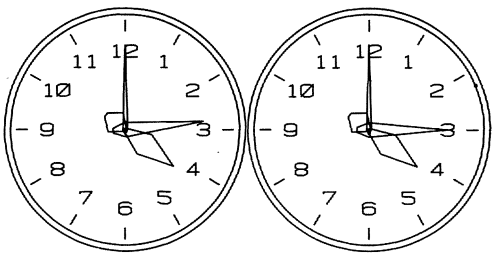


Fig. 5.32. GKS example 2 output

Sample Listing 5.3. The GKS program for Fig. 5.26.

```
C     STATIC PRIMITIVE ATTRIBUTES
        INTEGER STRIN(8), RIND, GIND, BIND, BLAIND, WK
        INTEGER SOLI, DASHED, DOTTED, DASHDT
        INTEGER STROKE/2/
        REAL PX(2),PY(2), PX1, PY1, CUPX,CUPY, IUP
        REAL ANGLE, PI, RADIUS, CIRCX(2), CIRCY(2)
        DATA RADIUS/30./,CIRCX, CIRCY /50.,50.,80./
        DATA RIND/1/,GIND/2/,BIND/3/,BLAIND/4/,WK/1/
        DATA PI /3.1415927/
        DATA IUP /3.5/
        DATA STRIN/1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH/
        DATA SOLI/1/,DASHED/2/,DOTTED/3/,DASHDT/4/
C       INTEGER EMPTI/??????/ IS IMPLEMENTATION DEPENDENT
        CALL GOPKS (22)
        CALL GOPWK (WK,1,3)
        CALL GACWK (WK)
        CALL GSW (1,0.,100.,0.,100.)
        CALL GSELNT (1)
C     DEFINE PENS
C     PEN NO 1, LINETYPE = SOLID, COLOR = RED
        CALL GSPLR (WK,1,SOLI,1.,RIND)
        CALL GSTXR (WK,1,1,STROKE,1.,0.,RIND)
C     PEN NO 2, LINETYPE = DASHED, COLOR = GREEN
        CALL GSPLR (WK,2,DASHED,1.,GIND)
        CALL GSTXR (WK,2,1,STROKE,1.,0.,GIND)
```

Sample Listing 5.3 (continued)

```
C     PEN NO 3, LINETYPE = DOTTED, COLOR = BLUE
        CALL GSPLR (WK,3,DOTTED,1.,BIND)
        CALL GSTXR (WK,3,1,STROKE,1.,0.,BIND)
C     PEN NO 4, LINETYPE = DASHDOTTED, COLOR = BLACK
        CALL GSPLR (WK,4,DASHDT,1.,BLAIND)
        CALL GSTXR (WK,4,1,STROKE,1.,0.,BLAIND)
C     DRAW A CIRCLE USING PEN 1 (SOLID LINETYPE, RED)
        CALL GGDP (2,CIRCX,CIRCY,1,1,EMPTI)
        ANGLE = PI/2.
        PX(1) = CIRCX(1)
        PX(2) = CIRCY(1)
C     SET CHARACTER HEIGHT
        CALL GSCHH (IUP)
        DO 300 I = 1,8
        CUPX = SIN  (ANGLE)
        CUPY = COS (ANGLE)
        CALL GSCHUP (CUPX,CUPY)
C     USER-DEFINED PENS 1...4
        CALL GSPLI (MOD(I-1,4)+1)
        CALL GSTXI (MOD(I-1,4)+1)
        PX1 = CIRCX(1) + (RADIUS + 4.) * CUPX
        PY1 = CIRCY(1) + (RADIUS + 4.) * CUPY
        CALL GTX (PX1,PY1,1,STRIN(I))
        PX(2) = PX(1) + RADIUS * CUPX
        PY(2) = PY(1) + RADIUS * CUPY
        CALL GPL (2,PX,PY)
        ANGLE = ANGLE - PI/4.
  300 CONTINUE
        CALL GUPDWK (WK,1)
C     SHORT VERSION FOR DEACTIVATE AND CLOSE ALL WORKSTATIONS AND GKS
        CALL GECLKS
```

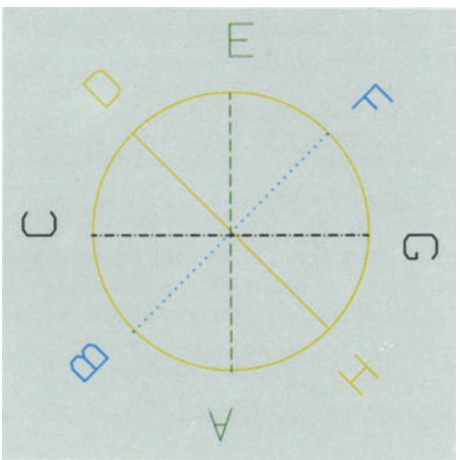


Fig. 5.33. GKS example 3 output

5.6 Summary

This chapter described mainly the hardware used in computer graphics and gave an example of a standard interface to the software (GKS).

Section 5.2 introduced to the most commonly used devices for interactive work and for hardcopy and documentation. The functionality of the devices was outlined in order to give the CAD programmer a feeling for the capabilities available in hardware.

Section 5.3 centered on workstations as today's combination of a powerful computer with graphical devices and special support for them to get a high degree of interaction. The appearance and especially the typical architectures were described.

Section 5.4 outlined some basic information which is required to understand the role of network architectures in CAD (primarily LANs). This was an attempt to introduce the most important terminology to the CAD programmer.

Section 5.5 gave an example in order to illustrate how the large variety of hardware can be handled by standard software. GKS was chosen as it is an existing international standard and shows the problems and the solutions in a very clear manner.

5.7 Bibliography

[ANDE83] R. Anderl, J. Rix, H. Wetzel: GKS im Anwendungsbereich CAD. Informatik Spektrum 6, 2 (1983), pp. 76–81.

[ARDE86] M. Arden, A. Bechtolsheim: SUN-3 Architecture – A SUN Technical Report. SUN Microsystems, Inc. (1986).

[BECH86] J. Bechlars, R. Buhtz: GKS in der Praxis. Springer-Verlag, Berlin (1986).

[BONO87] P. Bono, I. Herman (eds.): GKS Theory and Practice. Springer-Verlag, Berlin (1987).

[CHYL87] P. Chylla, H.-G. Heyering: Ethernet-LANs, Planung, Realisierung und Netzmanagement. Datacom (1987).

[EIGN85] M. Eigner, H. Meier: Einstieg in CAD, Lehrbuch für CAD-Anwender. Carl Hanser Verlag, München (1985).

[ENDE83] G. Enderle, K. Kansy, G. Pfaff, F.-J. Prester: Die Funktionen des Graphischen Kernsystems. Informatik-Spektrum 6, 2 (1983), pp. 55–75.

[ENDE87] G. Enderle, K. Kansy, G. Pfaff: Computer Graphics Programming, GKS – The Graphics Standard, 2nd Edition. Springer-Verlag, Berlin (1987).

[ENCA78] J. L. Encarnação: Logical Design Techniques and Tools. Siemens Forschungs- und Entwicklungsberichte 7, 6 (1978), pp. 332–335.

[ENCA81] J. L. Encarnação, W. Straßer (eds.): Geräteunabhängige graphische Systeme. Oldenbourg, München (1981).

[ENCA83] J. L. Encarnação, G. Enderle: Ein Überblick über die Entwicklung des Graphischen Kernsystems GKS. Informatik-Spektrum 6, 2 (1983), pp. 96–104.

[ENCA87] J. L. Encarnação, L. M. Encarnação, W. Herzner: Graphische Datenverarbeitung mit GKS. Carl Hanser Verlag, München (1987).

[ENDE84] G. Enderle: GKS Implementations Overview, 2nd Edition. Computer Graphics Forum, 3 (1984), pp. 181–189.

[FAER87] G. Färber: Bussysteme, serielle, parallele Bussysteme und lokale Netze. Oldenbourg, München (1987).

[FRAN86] R. Franck: Rechnernetze und Datenkommunikation. Springer-Verlag, Berlin (1986).

[GANZ81] R. Ganz, H.-J. Dohrmann: Farbgraphische Ausgabesysteme. ZwF 76, 5 (1981), pp. 223–239.

[GOER85] K. Goergen et al.: Grundlagen der Kommunikationstechnologie, ISO-Architektur offener Kommunikationssysteme. Springer-Verlag, Berlin (1985).

[HASL87] E. Haslinger: Lexikon der Personal Computer, Arbeitsplatzsysteme, Kommunikationsnetze. Oldenbourg, München (1987).

[HERZ85] W. Herzner: Einführung in GKS. Schriftenreihe der ÖGI, Wien (1985).

[HOBB81] L.C. Hobbs: Computer Graphics Display Hardware. IEEE Comp. Graph. and Appl. 1, 1 (1981), pp. 25–39.

[HOPG86] F.R. Hopgood, D.A. Duce, J.R. Gallop, D.C. Sutcliffe: Introduction to the Graphical Kernel System, 2nd Edition. Academic Press, Orlando (1986).

[I3EA85] IEEE 802.3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD), (1985).

[I3EB85] IEEE 802.4: Token Passing Bus Access, Method and Physical Layer Specifications (1985).

[ISOA85] ISO/IEC 7942: Graphical Kernel System (GKS). International Standard, (1985).

[KAUF87] F.-J. Kauffels: High Speed Local Area Networks. Datacom 5 (1987), pp. 68–72.

[KELL86] K.H. Kellermayr: Lokale Computernetze – LAN. Springer-Verlag, Berlin (1986).

[LILL81] F. Lillehagen: CAD/CAM Work Stations for Man-Model Communication. IEEE Comp. Graph. and Appl. 1, 3 (1981), pp. 17–27.

[LIND79] R. Lindner: Rasterdisplay-Prozessoren – ihre Bedeutung, Konzepte und Verfahren. Dissertation, FBInformatik, FG GRIS, Darmstadt (1979).

[MACA81] C. Machover: A Guide to Sources of Information about Computer Graphics. IEEE Comp. Graph. 1, 1 (1981), pp. 73–85.

[MACB81] C. Machover: A Guide to Sources of Information about Computer Graphics. IEEE Comp. Graph. 1, 3 (1981), pp. 63–65.

[REMM85] M. Remmele: Dezentrale Rechenzentren, Ausgabe, Verwaltung, Komponentenauswahl. Carl Hanser Verlag, München (1985).

[SALM87] R. Salmon, M. Slater: Computer Graphics, Systems & Concepts. Addison-Wesley, Wokingham, GB (1987), pp. 619–621.

[SCHI85] P. Schicker: Datenübertragung und Rechnernetze. Teubner, Stuttgart (1985).

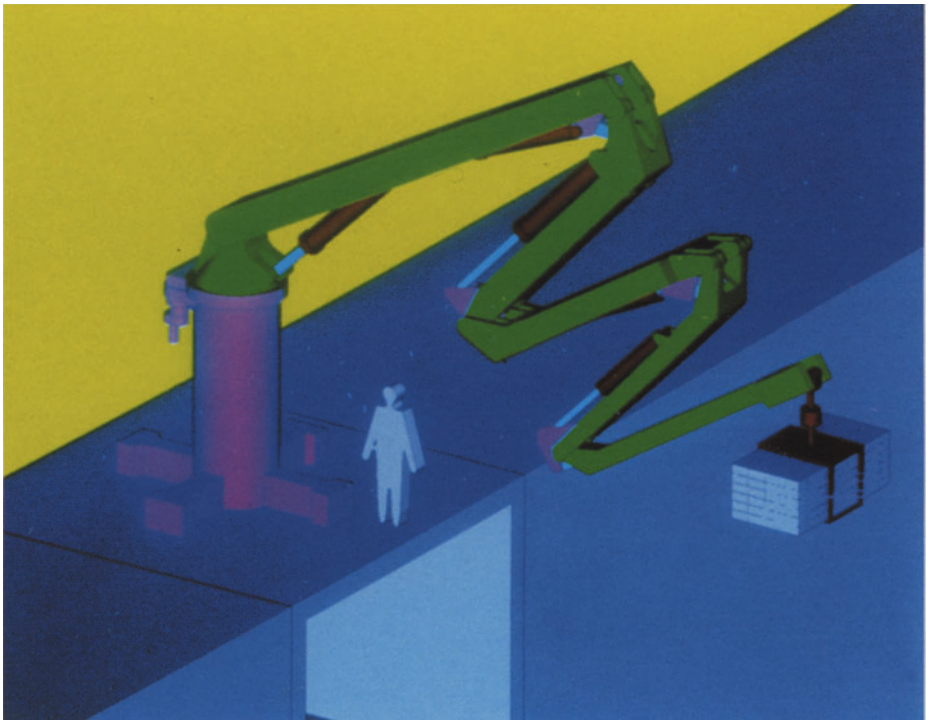
[SCHN79] P. Schnupp, Chr. Floyd: Software-Programmentwicklung und Projektorganisation, 2nd Edition. de Gruyter, Berlin (1979).

[SPRO85] R.F. Sproull, W.R. Sutherland, M.K. Ullner: Device Independent Graphics. McGraw-Hill, New York (1985).

[STRA81] W. Straßer: Hardware and System Aspects of Computer Graphics, in J.L. Encarnação, O. Torres, E. Warman (eds.), CAD/CAM as a Basis for the Development of Technology in Developing Nations. North-Holland (1981), p. 285.

[WALL76] V.L. Wallace: The Semantics of Graphics Input Devices. Computer Graphics 10, 1 (1976), pp. 62–65.

6 Engineering Methods of CAD



Simulation of a robot model
(courtesy of Kernforschungszentrum, Karlsruhe, Germany)

6.1 Geometry Handling

The statement of Voelcker and Requicha [VOEL78] "Geometry plays a crucial role in nearly all design and production activities in the discrete goods industries. Curiously, the industries' primary means for specifying geometry – two-dimensional graphics – has not changed significantly for more than a century. Dramatic changes are likely to occur in the next decade, however, because the deficiencies of current methods are retarding the progress of automation and are stimulating the development of new, computationally oriented schemes for handling mechanical geometry" is still true.

Voelcker and Requicha consider three different kinds of problems:

- Development of modeling schemes for representing as data the assemblies, stock (raw materials), and the capabilities of particular tools that affect manufacturing, assembly, and inspection processes.
- Development of algorithms that will automatically produce (from the data models of parts, assemblies, stock, and tools) manufacturing, assembly inspection plans, and command data for numerically controlled tools.
- Design, implementation, and testing of integrated computer systems which embody such representation and planning systems.

We will deal here with the first kind of problem. First we will very briefly present some fundamentals in geometry: in particular perspective transformation and rotation in 3D space; then we will discuss the problem of hidden-line and hidden-surface detection, and finally the geometric specification of parts and assemblies.

6.1.1 Introduction: Points in 3D Space

Transformations of points in 2D space and 3D space play a fundamental role in all geometric problems. The basic operations are:

- translation,
- rotation,
- scaling, and
- perspective view or projection.

Matrix algebra is the appropriate tool for performing these functions. Here we will discuss only the 3D case. For 2D, we would simply have to omit one dimension.

Points are usually represented as 3×1 column matrices or 1×3 row matrices. The operations of rotation and scaling can be represented as multiplication of the point vectors by appropriate square matrices. If row matrices are chosen for representing points (as we will do here), the multiplication is from the lefthand side: point · matrix. If one operation is to be followed by a second one, the second matrix is simply multiplied from the lefthand side again: point · matrix_1 · matrix_2.

Translation

Translation of a point is represented by adding the row matrix of the endpoint to the displacement vector. Thus, when a large number of points are undergoing the same

sequence of transformations (for example, when they are all to be rotated, then translated and finally scaled in the same way), the three operations cannot be combined into a single one, as would be possible for a sequence of three rotations (by multiplying the matrices in proper sequence). However, there is a way to convert the translation operation into a multiplication: the introduction of "homogeneous coordinates" [RIES81]:

- The point [x y z] has the homogeneous coordinates [x y z 1] or, even more generally, [ax ay az a], where a is an arbitrary scalar.

  Now, instead of translating [x y z] by adding, say, $[d_x\ d_y\ d_z]$, we write:

$$[ax\ ay\ az\ a]\cdot\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ d_x & d_y & d_z & 1\end{bmatrix}$$

Homogeneous coordinates allow us even to handle points at infinity properly: the homogeneous coordinates [x y z 0] identify a point at an infinite distance on the line directed from the origin of the 3D coordinate system towards the point [x y z].

Whether a system operates with natural coordinates or with homogeneous coordinates is an essential design decision. Systems that perform a large number of identical linear transformations will benefit significantly from homogeneous coordinates (interactive systems without hidden-line removal, for example). This statement applies particularly when special hardware is provided for performing the 4×4 matrix multiplication. Some high-performance graphics workstations offer this feature for the real-time 3D manipulation of wire-frame graphics (wire-frame model = a 3D model consisting of points and curves connecting them). For systems in which other functions dominate (such as hidden-line determination or non-geometrical applications), the natural coordinates may be the better choice.

Rotation

The rotation of a point in the x-y plane is shown in Fig. 6.1. Returning to natural coordinates, the rotation matrix $\mathbf{R_z}$ is given by Eq. (6.1):

$$\begin{bmatrix}\cos\alpha & \sin\alpha & 0\\ -\sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{bmatrix} \tag{6.1}$$

A rotation about an arbitrary axis may always be defined in terms of a sequence of three rotations by τ_x, β_y, α_z around the three axes (x, y, z), respectively. As each of the rotations is defined by a rotation matrix like Eq. (6.1), the complete 3D rotation is defined by the overall rotation matrix $\mathbf{R}$ where

$$\mathbf{R} = \mathbf{R_x}\cdot\mathbf{R_y}\cdot\mathbf{R_z} \tag{6.2}$$

with

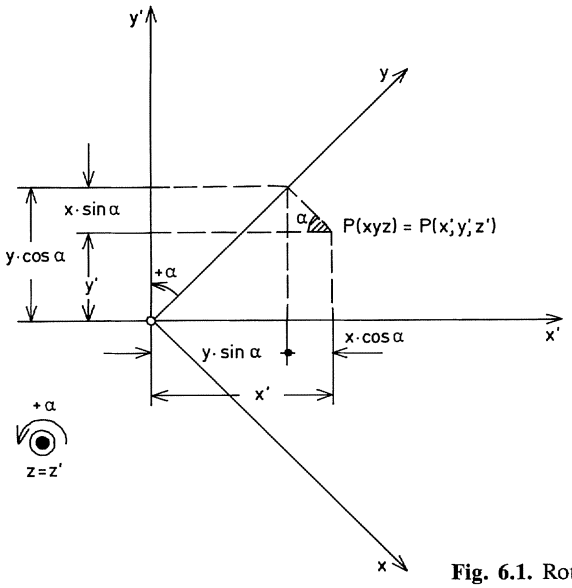


Fig. 6.1. Rotation in the xy plane

$\mathbf{R_x}$ = rotation in the yz-plane about the x axis followed by
$\mathbf{R_y}$ = rotation in the xz-plane about the y axis followed by
$\mathbf{R_z}$ = rotation in the xy-plane about the z axis.

The rotated point has the new natural coordinates

$$[x\ y\ z]\cdot\mathbf{R} \tag{6.3}$$

where

$$\mathbf{R} = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \tag{6.4}$$

and

$A = \cos\beta_y \cdot \cos\alpha_z$
$B = \cos\beta_y \cdot \sin\alpha_z$
$C = -\sin\beta_y$
$D = \sin\beta_y \cdot \sin\tau_x \cdot \cos\alpha_z - \cos\tau_x \cdot \sin\alpha_z$
$E = \sin\beta_y \cdot \sin\tau_x \cdot \sin\alpha_z + \cos\tau_x \cdot \cos\alpha_z$
$F = \cos\beta_y \cdot \sin\tau_x$
$G = \sin\beta_y \cdot \cos\tau_x \cdot \cos\alpha_z + \sin\tau_x \cdot \sin\alpha_z$
$H = \sin\beta_y \cdot \cos\tau_x \cdot \sin\alpha_z - \sin\tau_x \cdot \cos\alpha_z$
$I = \cos\beta_y \cdot \cos\tau_x$

Equations (6.3) and (6.4) describe a rotation around the origin of the coordinate system. The rotation around some arbitrary rotation center $[x_c\ y_c\ x_c]$ can be achieved

by first translating the point by $-[x_c\ y_c\ x_c]$ (which would bring the rotation center into the origin), then rotating and finally translating back by $+[x_c\ y_c\ x_c]$. The advantage of using homogeneous coordinates is that these three operations can very easily be combined into a single matrix multiplication.

Scaling

Scaling of a point also fits into the matrix multiplication scheme. A point whose coordinates should be scaled by factors s_x, s_y, s_z in the x, y, and z directions will have the new natural coordinates given as:

$$[x\ y\ z]\cdot\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \tag{6.5}$$

Scaling with respect to an arbitrary point is best done in homogeneous coordinates in a similar way as rotating around an arbitrary point.

Projection

For perspective view or projection, we restrict our discussion first to the central projection from an origin that is located on the z axis at $[0\ 0\ c_z]$ onto a plane parallel to the xy plane at an elevation q_z. The coordinates [X Y] of the projected point are then given by:

$$\begin{aligned} X &= \frac{c_z - q_z}{c_z - z}\cdot x \\ Y &= \frac{c_z - q_z}{c_z - z}\cdot y \end{aligned} \tag{6.6}$$

For the special case where $q_z = 0$, we obtain

$$\begin{aligned} X &= \frac{c_z}{c_z - z}\cdot x = \frac{1}{1 - z/c_z}\cdot x \\ X &= \frac{c_z}{c_z - z}\cdot y = \frac{1}{1 - z/c_z}\cdot y \end{aligned} \tag{6.7}$$

Using homogeneous coordinates, we obtain a rather simple 4×4 matrix for this projection. The projected point has the coordinates

$$[ax\ ay\ az\ a]\cdot\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.8}$$

A general central projection from an arbitrary point onto an arbitrary projection plane may be obtained by combining rotation and translation operations so as to achieve the standard situation described here. With homogeneous coordinates, these operations may be combined into a single matrix. For further details on transformations in 3D space see [GILO78], for example.

In CAD, parallel normal projection plays a central role, as standard design drawings represent their objects in this way. For a top view of an object, we have to project its points onto the xy plane from a point at infinity on the z axis. Using Eq. (6.7), we obtain for this standard case:

$$\lim_{c_z \to \infty} \frac{c_z}{c_z - z} = 1 \tag{6.9}$$

or, as we would expect:

$$\begin{aligned} X &= x \\ Y &= y \end{aligned} \tag{6.10}$$

This projection has several advantages. It clearly avoids a tangential intersection of line-of-sight and the projection plane, thus eliminating the problem of having to handle points at infinity. It avoids strange distortions of the projected picture, which sometimes make it difficult to recognize an object from a central projection. It requires a minimum of computation, as the projection simply implies dropping one of the coordinates of a 3D point. For top view, the z coordinate has no influence on the representation: it is only used for visibility testing. Parallel normal projection minimizes the program runtime and simplifies the visibility-test procedures.

The most general projection implies that we specify independently:

- the projection origin;
- the projection plane; and
- origin and directions of the 2D coordinate system in the projection plane.

Schuster [SCHU76] has treated the general projection problem using vector algebra in natural coordinates instead of homogeneous coordinates. Here, we briefly outline his approach. (**Bold face** letters indicate vectors in the subsequent paragraphs.) A point **p** is to be projected onto plane B. B is defined by a point $\mathbf{r} \in B$ and the normal vector **n** (see Fig. 6.2):

- 3D space: coordinates (x, y, z) with base vectors $\mathbf{e_x}$, $\mathbf{e_y}$, $\mathbf{e_z}$;
- 2D space: coordinates (X, Y) with base vectors $\mathbf{e_X}$, $\mathbf{e_Y}$; these unit vectors will be chosen such that the origin of the (X, Y) coordinate system coincides with the projection of the origin of the 3D space; the directions of the 2D coordinate system will be defined later;
- plane B: point **r** and normal **n**;
- 3D point: **p**;
- projection origin: **q**; and
- projection point: **x** in 3D space, **X** in 2D space.

The plane is given by equation

$$(\mathbf{x} - \mathbf{r}) \cdot \mathbf{n} = 0 \tag{6.11}$$

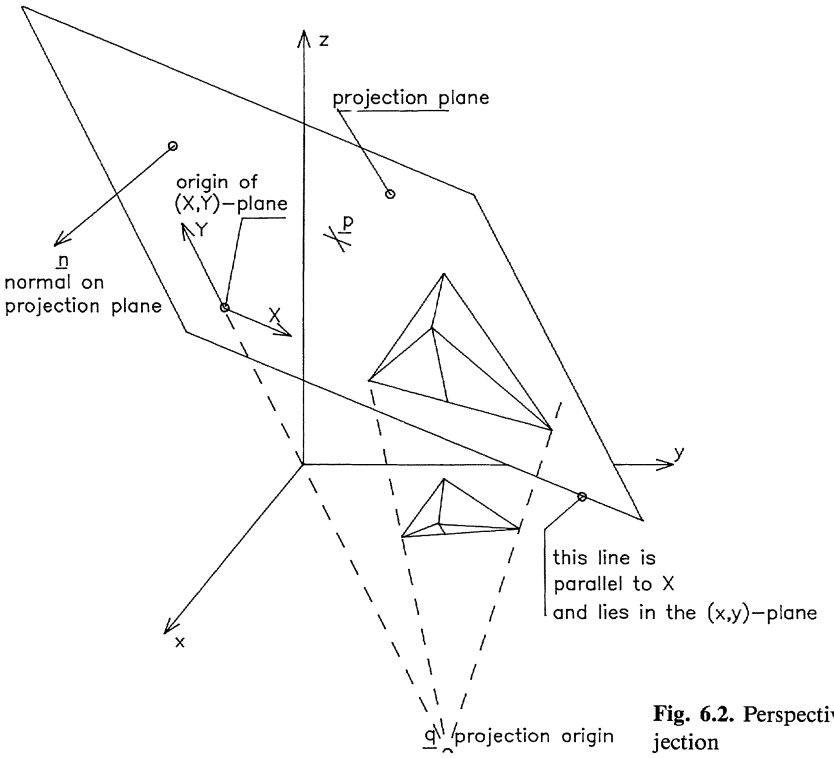


Fig. 6.2. Perspective projection

The projection beam through **p** and its projection **q** is given in vector notation as:

$$\mathbf{x} = \mathbf{p} + t(\mathbf{q} - \mathbf{p}) \tag{6.12}$$

with t as a scalar parameter.

From these two equations we obtain immediately:

$$(\mathbf{p} + t(\mathbf{q} - \mathbf{p}) - \mathbf{r}) \cdot \mathbf{n} = 0 \tag{6.13}$$

We can solve this equation for the parameter t, which can then be inserted into (6.12) to obtain the equation for the projected point in 3D space:

$$\mathbf{x} = \frac{((\mathbf{q} \cdot \mathbf{n}) - (\mathbf{r} \cdot \mathbf{n}))\mathbf{p} + ((\mathbf{r} \cdot \mathbf{n}) - (\mathbf{p} \cdot \mathbf{n}))\mathbf{q}}{(\mathbf{q} - \mathbf{p}) \cdot \mathbf{n}} \tag{6.14}$$

X in the 2D space must coincide with x in the 3D space:

$$X = X\mathbf{e}_X + Y\mathbf{e}_Y = \mathbf{x} \tag{6.15}$$

So far, the base vectors $\mathbf{e}_X$ and $\mathbf{e}_Y$ in the 2D space are arbitrary. The x direction $\mathbf{e}_X$ may, for instance, be chosen to coincide with the line of intersection of the projection plane B with the xy plane in the 3D space, in such a way that the 3D space origin is projected onto the 2D space origin. The equations of these planes in 3D space are as follows (with subscripts 1, 2 and 3 indicating the vector components in the 3D space):

B plane: $n_1x+n_2y+n_3z=0$
x–y plane: $z=0$

Thus, after normalization, we obtain:

$$\mathbf{e}_x=\frac{1}{(n_1^2+n_2^2)^{1/2}}\begin{bmatrix}-n_2\\ n_1\\ 0\end{bmatrix} \tag{6.16}$$

The Y component must be orthogonal to both $\mathbf{e}_x$ and $\mathbf{n}$. Hence:

$$\mathbf{e}_y=\mathbf{n}\times\mathbf{e}_x \text{ (vector cross product)} \tag{6.17a}$$

$$\mathbf{e}_y=\frac{1}{(n_1^2+n_2^2)^{1/2}}\begin{bmatrix}-n_1n_3\\ -n_2n_3\\ n_1^2+n_2^2\end{bmatrix} \tag{6.17b}$$

With Eqs. (6.14), (6.15), (6.16), and (6.17), we can now determine the components X and Y of the 2D vector **X**. Equations (6.14) and (6.15) are used, together with $\mathbf{e}_x$ and $\mathbf{e}_y$ as expressed by Eqs. (6.16) and (6.17). Multiplication by $\mathbf{e}_X$ makes the $\mathbf{e}_y$ component disappear. We obtain:

$$X=\frac{(\mathbf{q}\cdot\mathbf{n}-\mathbf{p}\cdot\mathbf{n})(\mathbf{p}\cdot\mathbf{e}_X)+(\mathbf{r}\cdot\mathbf{n}-\mathbf{p}\cdot\mathbf{n})(\mathbf{q}\cdot\mathbf{e}_X)}{(\mathbf{q}-\mathbf{p})\cdot\mathbf{n}} \tag{6.18}$$

Similarly, multiplying Eq. (6.15) by e_y results in:

$$Y=\frac{(\mathbf{q}\cdot\mathbf{n}-\mathbf{p}\cdot\mathbf{n})(\mathbf{p}\cdot\mathbf{e}_y)+(\mathbf{r}\cdot\mathbf{n}-\mathbf{p}\cdot\mathbf{n})(\mathbf{q}\cdot\mathbf{e}_y)}{(\mathbf{q}-\mathbf{p})\cdot\mathbf{n}} \tag{6.19}$$

After performing the vector multiplication and introducing the abbreviations:

$|\mathbf{n}_2|=(n_1^2+n_2^2+n_3^2)^{1/2}=1$
$L_i^2=1-n_i^2$
$d=\mathbf{q}\cdot\mathbf{n}$
$s=\mathbf{r}\cdot\mathbf{n}$

we obtain finally:

$$X=\frac{s(q_2n_1-q_1n_2)+p_1(sn_2-q_2L_3^2-q_3n_2n_3)+p_2(-sn_1+q_1L_3^2+q_3n_1n_3)+p_3n_3(q_1n_2-q_2n_1)}{L_3d-p_1n_1L_3-p_2n_2L_3-p_3n_3L_3}$$

$$Y=\frac{s(q_2n_2n_3-q_1n_1n_3+q_3L_3^2)+p_1n_1(sn_2-q_3)+p_2n_2(sn_3-q_3)+p_3(q_1n_1+q_2n_2-sL_3^2)}{L_3d-p_1n_1L_3-p_2n_2L_3-p_3n_3L_3} \tag{6.20}$$

The complete set of transformations that is relevant to three-dimensional geometry handling may be classified into subsets using the criteria of mathematical

theory [KLEM87], [LIET55]. Three subsets of transformations can be identified, each of them building a group and defining a related geometry:

1) The Metrical Geometry is defined by concatention of translations and rotations. This group of transformations deals with the objects themselves and not with their particular position in space.
2) The Affine Geometry is defined by concatenation of translations, rotations, and scalings. This group of transformations deals with finite objects and does not modify relative angles (parallel lines remain parallel).
3) The Projective Geometry is defined by concatenation of translations, rotations, scalings, and perspectives. This group of transformations is the most general formulation of transformations.

Group theory produces the following important results which allow the meaningful use of each of the groups in CAD applications:

1) The concatenation of several transformations taken from any single group is a transformation which again belongs to the same group.
2) The concatenation of transformations taken from any single group is associative.
3) The identity transformation belongs to each group.
4) The inverse of a transformation taken from any single group also belongs to that group.

6.1.2 The Hidden-Line/Hidden-Surface Problem

6.1.2.1 General Considerations

The problem of eliminating the hidden planes and edges of non-transparent 3D solids, the so-called visibility problem, has been tackled since the middle of the 1960s. Various algorithms have been designed by Appel, Encarnação, Galimberti and Montanari, Loutrel, Newell, Roberts, Schumacker, Warnock, Watkins, Weiss. For surveys see [SUTH74], [ENCA75], [NEWM79], [GILO78].

Visibility algorithms may be classified as hidden-surface algorithms and hidden-line algorithms. Hidden-line algorithms are designed for edge-oriented output tools, such as vector displays and plotters; hidden-surface algorithms are oriented towards raster output devices.

The basic procedural kernel of all these algorithms usually follows one of three distinct strategies, which may be classified as:

- surface test;
- point test; or
- combined surface/point test.

Surface test: Here, as the name implies, a surface element is the basic entity tested. (A surface element is a portion of the whole surface. Its interior is described by some mathematical form, and it is connected to adjacent surface elements along its edges). In its elementary form, this test deals with planar faces only. Its basic idea is that faces whose outward normal vector points towards the projection point, are visible, while

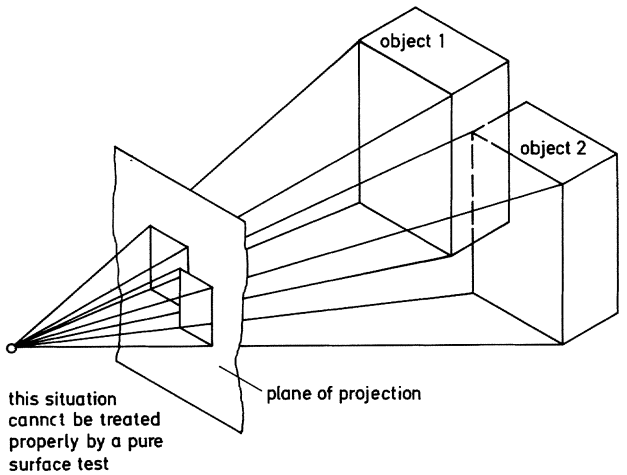


Fig. 6.3. The surface test cannot handle all situations properly

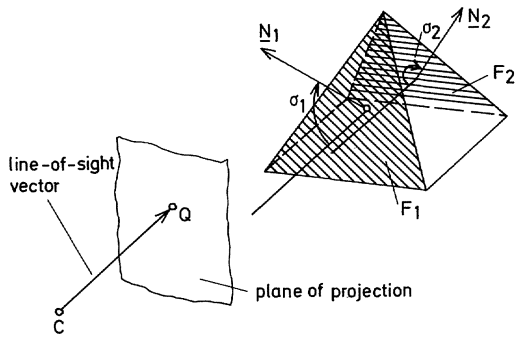


Fig. 6.4. Principle of the surface test

all others are invisible. This test does not take into consideration that a surface element of a solid may be hidden by another surface element of the same body, or of another body. Thus it can be applied to single convex solids only. Furthermore, the surface test assumes that all edges of a visible face are entirely visible. Figure 6.3 shows an example where a surface test would fail to determine the visibility properly. It could not determine that part of the front surface of solid 2 is hidden by solid 1. Formally, in the surface test the angle σ between the line of sight (from the projection origin to a face) and the normal vector **N** of the face (pointing outward from the solid) is determined from the inner product of these vectors according to Eq. (6.21). If the inner product is negative (or $\sigma < 90°$), then the face and of all its edges are visible as a whole:

$$\mathbf{N} \cdot \mathbf{CQ} = |\mathbf{N}| \; |\mathbf{CQ}| \cos(180° - \sigma) \leq 0 \tag{6.21}$$

N = external surface normal
CQ = line of sight

Hence, in Fig. 6.4 face F1 is visible, while face F2 is invisible. Since the surface test considers whole surfaces rather than single points, it is very fast but has limited

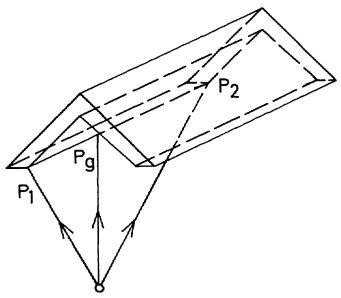


Fig. 6.5. Example illustrating the point test

applicability. A pure surface test can only be applied to *convex solids*, and this is often too strong a limitation.

Point test: In this method, a line or curve is broken up into very small segments, each of which is drawn only if a testpoint on the segment is not hidden by any surface in space. In Fig. 6.5, the line from $\mathbf{P}_1$ to $\mathbf{P}_2$ consists of two parts $\mathbf{P}_1 - \mathbf{P}_g$ and $\mathbf{P}_g - \mathbf{P}_2$. With a pure point test, many test points on $\mathbf{P}_1 - \mathbf{P}_2$ would have to be considered in order to locate $\mathbf{P}_g$ properly. This approach requires large amounts of storage space and computer time. In principle, these tests have the advantage of being universally applicable, notably even for curved edges resulting from sculptured surfaces. However, because of the computer resources required, the pure point tests are not practical.

Combined point/surface test: In this category we place all procedures that attempt to combine the two tests. The manner in which this combination is implemented distinguishes the visibility test procedures that have been published.

We will now present two procedures of the combined point/surface test type.

6.1.2.2 The Priority Procedure

This procedure was developed for the solution of the hidden-line problem for solids bounded by *planar faces*. The priority procedure is very general. It can treat more than a single solid and is not restricted to convex shapes. Hence, holes, gaps, and even individual surfaces (without thickness) can be treated. As a preparation for the priority procedure, the surface of the solid must be broken up into triangles (regardless of what its original representation was). For planar faces, this is a straightforward operation; but curved surfaces must be approximated by joining triangles. This segmentation into triangles will not be described further here (see [MESC66], for example). From now on, we will consider the solid's surface as an unordered set of triangles. It is now possible to devise the visibility strategy in a way that will highly optimize running time and storage requirements.

The priority procedure consists of two main steps:

1) assignment of priority;
2) determination of coverings.

Without loss of generality, we can assume that the projection is onto the xy plane with the projection origin on the positive z axis. This standard situation can always

be achieved by rotating and shifting the projection plane, the projection origin, and the solid accordingly. We start by collecting all surface elements F_i in a list, which we are now going to order according to their priority.

1) Assignment of Priority

This step of the algorithm determines the order of processing of the triangles due to mutual hidings (involving the hiding of one surface by another on the same solid).

Given a set of triangles, let:

g, **h** = surface points, defined by their coordinates x, y, z;
i, j = running indices for surfaces;
F = set of all surfaces.

We investigate points **g** and **h** of surfaces F_i and F_j:

$\mathbf{g} \in F_i$
$\mathbf{h} \in F_j$

The triangles to be processed are projected by the projection operator f onto the xy plane:

$f: F_i \rightarrow A_i$
$f: F_j \rightarrow A_j$

where

A = projection of F.

The elements of the sets A_i and A_j (the projected points) are defined by the x and y coordinates of their corners. A point **c** is now computed, such that (Fig. 6.6a):

$\mathbf{c} \in D$ where
D = intersection (A_i, A_j)

Hence, **c** is within the overlapping part of two projected triangles. From the inverse mapping of this point **c** to F_i and F_j, one obtains the two corresponding z coordinates on faces F_i and F_j. The highest z coordinate of **c** among these sets determines the highest priority. The list of all elements F_i is continuously reordered according to this priority.

If the intersection is the empty set, no priority assignment is possible. In this case, the surface F_i is exchanged with the last element of the surface list, the list length is shortened by one, and the sorting process to determine priority starts anew. After completion of the algorithm, the result is a list of triangles, which can be processed from top to bottom to determine mutual coverings. Fig. 6.6b shows an example. In this case the priority list would be as follows: 1, 2, 3, 5, 4.

In the case of a raster display, we have now almost completed the whole task: we simply output all elements in the inverse order of their priority. Any surfaces elements that would have to be hidden by others would thus be displayed first, but would be covered later either partially or entirely by elements of higher priority. In the case of vector-oriented output devices, however, we have to continue the hidden-line removal by software.

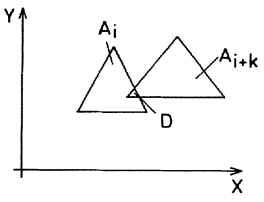
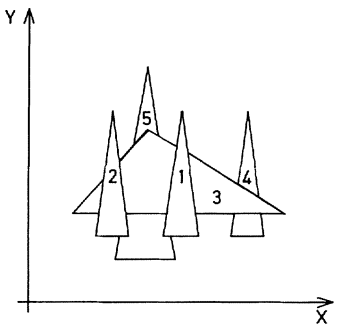


Fig. 6.6a. Determination of intersections in the projection for the priority procedure



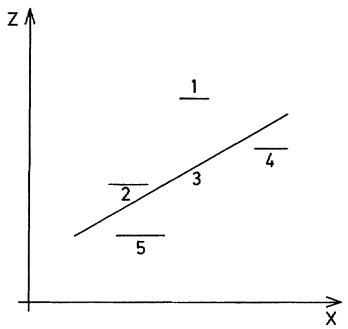


Fig. 6.6b. Assignment of priority

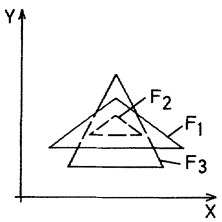


Fig. 6.6c. Removal of completely hidden boundaries

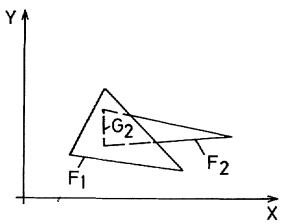


Fig. 6.6d. Removal of boundary lines (line G_2)

2) Determination of Coverings

After determination of priorities, we must investigate which parts of the triangles are to be drawn and which are not. For this purpose, all three boundary lines of a triangle are compared with all other surface edges of higher priority. We maintain a list of all (straight) line segments that will have to be drawn as visible. If a face is declared fully invisible, its boundary lines are removed from the line list, like those of F_2 in Fig. 6.6c. The next covering surface is then immediately examined.

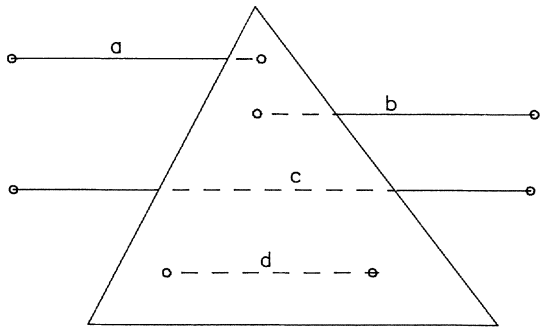


Fig. 6.7. Different line-covering situations encountered in the priority method

If a surface is partially visible, then only the completely hidden lines are removed from the line list (line G_2 in Fig. 6.6d). Maintaining the line list raises the storage capacity requirements of this method, but saves substantial amounts of processing time.

The algorithms that determine coverings must distinguish between the following four situations, which are illustrated in Fig. 6.7:

a) the end point of a partially hidden line is hidden;
b) the starting point of a partially hidden line is hidden;
c) both starting point and end point are not hidden;
d) both starting point and end point are hidden.

The formal treatment of these situations and their algorithmic implementation are described in [ENCA75].

6.1.2.3 The Overlay Procedure

The overlay procedure is applicable to solids whose surfaces are each defined by a so-called u-v grid. Each surface element is defined by a function with two parameters u and v:

$$\text{face}_j = \{f_j(u, v),\ u_{j1} \leq u \leq u_{j2},\ v_{j1} \leq v \leq v_{j2}\}$$

Thus, each surface element may be considered as being spanned by a u-v line grid. The procedure uses the overlaying of an imaginary Cartesian grid (on the projection plane) upon the projection of the u-v grid of the surfaces.

This procedure consists of the following steps:

a) calculation of the u-v line grid;
b) construction of the Cartesian grid;
c) assignment of u-v elements to the Cartesian grid;
d) calculation of the visibility of the nodes; and
e) determination of the visible u-v line elements.

Calculation of the u-v line grid:
Using the corresponding surface equations, the individual nodes of all the u-v line intersections are determined. Each node belongs to exactly one u-line and one v-line (see Fig. 6.8) and is determined by its x, y, and z coordinates.

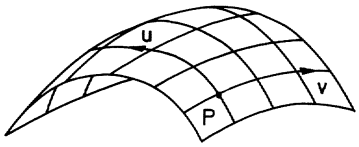


Fig. 6.8. Coordinates of the nodes of a surface element

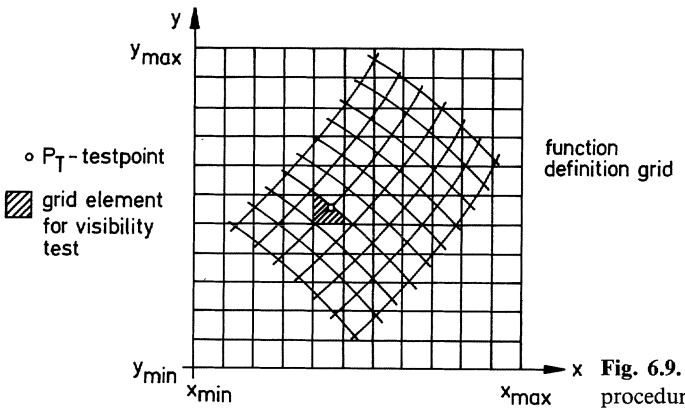


Fig. 6.9. The "overlaying" procedure

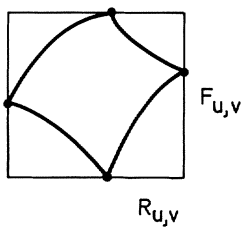


Fig. 6.10. The approximating rectangle $R_{u,v}$

Construction of the Cartesian grid:
Imagine an n by n Cartesian grid drawn on the projection plane. The grid lines form rectangles which will be used as a basis in the subsequent tests. The size of the grid can be chosen arbitrarily. In Fig. 6.9 a value of n = 11 was selected. The surface element defined by the u-v lines is overlaid on this grid. If a node is to be tested for visibility, we first determine which rectangle contains its projection. The visibility test now concerns itself only with this rectangle, and not with the whole surface. All the surface elements having sets of points in this rectangle must now be determined.

Assignment of the u-v elements to the Cartesian grid:
In this step, the individual surface elements are in this step approximated by rectangles, as shown in Fig. 6.10. The rectangles are constructed such that, after projection, their sides run parallel to the x and y axes. This is done to simplify the programming and to minimize computer running time. In the following steps d) and e), however, the true u-v elements will be considered instead of the approximation. Since the individual u-v elements are completely enveloped by their linear appoximation, no information needed for determining the visibility is lost.

Calculation of the visibility of the nodes:
First the Cartesian grid lines which envelop the test point are determined (see Fig. 6.9):

$$x_n \leq x \leq x_{n+1}$$

$$y_m \leq y \leq y_{m+1}$$

x, y = coordinates of the projection of the node to be tested.

Then we determine the surface elements whose projections have a non-empty intersection with the rectangle. For this purpose, we have to determine all elements which contain a point that will coincide with the test point after projection. The u-v elements are broken into two triangles and approximated by a plane. Now the triangles containing the test point are determined, and all corresponding z coordinates z_D are computed using the planar approximation. Now we compare all values of z_D and consider all nodes with

$$z_D > z_M$$

as being invisible, where z_M is either a predefined value or the minimum value of all z_D's corresponding to the same test point.

Determination of visible u-v line elements:
After having determined the visibility of the u-v nodes on all surfaces, the next task is to test the visibility of the connecting u-v grid lines. Visibility of two adjacent points does not necessarily imply that the whole connecting line is visible. Parts of the connection may be hidden by other surface elements. This is particularly important for coarse grids. Six different situations may be encountered, as listed in Fig. 6.11.

When only one end point of a u-v grid node connection is visible, we have to examine the connecting projection by means of test points in order to determine where the

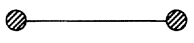
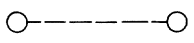
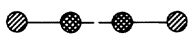
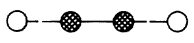
no.	case	comment	meaning
1		1st end-point is visible	... visible node
2		2nd end-point is visible	... invisible node
3		both points are visible	... visible intermediate point
4		both points are invisible	... visible line element
5		a test point on the u-v line not visible	... invisible line element
6		a test point on the u-v line is visible	

Fig. 6.11. Possibilities for the connection of nodes

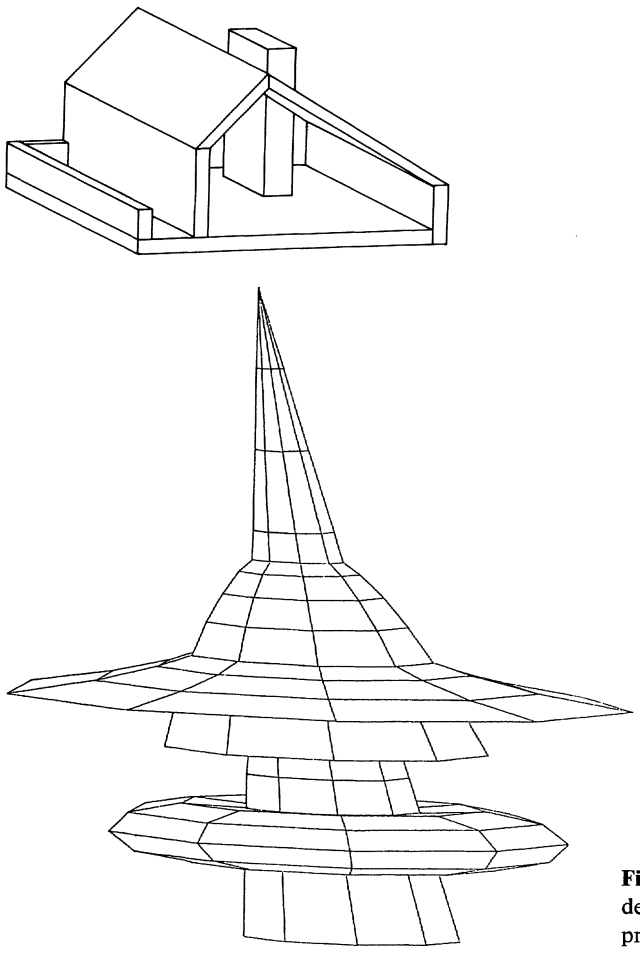


Fig. 6.12. Examples of hidden-line elimination with the priority procedure

visibility ends. This testing proceeds in discrete steps, which have to be chosen as a compromise between accuracy (small steps) and running time (large steps). The same procedure is required for testing the connection between two visible or two invisible nodes in order to detect any section of the connecting line that may be invisible (or visible).

Figure 6.12 shows two examples, which were projected using the two visibility test procedures described above.

6.1.2.4 Generalization of the Visibility Problem

A visibility algorithm consists of several major steps. Each step provides a particular mapping, and the total algorithm is a concatenation of such mappings. Consequently, interposed between the domain of a visibility transformation (the set of 3D objects) and its range (the set of visible segments), there may exist a sequence of intermediate representations. A formal definition of a visibility algorithm is possible.

Definition:
A visibility algorithm is a quintuple

$$VA = \{O, S, I, \Sigma, \Phi\}$$

where

O is a set of 3D objects. A 3D object is defined as a set of coordinates plus a set of relations specifying the 3D object topology. In general, a topology may be represented as a tree. "3-dimensional scene" is the root; the nodes are 3D objects, faces, edges, and start/end points of edges;

S is a set of visible segments in 2D (the result of the hidden-surface transformations). These segments are the visible parts of the elements of O;

I is a set of "intermediate representations";

Σ is a set of strategy functions which control the sequence of application of all other functions of the algorithm;

Φ Set of "transition functions" = {PM, IS, CT, DT, VT}

where

PM is a function that produces the perspective views ("projective mapping"). Hence, the domain of PM is 3D, and its range is 2D;

S is a function that calculates the intersection of two graphical items. In 2D the items are two line segments, in 3D they are a polygon and a line segment;

CT is a function that performs a "containment test" in 2D. CT checks whether a point is inside a given bounded surface. The result of CT is Boolean. It is "true" if the point is contained, and "false" otherwise;

DT is a function that performs a "depth test". DT compares two points and finds out which one has the greater depth, depending on the point of observation;

VT is a function that performs a "visibility test" for a given surface. VT yields a Boolean value, "true" if the surface is potentially visible and "false" if the surface is totally invisible.

Using this definition, visibility algorithms can now be formalized and represented graphically by "strategy diagrams" [GILO78]. However, it is not a trivial task to convert the different algorithmic formulations of visibility-test procedures into a form that allows them to be mapped onto the generalized scheme. For seven visibility-test procedures, namely

- Appel's "quantitative invisibility" method;
- Encarnação's "priority" method (Sect. 6.1.2.2);
- Galimberti and Montanari's "nature" method;
- Warnock's "scan grid" method;
- Watkins' "scan line" method;
- Encarnação's "overlay" procedure (Sect. 6.1.2.3); and
- Weiss' "analytical" method;

a generalization is possible. Without going into further detail, let us note that such a generalization is required if one attempts to design and build a special-purpose computer for hidden-line and hidden-surface removal, which is not to be restricted to a single algorithm [HORN81].

6.1.3 3D Modeling

6.1.3.1 Introduction

During the design of a product, many aspects have to be considered, such as:

- function,
- shape,
- manufacturability,
- maintenance, and
- economics.

The design of the shape is a central part of all the activities, as all other aspects have a significant influence on the shape. Some branches of industry are concerned mainly with 2D shapes: electronic circuit layout, plant layout, and others. But even in these applications, the 3D aspect will have to be considered to some extent. For many applications the "$2\frac{1}{2}$D" approach is sufficient. The term $2\frac{1}{2}$D does not have a precise definition; it merely indicates that not all aspects of three-dimensional geometry have to be fully considered. Geometrical arrangements that can be described as layers of 2D layouts are called $2\frac{1}{2}$D problems. Electronic circuit boards are a typical example. The third dimension is often indicated merely by an integer identifying the sequence number of the respective layer. In $2\frac{1}{2}$D problems, we know automatically that the individual objects cannot cross each other, nor can they be folded. Hence, the algorithms can be much simpler and faster than in the fully 3D case. A different case of "$2\frac{1}{2}$D" geometry is the design of objects with rotational symmetry in mechanical engineering; even the design of gearings does not yet involve full 3D problems, as long it is merely concerned with bodies of revolution with parallel axes. CAD methods for dealing with 2D and $2\frac{1}{2}$D objects were fairly well established by the end of the 1970s. The same is true for many types of objects in 3D space, particularly for objects with a single dominant dimension (networks of trusses, frames, or pipes). Surfaces in space (flat and sculptured) also have a deep and sufficiently broad theoretical foundation, upon which systems have been built. The theory of 3D objects with unrestricted complexity, however, is not yet fully developed. All systems that have been developed have certain restrictions with respect to their applicability, even when they are able to model almost everything by means of approximations.

For computer-aided design of three-dimensional objects, the geometric aspect of part and assembly specifications is critically important [VOEL78]. Standard drafting practice suggests that geometric specification should be viewed as a two- or three-phase process. Initially, a nominal or ideal 3D object – a "shape" – is defined, typically by a drawing that does not account for tolerances. In the second phase, tolerances are introduced; at this point one no longer defines a single 3D object, but rather a class of 3D objects which are functionally equivalent and interchangeable in assembly processes. Attributes that would be conveyed by notes in engineering drawings are specified in the final phase, or in conjunction with tolerancing.

Engineering drawings are an imperfect medium for the specification of parts. Because engineers and technicians possess vast stores of pertinent "world knowledge"

(the purpose of the device, general mechanical principles, etc.), they usually can extract from drawings the information needed to make and assemble parts correctly. Machines – or programs for interpreting drawings – usually cannot. Thus, new approaches are needed for the difficult problem of precisely specifying (to automatic manufacturing systems) what is to be made.

It is not hard to devise ad hoc schemes for manipulating geometry in computers, but these systems lack some basic properties that are essential for fully automatic production. For example: a reliable representation scheme should be complete and consistent; every part in a given class should have a representation, and every representation should specify exactly one part. None of the industrial graphics systems popular today exhibits these properties.

But there is more to geometric specification than nominal shape description. Any industrially viable medium must provide means for specifying tolerances, surface finishes, and similar geometric attributes in a complete and consistent manner. Further, a viable system must be convenient for others to use and it must be reasonably efficient.

6.1.3.2 Wire-Frame Models

Wire-frame models already have a long tradition in modeling 3D objects. Stress and strain analysis of truss and frame structures was among the first large-scale computer applications for design analysis. The geometric model for these analyses is immediately suited for graphic representation. The (usually straight) elements of the structure may be plotted directly after perspective projection of the corresponding nodes. The elementary geometrical analysis tools indicated in Sect. 6.1.1 are sufficient for this task. Curved structural elements do not pose any serious additional problems.

The same technique has been applied successfully for plant layout. In many respects, pipes may be represented as straight or curved lines. Components (vessels, pumps etc.) can be approximated by a wire frame to some extent for visualization purposes. A cylindrical vessel, for instance, needs only two circles and four straight connecting lines in a coarse wire-frame model. However, this approach has its limitations, where the number of lines in a model becomes more abundant. The perspective view will then produce a mere mess of lines, from which the observer can no longer reconstruct a mental model of the three-dimensional situation. The two principal limitations with respect to visualization are:

- the impossibility of removing lines that ought to be considered as hidden; and
- the lack of contour lines, which result from viewing sculptured surfaces.

Wire-frame models are suited for performing certain geometrical analyses with a design: for example, the distance of points can easily be retrieved. However, the wire-frame model can produce no information about surfaces and volumes. Questions of surface area, volume, weight, or possible interference of some body with another will remained unanswered. In fact, there is no way besides visual inspection of a number of perspectives by humans to tell whether the model represents a feasible 3D body or assembly of bodies.

Despite these limitations, we should not forget that many existing CAD systems are based on wire-frame models, and also that there are many useful applications for these models in the early phases of design. The simplicity of the geometrical algorithms (based on points and their connections) lends itself to hardware implementations, to that the model or the viewing point may be changed in real time under the control of the operator at a workstation. Many design variants can thus be constructed and inspected with a minimum of delay.

6.1.3.3 Surfaces in Space

The next step toward increased complexity is the treatment of surfaces in space. Depending on the type of surfaces, we distinguish systems for

- flat surfaces;
- sculptured surfaces based on flat surface approximations;
- sculptured surfaces based on patches or u-v grid lines;
- analytic surfaces; and
- combinations of these types.

The approximation of surfaces in space by flat elements has become common practice in finite-element analysis of membrane, shell, or plate structures. Accuracy of analysis requires that the surface be broken down into a large number of elements anyway. The degree of approximation required for the analysis is generally sufficient for all visualization purposes. For complicated shapes, hidden-line removal is mandatory but, as only flat surfaces are involved, a large number of visibility-test procedures are at the disposal of the application programmer. Quite often, a pure surface test (see Sect. 6.1.2) is sufficient, at least for some suitably chosen projections.

Many applications, however, call for the modeling of smooth surfaces. Typical such areas may be found in the design of aircraft, ships, and cars. The design task is generally the smoothing (or fairing) of a previously rough surface approximation. Various optimization criteria may have to be applied (steady change in curvature, elimination of changes in sign of curvature, or minimization of some weighted function of curvature). Only in rare cases can we describe the whole surface by a single analytical function. In general, the surface has to be composed from patches that join at their edges. The representation of sculptured surfaces as joining patches in parametric form,

$$x = x\,(u, v, \{p\});\ y = y\,(u, v, \{p\});\ z = z\,(u, v, \{p\})$$
$$\text{with } u1 < u < u2;\ v1 < v < v2$$
$$\text{and } \{p\} = \text{patch parameters}$$

has proven to be a most powerful method (see Sect. 6.1.3.5). The patch parameters {p} are best defined on the basis of values associated with the nodes of the patch (coordinates, derivatives, and curvature).

Different methods for representing these patches can guarantee various degrees of continuity along the edge (as when continuous curvature is required). Coons' representation of patches was the pioneering work in this field [COON67]. Coons'

patches belong to the *local* type of surface representation: if some change is made to one point, which forms the basis of some adjacent patches, then only the immediate environment of the surface is influenced. Other methods (such as Gordon's blending function interpolation [GORD71] are more global: a local change will influence the surface everywhere. One can notice an increasing trend towards the use of Bsplines [FORR72], [GORD74].

The elementary operations which ought to be provided by a system that deals with sculptured surfaces are [NOWC80]:

- definition and modification of surfaces;
- *interpolation*; that is, the evaluation of points that lie on the surface between the nodes of the defining u-v grid;
- computation of the lines of intersection of the surface with an arbitrary plane;
- computation of the lines of intersection of two surfaces; and
- *fairing*; that is, the determination of a surface that approaches a given rough approximation according to a specified smoothing criterion.

For sculptured surfaces, as well as for non-convex 3D objects, we have the very general problem that the edges of faces may be partially visible (parts of the edges may be hidden by faces of the same or other 3D objects). Hence, for a complete and accurate visibility test, we have to follow *all* the edges of all potentially visible faces and test (almost) *all* points on them to see whether they are hidden by any other face. The complexity of this task rises dramatically as the number of 3D objects (or the number of faces of each 3D object) is increased. The performance of 3D object modeling programs depends strongly upon their strategies to eliminate as quickly as possible many of the faces against which a point has to be tested. It is the old principle of "divide and conquer" that has to be used to cut the immense problem into a number of smaller ones. How the subdivision is accomplished is a matter of strategy, and represents a characteristic feature of the different visibility-test procedures. In any case, computer science techniques for sorting and searching are an indispensable ingredient of all these methods. The two combined point/surface test procedures described in Sect. 6.1.2 are typical examples of the subdivision principle. The principle has been investigated in greater detail in [CATM78], [COHE80], [DOO__78].

6.1.3.4 3D Solid Modeling

Representation schemes for 3D objects (often called "solids") should satisfy the following criteria:

- validity:
  We require that there exists a real 3D object corresponding to any given representation. A single line dangling in space, or M.C. Escher's famous drawings of impossible objects, may illustrate the notion of validity by counterexample;
- completeness:
  We require that all the operations we provide in a system be applicable to all possible representations of solids within the schema used. For instance, if hidden-line removal could be applied to convex solids but not to non-convex ones, we would consider the scheme as incomplete;

- uniqueness:
  We require that there exists *only one* 3D object corresponding to any given representation. Wire-frame representations may be ambiguous, and hence are not unique;
- conciseness:
  The schema for representation of solids should not contain redundant information;
- ease of creation and modification:
  In order to minimize the computational effort during interactions with the 3D model, the internal representation should be as close as possible to the mental schema that the operator prefers when building or modifying a solid or an assembly of solids;
- efficiency:
  The efficiency of the algorithms operating on the internal representation of solids depends significantly on that internal representation. Different representations may be better suited for different algorithms. Hence, it may be advantageous to jeopardize the principle of conciseness for the sake of greater efficiency, and to maintain some redundancy in the data model.

The elementary operations which we expect to be available for 3D models are:

- to build a model;
- to modify a model;
- to generate a projective display for area-drawing hardware (gray-scale or colour) with hidden surfaces removed;
- to identify objects (points, edges, faces, volume elements) in the 3D model by pointing to their 2D representations within a display;
- to evaluate collisions between separate solids; and
- to compute geometrical and inertial properties (surface area, volume, mass, center of mass, moments of inertia).

Other operations are desired in various applications:

- generation of manufacturing information;
- treatment of imprecise dimensions (tolerances);
- generation of shadows produced by various light sources;
- representation of light reflections on the surfaces; and
- treatment of translucent solids.

A number of commercial solid modeling systems are listed in Table 6.1. Three-dimensional modeling systems may be distinguished in various ways. Their fundamental characteristics are:

- the representation schema for solids;
- the user functions for building solid models;
- the types of surfaces allowed; and
- the language facilities provided for the user to formulate the modeling operations.

Two representation schemes are commonly used in 3D systems:

- the boundary representation (B-rep) scheme; and
- the constructive solid geometry (CSG) model.

Table 6.1. Commercial solid modeling systems

Solid modeling system	Supplier	Country	Nucleus	Type of modeler
ANVIL 5000/ OMNISOLIDS	MCS	USA		CSG/B-REP
BRAVO 3 (Solids Modeler)	Applicon	USA	Synthavision	CSG/B-REP
SOLIDDESIGN (CADDS4)	Computervision	USA		B-REP
CADIS-3D	Siemens	W. Germany	Romulus	B-REP
CAEDS	IBM	USA	Geomod	B-REP
CAM-X 3D	Ferranti	United Kingdom	Romulus	B-REP
CATIA SGM	IBM	USA	(Dassault-Syst.)	B-REP
CIMPLEX-DESIGN	ATP	USA		CSG/B-REP
CONCAD 2	Contraves	Switzerland	Romulus	B-REP
EUCLID	Matra Datavision	France	(CNRS)	CSG/B-REP
GEOMOD	General Electric	USA	(SDRC)	B-REP
ICEM	Control Data	USA	Synthavision	CSG
CIS-MEDUSA 3D	Computervision	USA		B-REP
MEDUSA	Prime	USA	CIS-MEDUSA	B-REP
ME Serie 30	Hewlett Packard	W. Germany	Romulus	B-REP
PATRAN	PDA Engineering	USA		HPAT/B-REP
PROREN 2	Isykon	W. Germany		B-REP
SOLIDS ENGINE (INSIGHT)	Phoenix Data Sys.	USA		OCTREE
STRIM TV	Cisigraph	France		B-REP
TECH 3D	Norsk Data	Norway	Compac	B-REP
TIPS-1	CAM-1	USA	University Hokkaido	CSG,CELL-D.
UNISOLIDS	McD. Douglas	USA	Public PADL-2	CSG

The boundary representation scheme is more familiar to the user who has previous experience with wire-frame models. Each solid is defined by its boundaries. Each boundary is a (planar or sculptured) surface bounded by edges of an adjacent boundary. Three or more edges join at nodes. The boundary representation is best suited for generating projective views, as the important elements for this operation (edges and faces) are readily available in the solid representation. Geometric and inertial properties may be computed from a boundary representation by means of Gaussian integration.

The constructive solid geometry representation is based on a two-level scheme. On the lower level, bounded volume primitives are defined on the basis of half-spaces (one half-space for a sphere, three for a circular cylinder, six for a square block). In

Table 6.2. Modeling concepts of selected commercial 3D modeling systems

System	Modeling type
Bravo3	CSG (facetted B-rep for interactive use)
Catia	B-rep
Cimplex	Hybrid (CSG + B-rep)
Euclid	Hybrid (CSG + facetted B-rep)
Geomod	B-rep
Icem	CSG (based on Synthavision)
Medusa	B-rep
Patran	Hyperpatches
Proren 2	B-rep
Romulus	B-rep
Series 7000	CSG (based on PADL-2)
Solids	Hybrid (CSG + B-rep)
Soliddesign	B-rep
Synthavision	CSG
Technovision	B-rep (based on COMPAC)
Unisolids	CSG (based on PADL-2)

simple cases, such as when only rectangular blocks are used, the half-spaces may be defined by parameters (like position, orientation, and size) associated with the volume primitive, rather than being represented explicitly in the schema. On the second level, these primitives are combined by Boolean set operators (union, intersection, difference). More precisely, we have to use the regularized form of these set operators [REQU82], [TILO80]. Figures 6.13a through 6.13e illustrate the effects of particular operators. The principle advantage of CSG is that it guarantees the validity and uniqueness of the model: a boundary representation can always be derived in a unique way.

The boundary representation is not suited for input. Most systems (see Table 6.1) offer volume elements or half-spaces together with the above mentioned set operations for formulating the model, in a way that is consistent with the internal CSG representation.

Another important technique for defining a solid is the sweeping operation. In the most general case, a bounded surface element is moved along an arbitrary trajectory in space. In most practical cases, the sweeping operation corresponds to a translation along a straight line, or to a rotation. These sweeping operations are particularly useful for modeling manufacturing processes.

Solid modeling systems generally allow one of the following classes of surfaces:

- Planes:
  although most industrial products have curved surfaces, solid modeling based on planar surfaces has a wide range of possible applications. Finite element analysis of a 3D object is generally based on a model, which represents the 3D object by a large number of small blocks (pyramids, for instance), thus approximating the surface in terms of numerous planes. The same principle of approximation may

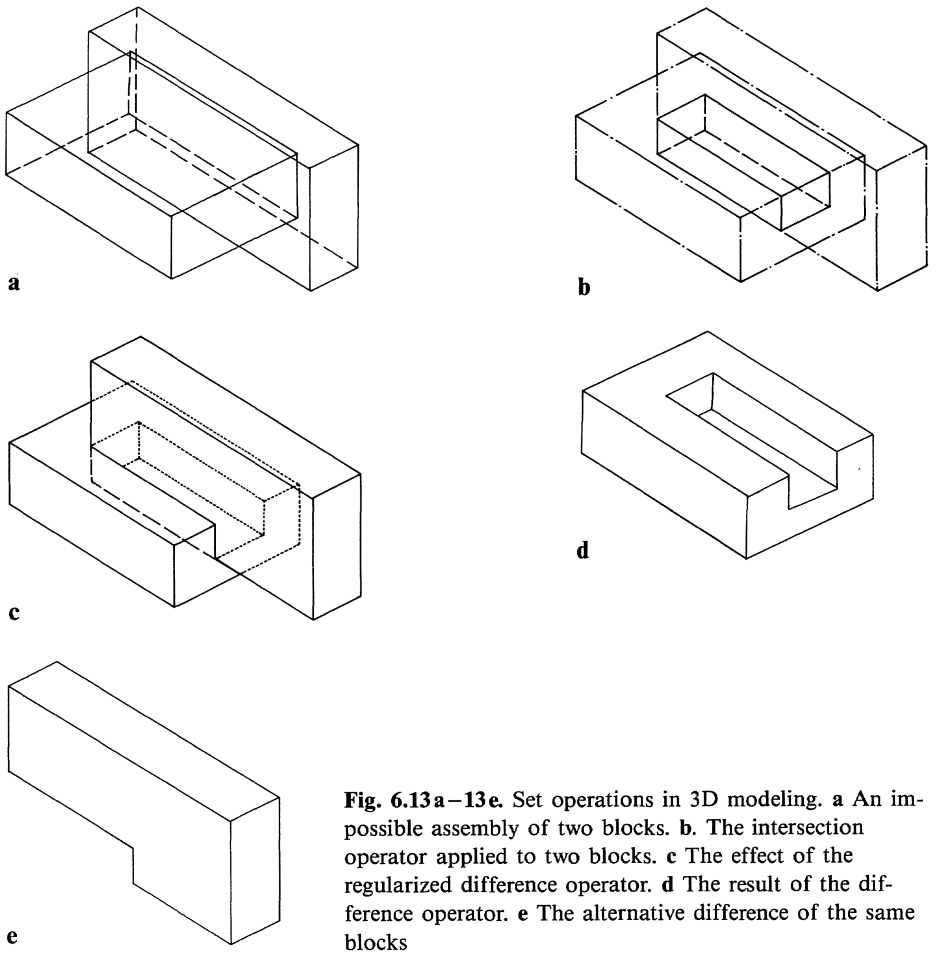


Fig. 6.13a–13e. Set operations in 3D modeling. **a** An impossible assembly of two blocks. **b.** The intersection operator applied to two blocks. **c** The effect of the regularized difference operator. **d** The result of the difference operator. **e** The alternative difference of the same blocks

be generalized to other applications: display with hidden-line or hidden-surface removal, or geometric and inertial analysis. Such approximations may require hundreds or thousands of flat patches to be treated. However, the algorithms for treating planar surfaces are simple, well known, and fast. Some care must be taken not to show those lines between patches that result from the method of approximation but that do not exist in reality.

- Quadric surfaces:
  quadric surfaces are defined by a second-order polynomial in the three spatial coordinates. Quadric surfaces are very popular in solid modeling systems. Most systems, however, do not treat the general quadric surface, but are restricted in some way. Spheres, cylinders, and cones (and planes, of course) are the common quadrics. Some systems provide the facility to handle toroidal surfaces as well. The need for treating the torus arises from the many practical occurrences of this shape.

- Sculptured surfaces defined on a u-v grid:
  sculptured surfaces, as the kind popular in surface-handling systems (B-spline patches, for instance), so far play an "outsider" role in solid modeling. The algorithms for performing the required operations in 3D modeling have not yet been developed to a state of satisfactory completeness and efficiency. In particular, the combination of solids defined by different types of surfaces (some by quadrics, others by patches) is an as yet unresolved problem.
- Super-quadric surfaces:
  the theory of super-quadrics has made some progress [BARR81]. Super-quadrics are generalizations of quadrics. For simplicity, we can use a two-dimensional space to illustrate this generalization.
  While

  $$x = a \cos \delta;\ y = b \sin \delta$$

  is the parametric representation of an ellipse, the superellipse

  $$x = a \cos^p \delta;\ y = b \sin^p \delta$$

  represents a wide range of two-dimensional shapes, from a slightly rounded rectangle (with sides a and b) to an image which looks like an "X" (or two thin sticks of lengths a and b crossing each other perpendicularly at their mid points). Superquadrics can describe solids of relatively complicated shape by means of very few surfaces, at least in an approximate way. They also have the potential to describe slightly rounded edges on an otherwise rather square body. Superquadrics have not found their way into practice.
- Fillets and chamfers:
  Most technical objects have edges that deviate only slightly from the ideal mathematical shape. Fillets and chamfers are typical examples of such deviations. Two approaches may be taken to handle these features: the "correct" representation based on the same techniques as the overall solid model (boundary representation or constructive solid geometry) or the approximate representation as an attribute associated with the edges of a boundary representation. In the latter case, these deviations from ideal geometry cannot be treated by the same overall algorithms. They can be used to modify the displayed picture locally in a manner suitable for perception by a human. But their treatment in hidden-line or hidden-surface algorithms, as well as in geometric or inertial computations, is incomplete. Efficiency considerations nevertheless call for such a simplified special treatment of minor local modifications of the geometry.

Historically, solid modeling systems were first oriented towards batch data processing. This was due to the large amount of storage and computer time required for analyzing 3D models. Consequently, the original language interface for the user was either a package of subroutines or a command or programming language. Even today, 3D model analysis is typically a batch job requiring several minutes of fast processor time for non-trivial problems. For building the model, however, graphic interaction is becoming attractive. Hidden-line removal is generally suppressed during model building. The resulting picture in the building phase very much resembles a wire-frame model, showing all (hidden and non-hidden) lines; but the data structure being built represents the correct 3D model.

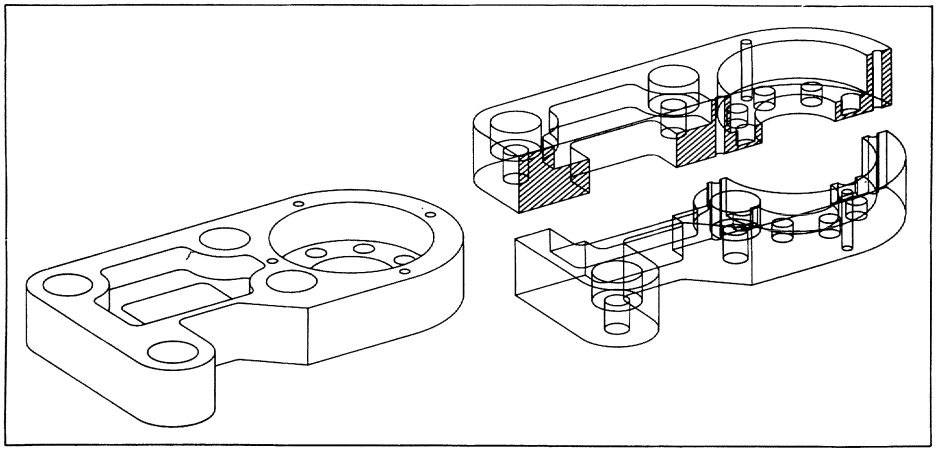


Fig. 6.14. A perspective view of a 3D solid model (line drawing)

The computer graphics literature gives examples of computer generated pictures of three-dimensional objects, that approach the quality of color photographs. Light source reflections on the surfaces, translucency, and shadows significantly contribute to this effect. Such pictures may be generated on color raster display devices using the "ray-casting" technique. With this technique, an area of the projection plane is divided into a number of pixels (picture elements); lines of sight are followed from the anticipated position of the eye to each one of these pixels; the point of intersection of this line of sight with the first surface is then analyzed by continuing along the lines to all light sources; any light source that is not hidden by interfering other surfaces will contribute to the intensity and color of this visible point. Similarly, lines of sight are followed through translucent material. For high resolution images (for example, 1024 by 1024 pixels) the computational effort is generally beyond that which can be justified in a CAD environment.

Figures 6.14 (courtesy of Ferranti Cetec Graphics, Ltd.) and 6.15 (courtesy of Applicon) show two examples of projective representations of solid models. Other examples will be given in Chap. 8.

6.1.3.5 Mathematical Description of Curves and Surfaces

The principle of the mathematical description of curves and surfaces is an instruction for generating these objects from a rather small set of data values. Such instructions require only little storage space and provide the additional benefit that important properties like tangents and curvatures may be derived from these data values exactly by applying rigorous mathematical formulas.

Curves and surfaces are represented mainly in parametric form as so-called *parametric curves* or *parametric surfaces*:

$$\mathbf{C}(u) = \begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix}$$

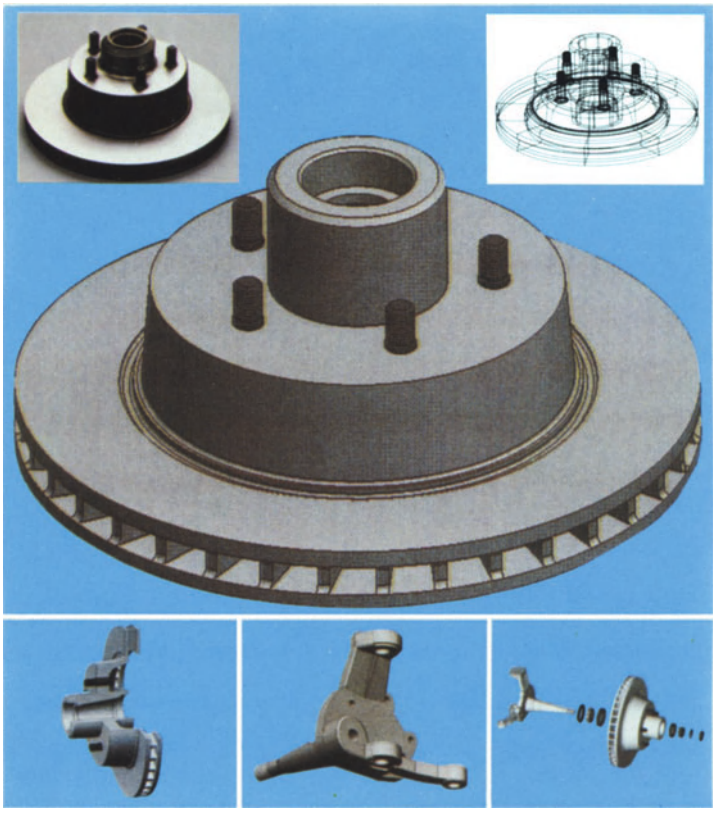


Fig. 6.15. A perspective of a 3D solid model (color raster graphics, shaded picture)

$$\mathbf{S}(u, v) = \begin{bmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{bmatrix}$$

where **C** and **S** are the two-dimensional or three-dimensional point vectors defining the curve or surface, respectively, in terms of the vector coorrdinates x, y, and z (the latter for three-dimensional objects only) while the parameters u (and v, for surfaces) vary continuously over a certain range of values. The shape of these curves and surfaces is controlled by certain characteristic *control points* **P**.

This representation permits the unambiguous and compact definition of multivalued objects like spiraling curves or toroidal surfaces. The transformation of parametric curves and surfaces is easily achieved by transforming the control points. The parametrization is, however, not unique; different parametric representations may define the same geometric object (the parameter values associated with each point on the object would of course be different for each parametric representation).

The parametric descriptions of curves and surfaces are based on control points. In general, each control point is multiplied by a weighting function which depends on the parameters. This weighting function determines the influence of the related

control point for each parameter value. The sum of these weighted control point vectors produces the final point:

$$\mathbf{C}(u) = \sum_{i=0}^{n} \mathbf{P_i} \cdot W_i(u)$$

with the control points

$$\mathbf{P_i} = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}$$

and the weighting functions $W_i(u)$;

$$\mathbf{S}(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{P_{ij}} \cdot W_i(u) \cdot V_j(v)$$

with the control points

$$\mathbf{P_{ij}} = \begin{bmatrix} x_{ij} \\ y_{ij} \\ z_{ij} \end{bmatrix}$$

and the weighting functions $W_i(u)$ and $V_j(v)$.

This kind of weighting function for surface definition consists of a product of the separate weighting functions in u and v.

The definition of the weighting functions determines the shape and the behavior of the resulting curve or surface. Two basic goals of parametric curve and surface representation may be distinguished: interpolation and approximation. In the first case, the curve (or surface) will pass through the control points, in the latter case it will not do this. We now present a brief description of some important classes of parametric representations for curves. This can be extended easily to surfaces by using the multiplication method for the weighting functions.

Lagrange interpolation

$$\mathbf{C}(u) = \sum_{i=0}^{n} \mathbf{P_i} \cdot L_{i,n}(u) \quad \text{with } 0 \leqq u \leqq 1$$

$$\text{with } L_{i,n}(u) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{u \cdot n - k}{i - k}$$

The behavior of these curves is as follows:

1) The control points P_i are interpolated with parameter $u = i/n$
2) Every control point has global control upon the curve: changing one control point influences the whole curve.
3) The degree of the curve is n. The curve tends to oscillate for increasing n.
4) The concatenation of two Lagrange curves has, in general, zero-order continuity.

Cubic-Hermite interpolation

$$\mathbf{C}(u) = \sum_{i=0}^{1} \mathbf{P}_i \cdot H_i(u) + \sum_{l=0}^{1} \mathbf{T}_l \cdot H_l(u) \quad \text{with} \quad 0 \leqq u \leqq 1$$

This scheme takes tangential conditions into account. The tangents are $\mathbf{T}_0$ at $\mathbf{P}_0$ and $\mathbf{T}_1$ at $\mathbf{P}_1$. The weighting functions are:

$$H_0(u) = (1+2u)\cdot(u-1)^2$$
$$H_1(u) = (3-2u)\cdot u^2$$
$$U_0(u) = u\cdot(u-1)^2$$
$$U_1(u) = (u-1)\cdot u^2$$

The behavior of these curves is as follows:

1) The control points $\mathbf{P}_0$ and $\mathbf{P}_1$ are interpolated.
2) The tangents are interpolated as well; hence, we have a smooth change between the two control points.
3) The concatention of two Cubic-Hermite curves has, in general, first-order continuity.

Bézier approximation

$$\mathbf{C}(u) = \sum_{i=0}^{n} \mathbf{P}_i \cdot B_{i,n}(u) \quad \text{with} \quad 0 \leqq u \leqq 1$$

$$\text{with } B_{i,n}(u) = \binom{n}{i} \cdot (1-u)^{n-i} \cdot u^i \quad \text{(Bernstein base functions)}$$

The behaviour of these curves is as follows:

1) For every particular parameter value u of the domain the resulting point is a convex combination of the control points. The curve, therefore, lies entirely within the convex hull of the control points.
2) The first and last control points are interpolated.
3) The tangent vectors of the curve in the first and last control points have the direction of the related polygon segment (the direction from the end points to their immediate neighbours).
4) Two Bézier curves may be concatenated with first-order continuity.
5) Every control point has global control (influences the entire curve).
6) The degree of the curve is n and the curve is variation diminishing.
7) The curve could be evaluated recursively with the help of the algorithm of de Casteljau.

Bspline approximation

This class of curves possesses great flexibility in the definition of the weighting functions. A vector of parameter values u (called the knot vector) (u_0, u_1, ... u_l) with

$u_i \leqq u_{i+1}$ is used for the recursive definition of the weighting functions. The length and the structure of the knot vector depend on the desired behavior of the resulting curve.

$$\mathbf{C}(u) = \sum_{i=0}^{n} \mathbf{P_i} \cdot N_i^r(u) \quad \text{with} \quad u_0 \leqq u \leqq u_l$$

with

$N_i^0(u) = 1$, if $u \in [u_i, u_{i+1}]$

$N_i^0(u) = 0$, otherwise

$$N_i^r(u) = \frac{u - u_i}{u_{i+r} - u_i} \cdot N_i^{r-1}(u) + \frac{u_{i+r+1} - u}{u_{i+r+1} - u_{i+1}} N_{i+1}^{r-1}$$

The behavior of these curves is as follows:

1) The parameter domain is determined by the first and last knot.
2) The parameter range between two adjacent knots generates one segment of the curve.
3) The degree of the curve is independent of the number n of control points and the curve is variation diminishing.
4) Every control point has local control upon the curve (influences only its neighbourhood).
5) For every particular parameter value of the domain the resulting point of the curve is a convex combination of the affected control points for the segment $u_i \leqq u \leqq u_{i+1}$. Each segment of the curve lies, therefore, within the convex hull of its related set of control points.
6) The task of concatenating two Bsplines is reduced to the problem of defining a proper knot vector for the joint set of control points.
7) The knot vector consisting of k times 0 and k times 1 defines weighting functions which are identical to the Bernstein weighting functions of degree $(k-1)$ for a Bézier curve.

Approximation with rational curves

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} \mathbf{P_i} \cdot R_i(u)}{\sum_{i=0}^{n} w_i \cdot R_i(u)}$$

where the $R_i(u)$ are polynomial functions in u.

Depending on the chosen functional basis for all of the polynominals as either monomial, Bernstein, or Bspline the resulting curve is called a rational, rational Bézier, or rational Bspline curve.

The general behavior of these curves is as follows:

1) The common denominator $\sum_{i=0}^{n} w_i \cdot R_i(u)$ could be interpreted as

division by a homogeneous coordinate (see 6.1.1). Therefore, the curve could be treated as a homogeneous four-dimensional non-rational curve:

$$\mathbf{C}(u) = \sum_{i=0}^{n} \begin{bmatrix} \mathbf{P}_i \\ w_i \end{bmatrix} \cdot R_i(u)$$

This also fits with the transformation schema which is in fact a multiplication of the homogeneous coordinate vector by a 4×4 matrix.

2) The set of exactly describable curves grows with this very powerful method. A non-degenerate conic (ellipse, parabola, or hyperbola), for instance, could be represented as a rational curve of second degree.
As an example, let us investigate a rational representation of a circle:

$$\mathbf{C}(u) = \begin{bmatrix} 1-u^2 \\ 2u \\ 0 \\ 1+u^2 \end{bmatrix}$$

$\mathbf{C}(u)$ in fact represents a circle which is proved as follows:

$$x(u)^2 + y(u)^2 + z(u)^2 = \frac{(1-u^2)^2}{(1+u^2)^2} + \frac{(2u)^2}{(1+u^2)^2} + 0 = 1$$

For more detailed information see references [BOEH84], [KLEM86], [FORR68], [FAUX83], [TILL83], and [GROS57].

6.2 Numerical Methods

6.2.1 Introduction

In the late 1960s and especially in the early 1970s, a tremendous development of numerical methods for design applications took place. This rapid development resulted from a feedback process involving both computer technology and engineering sciences:

- computers became increasingly powerful, and cheaper as well. Thus methods which required many computations and a large memory could now be applied at moderate expense, while in previous decades hundreds of man years would have been required to do the same computations manually; and
- the successful application of numerical methods led to intensive research and the goal of making more and bigger problems tractable by these methods.

Numerical methods have had their greatest impact on the analysis part of the design process. The most widely known group of methods are the *finite element methods*; they are primarly used to determine stresses and deformations in structural components for prescribed load cases. *Finite difference methods* play a dominant role in determining the forces which are exerted by fluids (gases or liquids) upon structures. Many methods have been developed for the *simulation* of dynamic processes (both continuous and discontinuous). *Optimization methods* may to some extent

replace human judgment in the analysis-synthesis-evaluation loop of the design process. So far, however, their applicability is restricted mainly to variational design (see Fig. 3.13). Whether deterministic optimization methods (as opposed to artificial intelligence methods) are suited to play a major role in the synthesis process (contributing to the design of a schema rather than determining optimal values of attributes in the schema) is still an open question.

The progress achieved in many industries (such as aerospace, nuclear, electronics, and armaments) would not have been possible without the integration of numerical methods into the design process. However, the problems increased along with the computing power. For many problems, even today's computers are too small and too slow.

6.2.2 Finite Element Methods

Finite element methods (FEM) are most widely applied in structural analysis, although applications in fluid flow and thermal analysis are also quite successful. A tremendous literature on finite element methods is available. One of the fundamental sources is [ZIEN77]. It is not our intention to provide a thorough introduction to the numerical method of finite elements. In this respect, the reader should study the relevant literature, such as [GLOW79], [GALL75], [PILK74]; instead, we will concentrate on the question of embedding finite element analysis methods into the CAD process. The importance of finite element methods for CAD stems from the fact that a number of general-purpose finite element programs has become commercially available, and a large community of engineers has obtained the necessary expertise to apply these programs to their problems. Detailed knowledge of the finite element theory and the associated mathematical methods is not required for most practical applications.

In order to imbed finite element programs properly into the design process, the availability of such a program alone is not sufficient. Figure 6.16 shows a family of programs, which is a prerequisite for successful applications.

- As a first step, based upon the geometrical definition of the object to be analyzed, a finite element mesh must be generated. For small problems this mesh may be prepared manually; but for problems with more than a hundred elements or so, automatic mesh generation by a (usually batch) program is advised.
- In most cases, inspection of the generated mesh by experienced users will indicate that the first mesh is not completely satisfactory. The user may want to refine the mesh in certain areas to improve the resolution and accuracy of the analysis; he may want to make the mesh coarser in irrelevant areas for reduction of the computer costs, or he may want to add geometrical details which are significant for the structural analysis but could not be derived automatically from the geometrical data in the primary data base. This modification of the finite element mesh is best done at an interactive graphics terminal.
- Material data and data describing the load cases must then be added to the finite element program data base.
- Finite element programs usually require significantly more computer resources than their input generators and post-processors. They are typically large batch

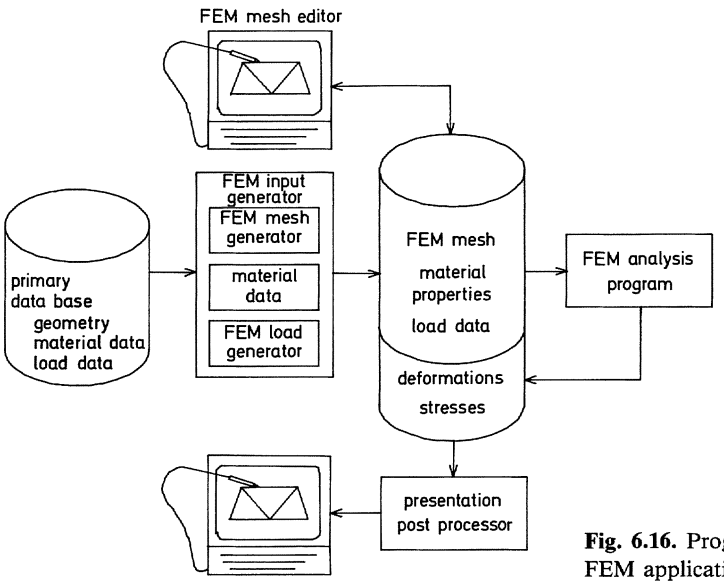


Fig. 6.16. Program structure for FEM applications in CAD

programs which may consume hours of computer time, depending upon problem size and complexity. It is common practice to have the input generators and the post-processor near the design engineer and to submit the finite element analysis job to a large remote computer (see Sect. 4.3.4).

- Presentation of the results for complex structures is in itself a nontrivial task. For interpretation and evaluation of the results, the user may want to see certain parts of the structure from various view points, with the resulting deformations and stress representations superimposed. Here again, interactive graphics has become a fruitful tool.

Quite often, the interpretation of the result leads to the need for further analysis of the same structure with a modified mesh and/or modified load case, in a repeated application of the above steps, until the results are considered reliable. If the results are unsatisfactory in the light of the specification, another design iteration will be required, which is likely to produce changes in the geometrical data or material data of the primary data base. The finite element analysis will have to be repeated, starting either from a new input generation or perhaps from a modification of the obsolete finite element data base. The latter procedure may be cheaper, but it introduces the risk of inconsistencies, along with the problems of data validity which were described in Sect. 3.3.5.1.

Programs for generation of the primary input are usually closely related to the objects to be analyzed. They are tailored to the particular application and to the schemas used to represent geometry, loads, and material information in the primary data base. They are generally not portable to other design objects or companies. Programs for interactive mesh modification (which in medium-size problems may also be used for mesh generation from scratch) and presentation post-processors are commercially available for a number of finite element programs. Quite a few of them provide interfaces for more than one such program.

Table 6.3. Comparison of general-purpose structural mechanics programs

KEY:
+ = YES
- = NO
U = undefined

PROGRAM NAME	DISCRETIZATION METHOD: DISPLACEMENT	HYBRID	MIXED	FE-FOURIER SERIES	FINITE DIFFERENCE	POINT MATCHING	FORWARD INTEGRATION	OTHER	TYPES OF GEOMETRY: 1 DIMENSIONAL	2 DIMENSIONAL	1.1 2 DIM. CURVED	3 DIMENSIONAL	LOADING CASES: STATIC	FREE MODES	FORCED MOTION	TRANSIENT (GENERAL)	STATIONARY TEMP.-FIELDS	INSTAT. TEMP.-FIELDS	TYPES OF MATERIALS: ISOTROPIC	ANISOTROPIC	LAYERED	SANDWICH	TEMP.-DEPENDENT	NONLIN. ELASTIC	RIGID-PLASTIC	ELAST.-STRAIN HARD.	OTHER TYPES	LARGE DEFLECTIONS	COMPUTER REQUIREM.: IN-CORE PROGRAM	MIN. MEMORY SIZE (K WORDS)	NO. OF DISK-FILES	NO. OF TAPES	RANDOM ACCESS	SOFTWARE ASPECTS: PROGRAM LANGUAGE	PREPROCESSOR	POSTPROCESSOR	PLOT ROUTINES	INTERACTIVE MODES	RESTART CAPABILITY	ERROR DIAGNOSTIC	SUBSTRUCTURING	DISTR. BY DEVELOPER	DISTR. BY SOFTWARE CENTER
1 AC50-A	U	U	U	U	U	U	U	U	-	+	-	-	+	-	-	-	+	-	+	+	+	+	-	-	-	-	-	-	+	20	-	-	-	F4	-	-	-	-	-	-	-	+	-
2 ADEPT	-	-	-	-	-	-	+	-	+	-	-	-	+	-	-	-	-	-	+	U	-	-	-	-	+	-	-	-	-	76	-	-	-	AT	-	-	-	-	+	+	+	U	U
3 AMSA 20	+	-	-	-	-	-	-	-	+	+	-	+	+	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	35	U	U	U	F5	-	-	-	-	-	-	+	U	-
4 ANSYS	+	-	-	+	-	-	+	-	+	+	+	+	+	+	+	+	+	U	+	+	+	+	+	-	-	+	+	+	-	50	1	-	+	F	+	+	+	+	+	+	+	+	-
5 ASAS	+	-	-	-	-	-	-	-	+	+	-	+	+	-	-	-	-	-	+	+	-	-	-	+	-	+	+	+	-	32	U	U	U	F	-	-	-	-	+	+	-	+	-
6 ASKA	+	-	-	+	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+	+	-	+	-	+	+	+	+	+	-	32	1	U	+	F4	+	+	+	-	+	+	+	+	+
7 BASY	+	+	-	-	-	-	-	-	+	+	+	+	+	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	16	U	U	U	F	-	-	-	-	+	+	-	+	-
8 BERSAFE	+	-	-	-	-	-	-	-	+	+	+	+	+	-	-	-	-	+	+	-	-	-	+	-	-	+	+	+	-	99	9	+	+	F4	+	+	+	-	+	-	-	+	-
9 BOSOR 4	-	-	-	-	+	-	-	-	-	-	+	+	+	+	-	+	-	-	+	+	+	+	+	-	-	-	-	-	-	64	1	-	+	F4	-	-	+	-	-	+	-	+	+
10 COSA	+	-	-	-	-	-	-	-	+	+	-	+	+	+	+	-	-	-	+	+	-	+	-	-	-	-	-	+	-	64	1	-	+	F	-	-	-	-	+	+	+	+	-
11 DYNAS	U	U	U	U	U	U	U	U	+	+	-	-	+	+	+	+	-	-	+	-	-	-	-	-	-	-	-	U	-	80	8	-	+	F	-	-	+	-	+	+	+	+	-
12 EASE 2	+	-	-	-	-	-	-	-	+	+	+	+	+	-	-	-	+	-	+	-	-	-	-	-	-	-	-	-	-	30	1	-	+	F/A	+	-	+	-	+	+	+	-	+
13 ELAS 75	+	-	-	-	-	-	-	-	+	+	+	+	+	-	-	-	+	-	+	+	+	+	-	-	-	-	-	-	-	20	-	-	U	F4	-	+	+	-	-	+	-	-	+
14 FARSS	+	-	-	-	+	-	-	-	-	-	+	+	+	+	+	+	-	-	+	+	-	-	+	-	-	-	-	-	-	20	1	-	+	F4	+	+	+	+	+	+	-	+	-
15 FESAP	+	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	-	-	+	+	-	-	+	-	-	-	-	-	-	20	8	-	1	F4	+	+	+	-	-	+	-	-	-
16 FLHE	+	-	-	-	-	-	-	-	+	+	-	+	-	-	-	-	+	+	+	-	-	-	-	-	-	-	-	-	-	60	U	U	U	F	-	-	-	-	-	+	U	+	-
17 ISOPAR SHL	+	-	-	-	-	-	-	-	-	+	+	+	+	-	-	-	-	-	+	+	+	+	-	-	-	-	-	-	-	65	9	-	+	F	+	-	-	-	-	+	-	+	-
18 ISTRAN/S	+	-	-	-	-	-	-	-	+	+	+	-	+	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	+	64	20	-	+	F/A	+	-	+	-	+	+	-	+	-
19 KSHEL	-	-	-	-	-	-	+	-	-	-	+	-	+	+	+	+	-	-	+	+	+	+	+	+	-	-	-	+	-	33	-	+	-	F	-	-	-	-	-	+	-	+	-
20 MARC	+	-	-	-	-	-	-	-	+	+	+	+	+	-	-	+	-	-	+	-	+	-	-	-	-	+	-	+	-	60	U	U	U	F	+	-	-	-	+	U	-	-	+
21 MINIELAS	+	-	-	-	-	-	-	-	+	+	+	-	+	+	+	-	+	-	+	+	+	+	-	-	-	-	-	+	-	32	-	-	-	F4	-	-	-	-	-	-	+	-	-
22 NASTRAN	+	-	-	-	-	-	-	-	+	+	+	+	+	+	-	+	+	-	+	-	+	+	-	-	+	-	+	-	-	50	U	-	+	F	+	+	+	-	+	+	+	-	+
23 NEPSAP	+	-	-	-	-	-	-	-	+	+	+	+	+	-	-	-	-	-	+	+	+	+	+	-	-	+	-	+	-	45	12	+	+	F	+	+	+	-	+	+	-	+	+
24 NONLIN 2	+	-	-	-	-	-	-	-	+	-	-	-	+	-	+	-	-	-	+	-	-	-	-	-	-	+	-	+	-	57	1	-	+	U	-	-	-	-	-	-	+	-	-
25 NONSAP	+	-	-	-	-	-	-	-	+	+	+	-	+	+	-	+	-	-	+	+	-	+	+	+	+	-	+	+	-	128	-	-	-	F	-	-	-	-	+	-	-	+	-
26 NOSTRA	+	-	-	-	-	-	-	-	+	+	+	+	+	-	-	-	-	+	+	-	+	-	-	-	-	-	-	+	-	28	6	-	+	F	+	+	+	+	+	+	-	U	U
27 PAFEC 70	+	+	-	-	-	-	-	-	+	+	+	+	+	+	-	-	+	-	+	+	+	-	-	-	-	-	-	-	-	10	1	-	+	F	+	+	+	-	-	-	-	+	-
28 FRAKSI	+	+	-	-	-	-	-	-	+	+	-	-	+	-	-	-	-	-	+	+	-	-	-	-	-	-	-	+	-	33	4	-	+	F4	-	-	+	-	-	+	+	-	-
29 REXBAT	+	-	-	-	-	-	-	-	+	+	-	+	+	-	-	-	+	-	+	+	-	-	+	-	-	-	-	-	-	65	10	-	+	F	+	+	+	-	+	+	+	+	-
30 SABOR/DRASTIC6	+	-	-	+	-	-	-	-	-	+	-	+	+	-	+	+	-	-	+	+	+	-	-	+	+	+	-	+	+	45	13	1	+	F	-	+	+	-	+	+	-	+	-
31 SAMBA	+	-	+	-	-	-	-	-	+	+	-	-	+	-	-	+	-	-	+	+	+	-	-	-	-	-	-	+	-	24	1	-	+	AL	-	-	+	-	+	+	+	+	-
32 SAP IV	+	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	-	+	+	+	+	+	-	-	-	-	-	-	18	18	-	+	F4	-	-	-	-	+	+	-	-	+
33 SATANS	-	-	-	+	-	-	-	-	-	-	+	-	+	-	+	+	+	-	+	-	-	+	-	-	-	-	-	-	+	36	-	-	-	F4	-	-	-	-	+	-	-	-	+
34 SESAM 69	+	-	-	-	-	-	-	-	+	+	+	+	+	+	-	-	-	-	+	+	-	-	-	-	-	-	-	-	-	40	+	+	+	F4	-	-	-	-	-	-	+	+	-
35 SHORE	-	-	-	-	+	-	-	-	-	-	+	-	+	-	+	-	-	+	+	-	-	+	-	-	+	-	-	-	-	62	1	+	+	F5	-	-	+	+	+	-	+	-	-
36 STARDYNE	+	-	-	-	-	-	-	-	+	+	-	+	+	+	+	+	+	-	+	+	+	+	-	-	-	-	-	-	-	32	4	+	+	F	+	+	+	-	+	+	+	-	+
37 STARS	-	-	-	-	+	-	-	-	+	-	+	-	+	+	+	+	+	+	+	+	+	+	+	+	-	+	-	+	-	U	U	U	U	F	+	-	+	-	+	+	-	U	U
38 STRIP	-	+	-	-	-	-	-	-	+	+	+	-	+	-	-	-	-	-	+	-	+	-	-	-	-	+	-	+	-	64	+	U	+	F4	+	+	-	-	+	+	+	+	-
39 TEXGAP	+	+	-	+	-	-	-	-	-	+	+	+	+	-	-	+	-	-	+	+	+	-	-	-	-	-	-	-	-	25	9	-	+	F4	+	+	-	+	+	-	+	-	+
40 TIRE	-	+	-	-	-	-	-	-	-	-	+	-	+	-	-	+	-	-	+	+	+	-	-	-	-	-	-	+	-	185	U	U	+	F/A	-	-	-	-	-	-	-	-	+
41 TITUS	+	-	-	-	-	-	-	-	+	+	+	+	+	-	-	-	+	+	+	+	-	-	+	-	-	-	-	-	-	27	U	U	+	F	-	+	+	-	+	+	-	+	-
42 VISCEL	+	-	-	-	-	-	-	-	+	+	+	+	+	-	-	-	+	-	+	+	-	-	+	-	-	-	-	-	-	20	15	-	+	F	+	-	+	-	-	+	-	-	+
43 ZP 26	+	-	-	-	-	-	-	-	-	+	-	+	+	-	-	-	-	-	+	+	-	+	-	-	-	+	-	+	-	98	-	-	-	F4	+	+	+	-	-	+	-	+	-

Many finite element programs were developed for application in a limited environment only. The flexibility of the finite element method, however, led to the development of a market which has been well established since the early 1970s. Some of the finite element programs which have found international distribution are summarized in [FENY73]. Some of them are well known in the CAD community: these include ANSYS [SWAN00], ASKA [SCHR72], NASTRAN [MACN72], MARC [AYRE73], SAP [BATH73], and STRUDL-II [NELS72]. The summary of 43 general-purpose structural mechanics programs shown in Table 6.2 is taken from [RUOF74] and demonstrates the predominance of the finite element method for general-purpose structural mechanics.

In the 1960s and until early in the 1970s, the development of finite element programs was at least partly concentrated in specialized research institutes. Comparative reviews and conferences about recent developments were quite common. Since then, many of the better-known programs are supported by and commercially available from various software houses or computer manufacturers. Reliable and up-to-date information about the actual power of particular finite element programs should be obtained from the organizations that maintain these programs for sale, rent, or remote use (software houses, computer manufacturers, or computer networks).

Flexibility is the advantage of the finite element method. Thus, expertise obtained through the analysis of one type of structure is immediately helpful for other structures. The most widely used type of analysis is static analysis, based on the linear theory of elasticity. Problems with several times ten thousand degrees of freedom are not unusual in static analysis. Dynamic analyses are also quite common. Because these require more computer power, the structural model is generally condensed to a smaller number of degrees of freedom (up to several thousand). Many programs support the analysis of free vibrations, transient response to time-varying loads, and power spectrum analysis for random excitations (earthquake loads, for instance). Because the individual finite element programs do not overlap completely in their capacities and requirements (in terms of analysis capabilities, library of finite element types, users' convenience, computer resource requirements), many organizations employ more than one finite element program.

6.2.3 Finite Difference Methods and Other Methods

As with finite element methods, this book does not intend to initiate the reader into the numerical aspects of finite difference methods. The reader is referred to the extensive and easily accessible literature on this subject. Finite difference methods are generally based on approximate representations of partial differential equations, while finite element methods are commonly derived from integral representations of the problem. Finite difference methods are presently used in more domains than finite elements. They are often related to the determination of loads. Examples of domains of application are:

- fluid dynamics (determination of pressure fields);
- thermodynamics (determination of temperature fields); and
- neutron physics (determination of radiation fields in nuclear reactors).

However, finite element methods have also begun to spread into these domains (see, for instance, [HUGH79]).

The variety of finite difference methods is larger than that of finite element methods. There are several reasons for this. One is the variety in the types of underlying differential equation (the Navier-Stokes equation, the Poisson equation, and the Stefan-Boltzmann equation are used in the above domains). Another reason is the greater dependency of finite difference formulations on geometry. Finite difference programs which are suited to solve fluid dynamics problems in rectangular boxes are not in general applicable to the same problems in cylindrical geometry or in networks of pipes. As a consequence, programs based upon finite difference methods are closely associated with particular objects, and not easily transferable. Thus, no large market has developed like the one for finite element programs. On the other hand, the source code of each finite difference program generally exhibits more similarity to the underlying physical (differential) equation, and is hence more easily understood, modified, and adjusted to other problems. The finite difference approach tends to lead to a relatively large library of small and clearly structured separate programs which are adjusted to actual needs, rather than one big and powerful (finite element) program which should suit many purposes. This approach calls for an environment which provides the necessary software knowledge, and adequate tools for software development and modification (see Sect. 4.1.3.2). "Black box" finite element programs are less demanding in this respect.

The same arguments apply to other methods which have undergone significant progress within the last few years: spectral methods and boundary integral equation methods. The breakthrough in spectral methods came with the invention of the Fast Fourier Transform algorithm [COOL65]. Conventional Fourier transformation requires a number of significant operations (multiplications, for instance) which grows by a factor of n-squared, where n is a figure characterizing the desired resolution. The fast Fourier transformation reduces this to an $n \cdot \log(n)$-dependence. As an example, for doubling the resolution in all directions of a three-dimensional problem, the conventional trensformation would require 64 times the number of operations, while fast Fourier transformation increases the number of operations only by a factor of about 8. For higher resolutions, the difference may easily reach several orders of magnitude [GOTT77].

Boundary integral equation methods [HESS67], [KRIE80] significantly reduce the size of a problem: a three-dimensional problem of fluid flow in a three-dimensional control volume is reduced to a merely two-dimensional problem related to the two-dimensional surface of the control volume. This advantage is offset to a certain extent by the fact that the resulting system of linear equations has a fully populated and irregularly structured matrix of coefficients. For this reason, the boundary integral equation method generally cannot take advantage of the efficient algorithms which have been developed for sparse and cleanly structured (banded) matrices as they arise in conventional finite difference or finite element schemes. Combinations of the two methods with the goal of benefiting from the merits of both have been developed [ZIEN79].

These new methods provide very high computational efficiency, but their range of applicability is sometimes restricted by assumptions regarding the geometry (symmetry assumptions, for instance) or the class of solutions (potential fields). Never-

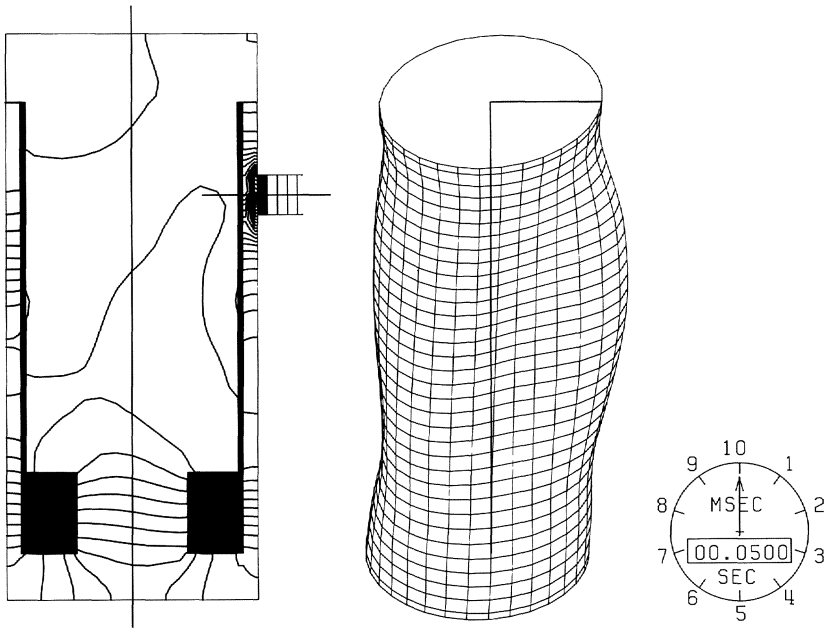


Fig. 6.17. Pressure field in a reactor pressure vessel (left) and deformation of the vessel internals (right, displacement amplified) after a postulated pipe break: the results of a combination of finite difference and spectral techniques

theless, such advanced methods have made problems tractable at moderate cost which would have been too expansive to solve with more conventional finite element or finite difference schemes [SCHU78]. As an example, consider the analysis of a large cylindrical component (diameter: 4.3 m; height: 8.2 m; wall thickness: 80 mm) in a power reactor pressure vessel after the rupture of a large coolant pipe. In this case, the feedback regarding the influence of the structural deformation upon the pressure field must be analyzed concurrently, and superimposed on the pressure field as it results from the primary cause. Forty-four thousand degrees of freedom describing the pressure field and the flow field in the water plus 1000 degrees of freedom describing the structural behavior during the affected time period can be analyzed in a couple of hours. The pressure field and the structural deformation as viewed in such an analysis are shown in Fig. 6.17.

This example indicates a trend which has become more and more important in design analysis. For the first screening investigations in a preliminary design phase, individual aspects – such as static stress considerations – may be considered separately. In detail design, however, functional aspects, structural aspects, economics, manufacturing, and many other aspects will have to be investigated concurrently and with respect to their interactions. In this phase of design analysis, it becomes essential to reduce the amount of overhead required for switching between various analysis programs.

6.2.4 Simulation

6.2.4.1 Survey

"Simulation is a very wide-open and somewhat ill-defined subject of great importance to those responsible for the design of systems as well as those responsible for their operation" [SHAN75]. A similarly broad definition is given in [GORD75]: "We therefore define system simulation as the technique of solving problems by observation of the performance, over time, of a dynamic model of the system". Simulation methods are applied in many areas [HIGH79]: social, ecological, and agricultural systems; military and business applications; health care; manufacturing, marketing, production planning and control; financial systems; computers and computer networks; energy forecasting; and many others. It is surprising that (with exceptions such as in [NOWA80]) the literature on simulation methods does not adequately reflect the importance of simulation in design. In fact, a large portion of the analysis aspect of the design process is devoted to simulation.

In general, simulation methods are divided into two groups:

- discrete simulation;
- continuous simulation.

In addition, combinations of the two in one system have been developed, but are less widely known. The major differences among the various simulation systems are:

- the organization of time and activities;
- the naming and structure of entities;
- the testing conditions for activities;
- the type of statistical tests possible on data; and
- the ease of changing the model structure.

In any case, a "model" of the real system is established, and the changes of the state of this model are observed. Such a model consists of

- a *system*, which is constituted by
- *entities*. Each entity is characterized by a set of
- *attributes*, which describe the state of the entities.
- *Activities, events,* or *processes* may create and delete entities or change their state.

If we combine the terms "system", "entities", and "attributes" into "schema" and replace "activities, events, or processes" by "operation", we note a similarity between the concepts used for simulation and those described in Chap. 3 for databases and design processes. The difference between continuous and discrete simulation lies in the assumptions regarding the continuity of changes of state. Discrete simulation deals with discrete changes of state (creation and deletion of entities, individual and finite changes of attribute values). Typical applications may be found in the simulation of digital systems (computers). Continuous simulation approximates continuous changes of state, as described by a set of ordinary differential equations, by appropriate numerical integration. (A typical example is the behavior of a motor vehicle on a bumpy road). Many simulation problems may be considered as deterministic: in this case the dynamic behavior is fully defined by the model and its initial state. Quite

often, however, the system behavior is influenced by stochastic processes. This applies to both technical applications (roughness of a road, failure rate of mechanical components, work load in a computer network) and non-technical ones (such as those encountered in marketing). Thus, a significant part of the development of simulation methods was devoted to the treatment of stochastic processes.

Simulation can be done completely on a computer, or may integrate real components and people as part of the process. As an example, the dynamic behavior of a car or an aircraft may be simulated with a human driver or pilot involved in a real-time simulation.

The principal difference between simulation as part of the design process and many other simulation applications is that the systems or objects which are simulated for design analysis do not yet exist in reality. They exist only as conceptual models in human brains, and partly as data representations in data bases.

6.2.4.2 Simulation Languages

The schematic overview of simulation languages shown in Fig. 6.18 is based on [SHAN75]. Continuous system simulation was originally the domain of analog computers. Their basic functions (integration, multiplication by a constant, summation,

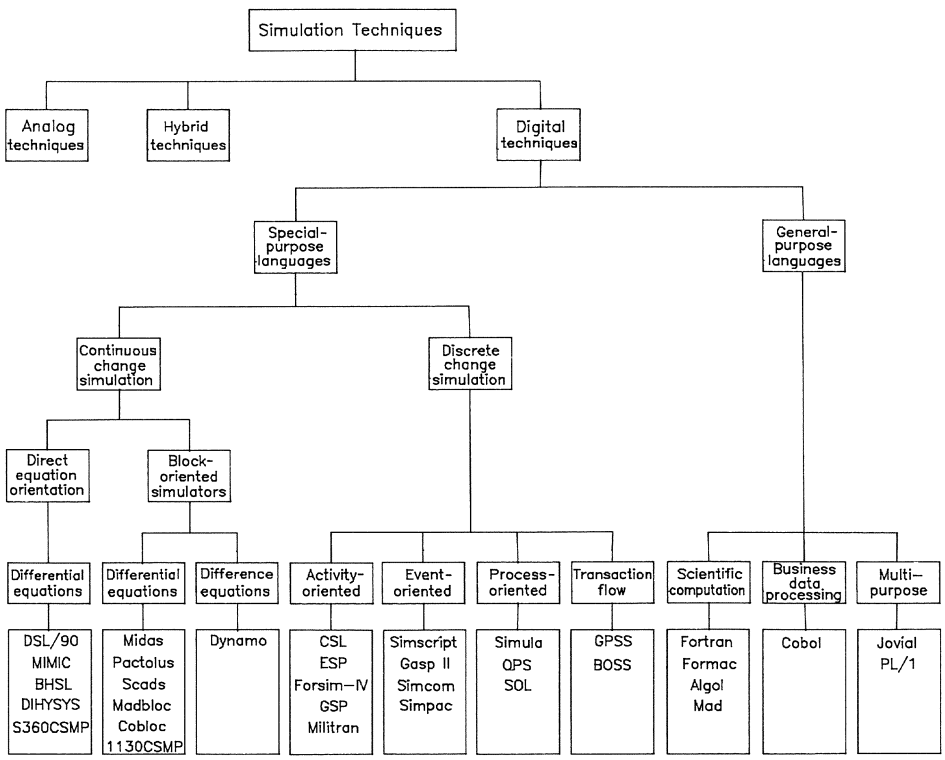


Fig. 6.18. A classification of simulation techniques

and – to some extent – multiplication of variables and generation of functions of a single variable) satisfied many needs. They were easy to handle, and faster than digital computers. The situation, however, has changed with time. The need to include more functions (multiplication and division, functions of several variables, logical decisions) led to combinations of analog and digital techniques in hybrid computers. Because of the growing speed of digital computing and the development and spread of programming languages for continuous simulation, digital simulation techniques became more and more dominant. Special-purpose languages for continuous simulation are based on two different representations of dynamic problems:

- block diagram representation; and
- differential equation representation.

The first simulation languages were block-oriented (such as DYNAMO [PUGH61], the original version of CSMP and others). The step from analog to digital was easy with these languages because of their common (block-oriented) way of representing models. Large and complex problems, however, would require huge block diagrams to be drawn. Such large problems may be formulated more concisely by a set of differential equations in mathematical notation. The differential-equation-oriented languages basically aim at a formal rewriting of these equations in machine-readable form, without intermediate graphic representation as block diagrams. Examples of these languages are DSL/90 [SYN__66] and CSMP III [CSMP72].

Quite often, dynamic problems which are described by partial differential equations may be treated by continuous simulation languages. For this purpose, the partial differential equation is approximated by finite differences in the space domain. Thus the continuous field is discretized in space. The resulting set of ordinary differential equations – with time as the only independent variable – makes problems tractable by conventional simulation programs, which otherwise would require the application of more sophisticated numerical methods, like finite elements or finite differences in more dimensions.

A similar separation may be found in early discrete simulation languages. Flowchart-oriented languages based on GPSS [BOBL76], [GORD78] are particularly easy to learn, while statement-oriented languages like SIMULA [DAHL70] provide more flexibility.

In activity-oriented languages, the components of the simulated system (the entities) are either idle (do not change their state) or else perform activities in time which – if certain conditions are met – lead to instantaneous changes of the system state. By the cyclic scanning of all activities from time step to time step and modification of the system state accordingly, the simulation proceeds through small intervals in time.

Many problems may be formulated more effectively in an event-oriented language. With languages of this type, time proceeds in jumps from one event to the next one, each resulting in a discrete change in the system state. Upon each event, the system components are scanned to see whether they will give rise to one or more events in the future under the new system conditions (for instance, sending a message will cause a future event: the arrival of the message at some other place). All future events are sequenced in time, and the event which will occur soonest determines the next time step.

Process-oriented languages attempt to combine the compact notation of activity-oriented languages with the efficiency of event-oriented languages. An example of such a language is SIMULA [DAHL70]. A single subprogram in SIMULA may represent many processes of a similar type, each of them in a different state. Each process will cause the corresponding program to resume execution at the appropriate reactivation point.

In addition to these special-purpose languages, simulation programs may be written in general-purpose languages like FORTRAN, C, or PASCAL. In the case where a digital computer is combined with real components (simulated aircraft and real cockpit), process control languages like PEARL [PEAR77] would be advisable. In the CAD environment, the use of general-purpose languages appears to be more widespread than the use of specialized languages, at least for the simulation of continuous processes. This is because:

- the simulation programs must interface with other programs and databases used in this environment. This interfacing is most easily achieved when only a single language (generally FORTRAN) is used for all purposes; and
- many organizations use a home-made simulation package – usually based on FORTRAN – which suits their own needs better than the so-called special-purpose simulation languages, which may be too "general purpose" in the particular case.

Many practical problems call for both discrete and continuous simulation. Pritsker and Young use the example of pilot ejection from an aircraft to illustrate such needs [PRIT75]. In the first phase of this process, the pilot's seat is guided by its mounting rails and moves under the forces of the ejection system. When the seat disengages from the rails, the equations of motion for the aircraft and the pilot become suddenly uncoupled from each other. Atmospheric drag forces start to act on the pilot. Thus, a discrete change in system state separates two phases of continuous

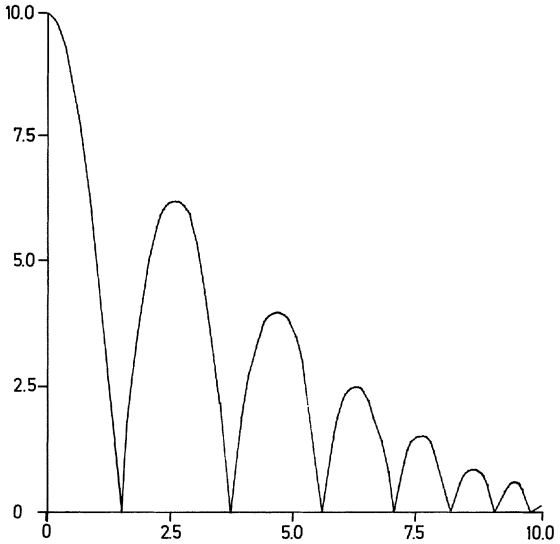


Fig. 6.19. The "bouncing ball" problem calls for combined continuous and discrete simulation

simulation. Another example has been treated in [SCHL70]: the bouncing ball. The falling ball changes height and velocity continuously under gravitational forces. A discrete change in state occurs when the ball is reflected from the ground (see Fig. 6.19). Simulation systems for problems of this type must be able to cater for situations which call for a discrete change in system state at unpredictable times within an otherwise continuous process. In addition to such features, the simulation system DYSYS provides for the exact simulation of transport processes at variable speed (such as material transport through a piping system with varying pumping power). Another system capable of combined continuous and discrete simulation is GSL [GOLD73].

6.2.5 Optimization

6.2.5.1 Problem Formulation

In a sense, design may be considered an optimization task. Even if the specification does not explicitly call for an "optimum" design, the designer will always try to optimize in one or more respects:

- production cost and time;
- ease of assembly and maintenance;
- ease of use; or
- time and resources required for the design.

Instead of posing an optimization criterion, the same aspects may set forth constraints which have to be met by the design. However, there are significant differences between optimization and design:

- Design, while aiming at an optimal solution within the limits of certain constraints, does not generally start from an explicit formulation of the optimization criteria and all the constraints. Even worse, in many cases design is confronted with an uncertain and moving target (see Sect. 3.1.2). The definition of what is "optimum" is commonly left to human judgment. While some constraints may be formally given (a maximum allowable stress value, for instance), the distance from such a limit is often associated with a certain value and included in the judgment. Often, this judgment must also take into account uncertainties in the specification. Thus the designer may decide not to select the solution which appears to be "optimal" based on the presently available information. He may rather prefer a "second choice" if it provides more flexibility for adaptation to changing requirements. What is meant by flexibility (or maintainability, manufacturability, etc.) is certainly not easily expressed in terms of formal parameters and a formal measure of merit function.
- Optimization assumes that a single measure of merit function (often called "objective function") and the complete set of constraints can be expressed formally in terms of the design parameters. The optimization task may be given as in [LUEN73]:

minimize $f(\mathbf{x})$
subject to $\mathbf{x} \in S$
$h_i(\mathbf{x}) = 0;\ i = 1 \ldots r$
$g_j(\mathbf{x}) \leq 0;\ j = 1 \ldots c$

In this formulation, $\mathbf{x}$ is the n-dimensional vector of the design parameters that have been identified as the ones which may be adjusted to obtain an optimum. S is an allowable domain in the corresponding n-dimensional space; and $f(\mathbf{x})$ is a single function of these parameters. Functions of the form $h_i(\mathbf{x})$ represent implicit restrictions (equality constraints). Each of them effectively subtracts one dimension from the design parameter space by defining a hypersurface on which the solution must lie. The (inequality) constraints $g_j(\mathbf{x})$ further reduce the allowable domain; they define other hypersurfaces in the parameter space which constitute boundaries for allowable solutions. Note that the above formulation is general enough to cover the maximization task

maximize $f'(\mathbf{x})$

as well. We simply substitute

$f(\mathbf{x}) = \text{arbitrary\_\_constant} - f'(\mathbf{x})$

Hence it does not matter whether we formulate optimization as finding the minimum or the maximum of a measure of merit. A special case is the problem of finding a feasible solution without imposing an optimum criterion. This case may be represented by an optimization task with a measure of merit

$f(\mathbf{x}) = \text{arbitrary\_\_constant.}$

Other constraint formulations are included in the above scheme as well:

$g'_j(\mathbf{x}) \leq C' \quad \text{or} \quad g''_j(\mathbf{x}) \geq C''$

may reduced to the above formulation by substituting

$g_j(\mathbf{x}) = g'_j(\mathbf{x}) - C' \quad \text{or} \quad g_j(\mathbf{x}) = g''_j(\mathbf{x}) - 1/C'',$

respectively.

Many textbooks on optimization describe methods for various classes of optimization problems [LUEN73], [AOKI71], [KUEN67], [FLET69], [WILD67]. The different classes may be characterized by:

- the continuity aspect;
- the dimensionality of the parameter space; and
- the type of functions allowed for f, h_i, g_j.

The methods for finding the optimum solution may be classified as

- techniques based on analytical approximations, using the functions themselves and their derivatives; or
- search techniques (often called "hill climbers").

Which one of the many different techniques is best suited for a given problem depends very much on the problem characteristics.

6.2.5.2 Optimization Problem Characteristics

Continuity

The overwhelming majority of optimization methods deal with continuous functions as well as continuous allowable domains in the parameter space. Application examples which may be found in the literature usually exhibit continuous behavior of the measure of merit, and of the constraint functions. With respect to the parameter space, discrete subsets are typical for many design applications. The discretization is often a consequence of standardization: diameter values, for instance, or screw sizes cannot be chosen from a continuous domain, but rather from a set of standard values. Two approaches may be used to solve such problems:

- We perform discrete optimization in a strict sense by selecting allowable parameter values only. Many optimization techniques will fail in this case because they are based on the assumption of continuity.
- We disregard the discreteness problem while solving the optimization problem. Once the optimum is found in the continuous parameter space, we select from the allowable set of discrete parameters those values which appear to be "closest" to the optimum without violating the constraints. Since, in most practical cases, the measure of merit is flat near the optimum – that is, a small deviation from the optimum usually has only an insignificant effect on the measure of merit – this approach will produce a solution which is at least close to the real optimum. The advantage of this approach is that it leaves us a greater choice of optimization techniques.

Dimensionality

A wide class of optimization problems is of dimensionality one. The problem may then be considered as the task of finding the minimum (or maximum) of a curve in a certain interval. Search methods for solving this problem include the Fibonacci Search and the Golden Section Search [WILD67]. These methods differ in their strategies for selecting the inspection points within the interval. If the interval is not limited, the Reversed Fibonacci Search may be used. Other methods apply curve-fitting techniques to the function: Newton's Method or the Method of the False Position [LUEN73]. These (direct) methods require the evaluation of the derivative of the measure of merit, or its approximation by a difference quotient.

Problems with a parameter space of higher order may be classified as "small" (up to order five), "intermediate" (from order five to a hundred), and "large" scale (up to a thousand), according to Luenberger. Today's theoretical and computational capabilities permit direct solution techniques for many small problems, while the intermediate scale is the domain of a large number of general-purpose search techniques (except for linear problems, which are more easily tractable by direct techniques since their derivative evaluation is trivial). Large problems call for special methods that account for specific structures in the problems themselves. Optimization for design, however, is far from ready for application on this large scale.

Function Type

The following classification according to function types is generally used:

- linear problems;
- nonlinear unconstrained problems; and
- nonlinear constrained problems.

Linear Problems

A linear optimization (or "linear programming") problem is defined by [KUEN67]:

$$\text{maximize } f(\mathbf{x}) = \Sigma\, a_{0i} \cdot x_i$$

subject to

$$g_j(\mathbf{x}) \geq \Sigma\, (a_{ji} \cdot x_i - a_{j0})$$

with $i = 1 \ldots n$; $j = 1 \ldots m$.

A different formulation introduces auxiliary variables x_{n+j} for expressing the constraints

$$\Sigma\, a_{ji} \cdot x_{n+j} - a_{j0} = 0 \qquad x_{n+j} \geq 0$$

A graphical representation of the optimization task is possible only for two parameters. Figure 6.20 illustrates the problem

$$\text{maximize } f(\mathbf{x}) = x_1 + 3 \cdot x_2$$

subject to

g_1: $x_1 \geq 0$ $\qquad$ g_4: $x_1 + 4 \cdot x_2 \leq 20$
g_2: $x_2 \geq 0$ $\qquad$ g_5: $x_1 - x_2 \leq -1$
g_3: $3 \cdot x_1 + x_2 \leq 24$ $\qquad$ g_6: $x_2 \leq 6$

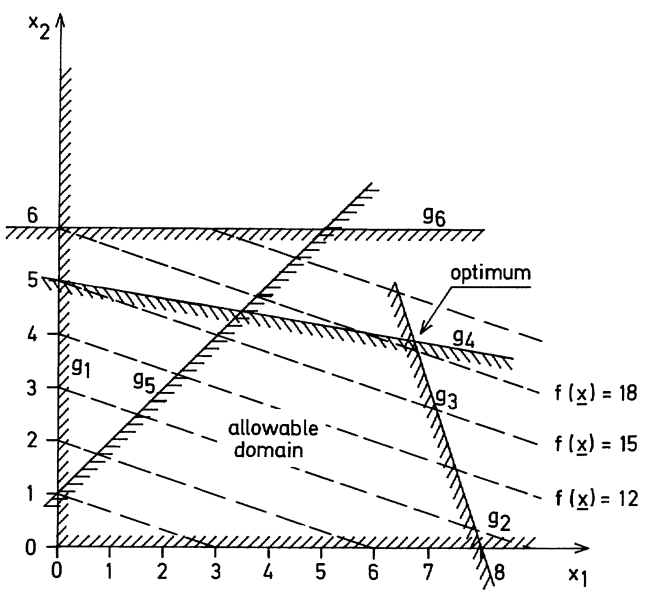


Fig. 6.20. A linear programming example in two dimensions

The allowable domain of parameter values is indicated for each of the constraints by lines hatched on their allowable side. Note that the last of the constraints is irrelevant, since the combination of the remaining constraints is more restrictive. The isolines of the measure of merit

$$f(\mathbf{x}) = \text{const}$$

indicate the optimum location on the boundary of the allowable domain.

The basic approach for solving linear programming problems is the SIMPLEX method. This algorithm, and others which are based on it but modified for higher efficiency, are extensively covered in textbooks (such as [LUEN73]). For FORTRAN and Algol listings of linear programming algorithms, see [KUEN67].

Nonlinear Unconstrained Problems

One of the oldest and most widely used methods for minimizing a function of several variables is the Method of Steepest Descent. An equivalent name is the Gradient Method. At a starting point x_0 in the parameter space, the gradient $\mathbf{g}(\mathbf{x})$ is determined and a new position

$$\mathbf{x}_1 = \mathbf{x}_0 - a \cdot \mathbf{g}(\mathbf{x}_0)$$

is chosen such that

$$f(\mathbf{x}_1) < f(\mathbf{x}_0)$$

is minimal for all $a > 0$.

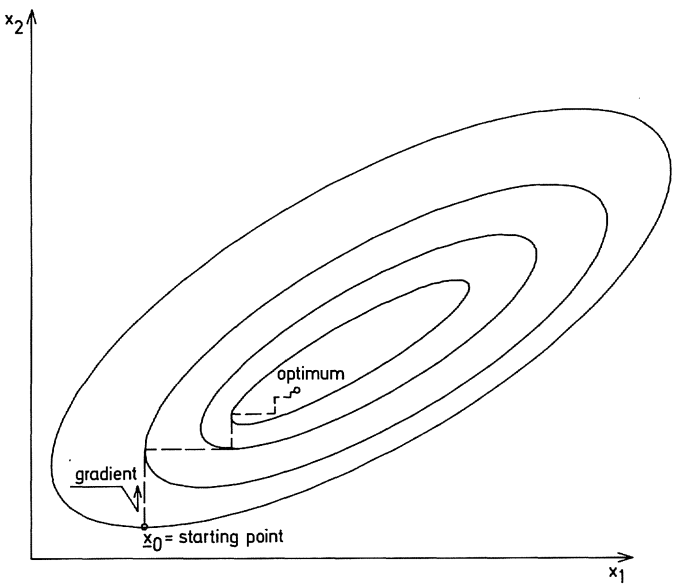


Fig. 6.21. Approach to optimum with the Gradient Method

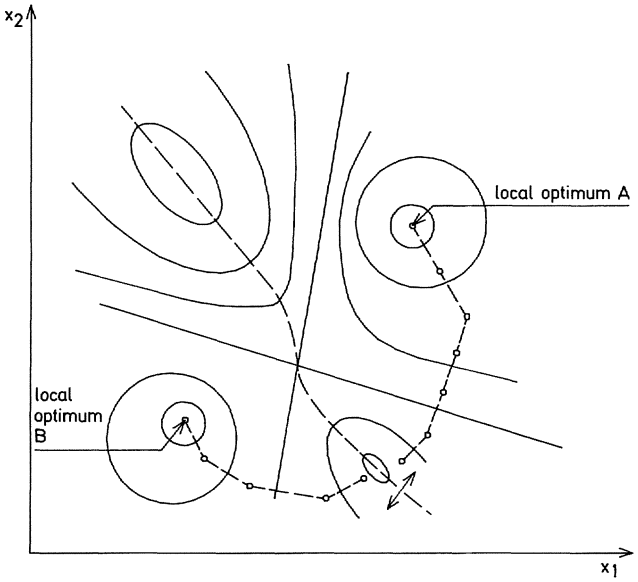


Fig. 6.22. The starting point of the Gradient Method determines which local optimum is found

Point $\mathbf{x}_1$ is then used as a new starting point. Figure 6.21 shows the successive steps of this method for a two-dimensional case. The method will converge to a local optimum. But in the general case the optimum which is thus found may not be the "best" one. It may be better than all other points in a certain neighborhood, but there may exist a better global optimum. When there are several local optima, the one found will depend on the starting position $\mathbf{x}_0$, as illustrated in Fig. 6.22.

A particular case of a nonlinear measure of merit is the quadratic problem

$$f(\mathbf{x}) = 1/2 \cdot \Sigma\, a_{ij} \cdot x_i \cdot x_j + \Sigma\, b_i \cdot x_i$$

For this problem, deep mathematical foundations are available [KUEN67]. More general measures of merit are often approximated locally by such a quadratic function and treated accordingly with the Gradient Method. A similar approach is the use of Newton's Method. Combinations and modifications of these two basic direct methods have been developed for improved efficiency [AOKI71]. For FORTRAN and Algol listings, see [KUEN67]. The common feature of these methods is the use of the derivatives of the measure of merit with respect to the design parameters.

Techniques which make use of the function values only are called search techniques. A primitive method of this type is Unidirectional Search (see Fig. 6.23). From a starting point $\mathbf{x}_0$ we move a certain step into one of various directions, holding all but one parameter constant. If the measure of merit is improved, we continue to move into this promising direction (possibly with increasing step size). Upon failure (no further improvement in this direction), we try other directions in the same way. If none of the directions indicates an improvement, we have found an optimum within the range of the step sizes. The accuracy may be improved as far as desired by continuing this process with a reduced step size. The Unidirectional Method may fail, however,

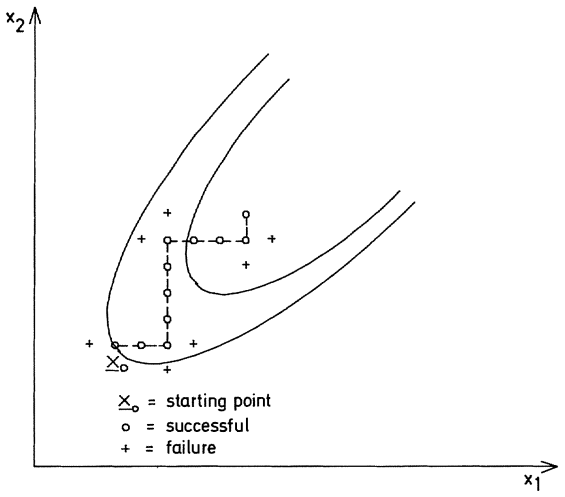


Fig. 6.23. Approach to optimum with the Unidirectional Method

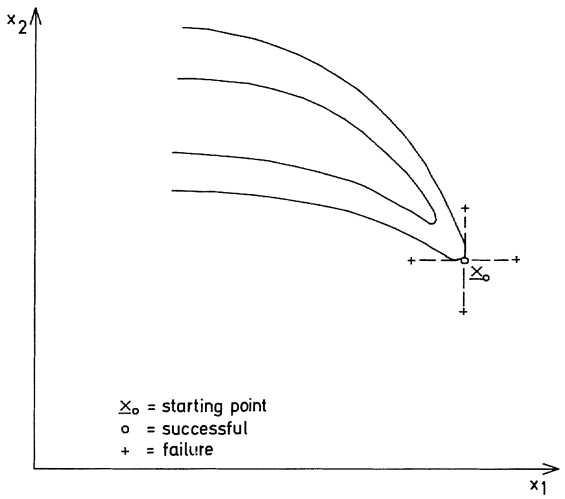


Fig. 6.24. A situation which causes the Unidirectional Method to fail

in extreme situations, as shown in Fig. 6.24. The problem of sometimes finding a local optimum – but perhaps not the global one – is the same with this method as for the Gradient Method.

A completely different type of search technique is the Random Search. Several procedures of this type have been developed, including the Evolution Strategy which is claimed to simulate biological evolution [SCHW77], [HECK78]. The Random Search techniques proceed by evaluating the measure of merit at points selected at random in the parameter space, using allowed values of **x**. They are not necessarily located close to previous inspection points, as with the other (deterministic) methods. The benefit of Random Search is that it depends much less upon the starting point and thus has the potential of finding the global optimum, even if the measure of merit

and the constraints are very complex functions in a parameter space of high dimensionality. Furthermore, Random Search can deal with discrete design parameters without any problem. Since a totally random search would likely produce a prohibitively large number of inspections, the various Random Search techniques all utilize strategies for modifying the probability distribution of the inspection points in accordance with past evaluations of the measure of merit.

Nonlinear Constrained Problems

The methods which are available for solving unconstrained optimization problems require modifications when constraints are introduced. Methods utilizing the derivatives of the measure of merit may be adapted to constrained problems by using the concept of the Feasible Direction [ZOUT60]: When the optimization algorithm reaches a boundary, it can no longer proceed in the direction of the gradient into the forbidden domain. A feasible way to proceed is the projection of the gradient onto the tangent plane of the constraints. For nonlinear convex constraints, this direction may need further correction to assure that the next inspection point will fall inside the allowable domain. The result is a zigzagging path formed by the sequence of inspection points (indicated in Fig. 6.25), which follows the boundary closely. It is worth noting that finding a feasible and usable direction (feasible = does not violate constraints; usable = improves the measure of merit) is in itself a linear optimization problem [AOKI71].

A different approach is the modification of the measure of merit by a suitably chosen penalty function $P(\mathbf{x})$ or a barrier function $B(\mathbf{x})$, as described, for instance, in [CZAP76]. We minimize $f(\mathbf{x})+P(\mathbf{x})$ or $f(\mathbf{x})+B(\mathbf{x})$, respectively, instead of $f(\mathbf{x})$ proper. The penalty function is null in the allowable domain but rises steeply as we move

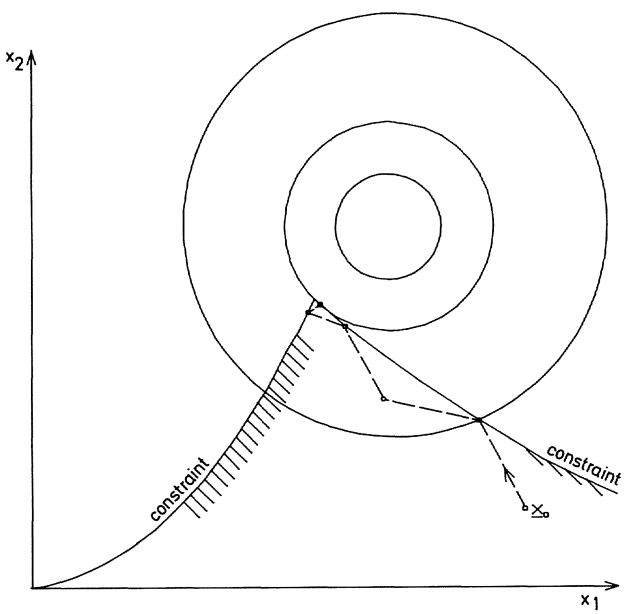


Fig. 6.25. Approach to optimum with the Method of the Feasible Direction

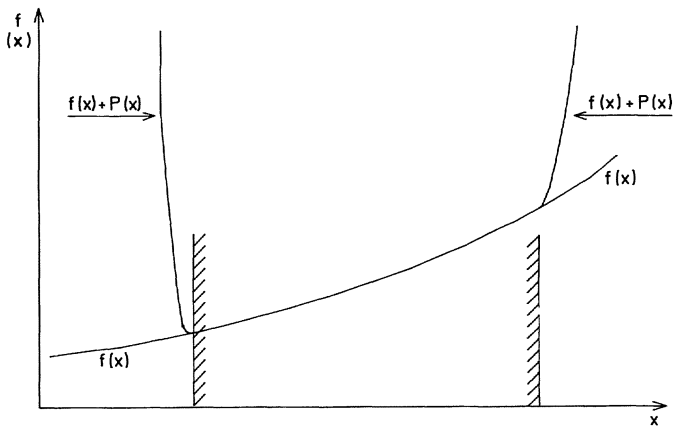


Fig. 6.26. A measure of merit modified with a penalty function

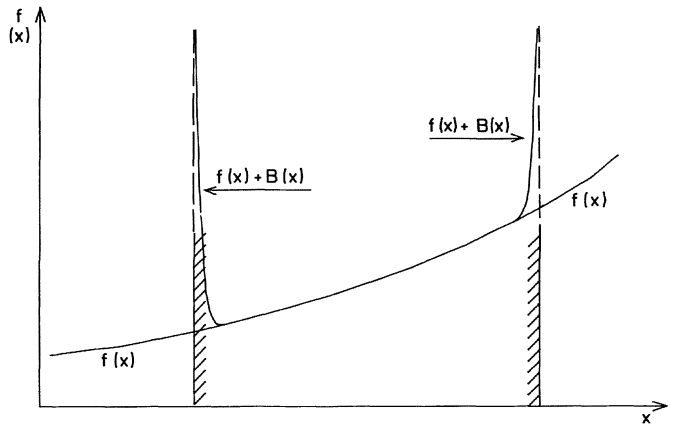


Fig. 6.27. A measure of merit modified with a barrier function

across a boundary into the forbidden region (see Fig. 6.26). The barrier function, however, modifies the measure of merit inside the allowable domain, as shown in Fig. 6.27. Thus, while with the penalty function approach we might slightly violate a constraint with our final solution, the barrier function approach may stop us slightly before reaching the actual feasible optimum. Both methods effectively push the solution to as close to the boundary as is needed, provided that their respective functions (barrier or penalty) are sufficiently steep.

Similar corrections must be applied to direct search techniques. The unidirectional search, for instance, would fail in a situation like the one shown in Fig. 6.28, if applied without further consideration. Random search techniques are readily applicable to constrained problems; due to their versatility they will succeed whether there is a boundary or not. However, the convergence rate may be improved if recognition of a boundary is used to modify the probability distribution of the next inspection points

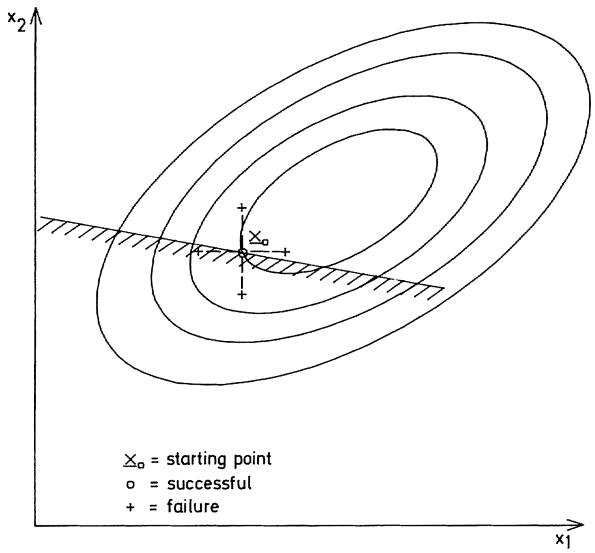


Fig. 6.28. A constraint which causes the Unidirectional Method to fail

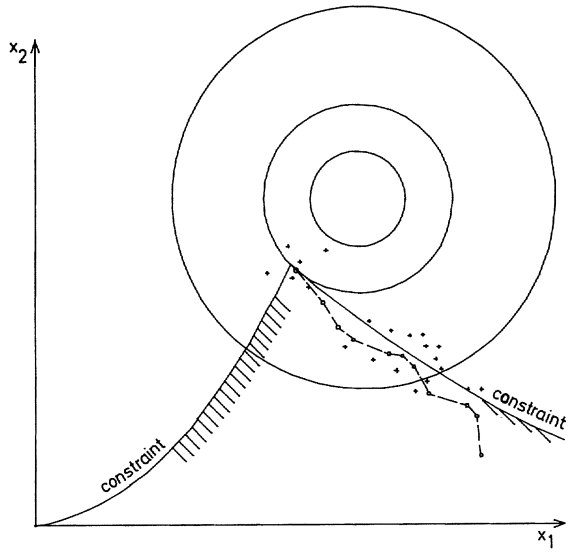


Fig. 6.29. Approach to optimum with Random Search

(as compared to blind random searching). Figure 6.29 shows the approach to optimum for a random search in the same case as for Fig. 6.25. Both figures are taken from [HEUS70].

6.2.5.3 Applications

The application of optimization techniques to design requires, first of all, a formal statement of the problem. The design task has to be properly prepared before op-

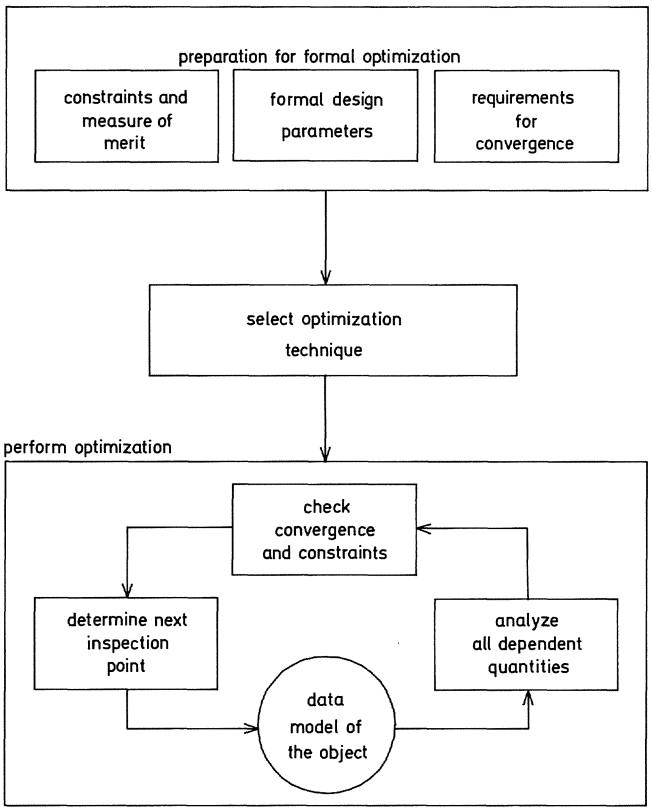


Fig. 6.30. The application of optimization methods in CAD requires a formal model and a formal measure of merit

timization techniques may be applied. The main steps to be taken by way of preparation are (see Fig. 6.30):

- Formalization of the design parameters and all dependent quantities used in the evaluation of the constraints and of the function of merit. This task corresponds to the definition of a schema for an object of design, and is thus a familiar task to the computer-aided design specialist.
- Formulation of the algorithms for computing the constraints and the measure of merit. In practical cases these functions are often not expressed directly in terms of the design parameters, but rather as functions of dependent variables (such as stress levels). Numerical analysis and simulation problems are generated as subtasks of the optimization.
- Definition of convergence criteria. Optimization is an iterative process in most practical cases. Criteria must be set forth to define when the solution should be accepted as final.

After these preparations, an optimization technique can be selected and the optimization may proceed. The individual steps of the optimization process are very

similar to variational design (see Sect. 3.2.2). The values of certain attributes of the design object have to be determined. The correspondence is as follows:

optimization	↔ *variational design*
determine next inspection point	synthesis
compute dependent quantities, including measure of merit and constraint functions	analysis
check violation of constraints and convergence to optimum	evaluation

Considerable resources (computer time, money, manpower) may be required for the actual performance of the necessary analyses for a single inspection point. If we assume that many evaluations (say, 50 to several hundred) are needed to approach the optimum, the resource requirements are likely to become unacceptable. A possible remedy is the use of the "response surface" technique [MYER71]. With this method, we consider each quantity $y_k(\mathbf{x})$ as a hypersurface in the $(n+1)$-dimensional space spanned by the n design parameters and by y_k. Before actually performing the optimization, we compute a sufficiently large number of points on these hypersurfaces, using the complete model of the design object and the most refined analysis techniques. Based on these points we try to represent the real hypersurface by an approximate one, which should be accurate enough but much easier (and hence less costly) to evaluate. This approximation is the "response surface". In the most simple case it would be a linear function of the design parameters. In many practical cases it turns out that the approximations have to account for dependence on only a small number of design parameters, if the quantity y_k is dominated by the influence of only a few parameters. This can drastically reduce the computational effort, and may make automatic optimization feasible in cases which otherwise would have had prohibitive resource requirements. It is interesting to note that the determination of a "good" approximate response surface is in itself an optimization task.

Despite the similarity between "design" and "optimization", the number of successful applications of optimization techniques to design problems appears to be surprisingly small. In 1979, a special issue of Computers and Structures on "Trends in Computerized Structural Analysis and Synthesis" contained only a single paper on optimization techniques for the synthesis of structures [NOOR79]. The reason for this lack of application examples is probably twofold:

- the difficulty in formulating a quantitative measure of merit or constraint function involving aspects like "manufacturability" and "maintainability", or even aesthetic aspects; and
- the very high dimensionality of the design parameter space in practical applications.

In structural design, the area of weight minimization for truss structures appears to be fairly well covered [SVAN81]. Examples of optimization applications in pre-contract ship design are given in [NOVA73]. Aoki also gives a number of optimization examples [AOKI71]. One of the problems involves optimization not only of a product, but of the whole project, including development, production, and use of the product. It is based on an empirical model for the total project costs of a three-stage launch vehicle. The model was developed by Lockheed Missiles and Space Company and was solved by Rush [RUSH67] in a parameter space of dimension 25 with 26 con-

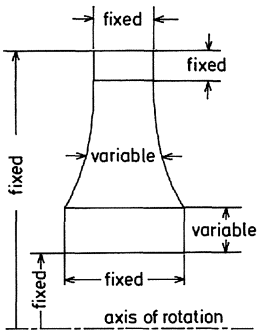


Fig. 6.31. Rotating disk; optimization example by deSilva [DESI69]

straints. This is an interesting example indicating the value of optimization in a very early design stage. Optimization techniques have also been applied in the early development phase of the German fast breeder project. Heusener used a combination of random search (while far from the optimum) and the feasible gradient method (when approaching the optimum) which gave an improved convergence rate over the use of either of these methods alone [HEUS70]; the problem involved 9 parameters and 11 constraints. Typically, 40 to 50 evaluations of the measure of merit were needed for finding an optimum. Since the analysis of each point in the parameter space would have required very costly thermal-hydraulic and neutron physics calculations, the response surface technique was used. DeSilva [DESI69] illustrates a minimum-weight design for a rotating disk subject to geometrical and stress constraints (see Fig. 6.31). The problem is quadratic in the variables which define the disk shape. Polygons with 4 to 11 corners were used to approximate the curved shape. Typically, 50 to a few hundred evaluations of the measure of merit were needed for the approach to optimum.

Although we may expect the number of automated design optimizations to increase in the future, in all likelihood they will be restricted to relatively simple subproblems (such as optimum weight design of not-too-complicated parts), or to very early stages of design in which the product or perhaps even the whole project may be described with sufficient accuracy by a model of small-to-medium complexity (say up to dimensionality 20 or 30). A possibly new approach would be an interactive optimum design, with man included in the system. The optimization algorithm (probably based on random search) would be guided by an operator who:

- monitors certain design constraints (such as unacceptable style) while others (such as gross dimensions) are monitored by algorithms;
- adds his personal judgment of the present design to the measure of merit; and
- guides the direction of search by influencing the probability distribution for the next inspection point.

6.3 Computer Graphics for Data Presentation

6.3.1 Introduction

The application of powerful numerical methods to the analysis of design objects generally produces huge amounts of data. In order to make these results useful for interpretation and evaluation, they need to be presented in a readily understandable form. Long lists of printed data are totally unsuited for comprehension by a person. Graphic representations are the appropriate solution [SCHL81]. The subject of computer graphics is closely related to numerical methods and, hence, has been included in this chapter. The superiority of graphic data presentation stems from two facts:

- Information contained in graphical representations can be perceived by a person in an immediate and integrated way (in parallel), while lists or printed data must be parsed more or less sequentially. "One picture is worth a thousand words."
- The graphic representation can produce evidence of features like "peaks" or "waves" or "shock fronts". Such features are often essential for the evaluation. Note that finite difference or finite element techniques do not intrinsically treat such features, but the results may be expressed most effectively in these terms.

To a great extent, graphic representations are used for the following purposes:

- immediate presentation of results to the engineer who is responsible for the analysis;
- reporting results to a customer or supervising authority;
- publications, presentations; and
- long-term documentation.

The requirements regarding turn-around time and visual appeal of the representations vary considerably. For the analyst, fast response has the highest priority: thus either graphic displays or fast plotters (such as electrostatic plotters) are appropriate. For reports and publications, aesthetic aspects become more important, which makes line plotters (particularly table plotters) more attractive.

For presentations and long-term documentation, the space requirements for the display medium can be minimized by utilizing graphic output on microfilm, microfiche, videotape or optical disk. Representation of analysis results on a display or plotter medium is only part of the task. In most cases (except for the immediate presentation of results to the analyst himself) the representation of the data must be edited. Coordinate axes must be added to diagrams; annotations, sketches or scaled-down drawings of the design object must be included with the data representation. This is a typical task for interactive graphics.

6.3.2 Functions of One Variable

6.3.2.1 Diagrams

The standard technique for representing a function of one variable is by means of a diagram. Various forms of diagrams (for single functions) are shown in Fig. 6.32

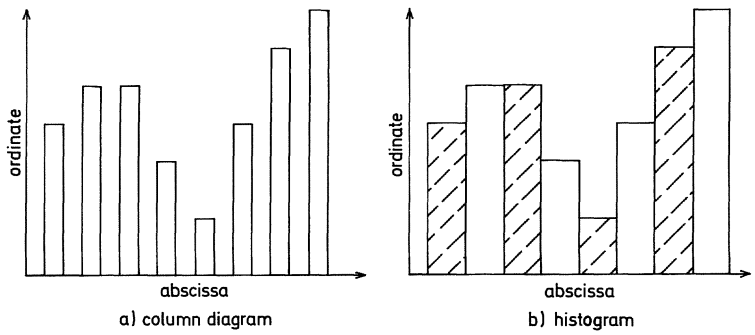


Fig. 6.32. Representation of a discontinuous function of one variable

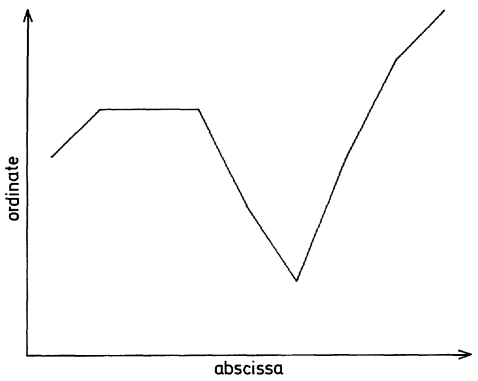


Fig. 6.33. Representation of a continuous function of one variable by linear interpolation

through Fig. 6.34. They are based on the same data points and are placed in order with an increasing degree of continuity. Column diagrams (Fig. 6.32a) or histograms (Fig. 6.32b) should be used to display discontinuous functions. Such functions occur quite often when the abscissa does not represent a continuous coordinate but rather a number of discrete values (such as different design options, or discrete power levels, for instance). Continuous diagrams (Figs. 6.33, 6.34) usually result from one-dimensional analyses (temperature, pressure, stress level distributions), time response simulations, or frequency response analyses. Here the abscissa represents a space coordinate, time, or frequency respectively. Alternatively the function may continuously depend on a design parameter (such as maximum allowable temperature). Figure 6.33 shows a linear interpolation of such a continuous function between computed data points. Figure 6.34 is a smooth representation of the same function. Both approximations and interpolations (in pieces, as with spline functions, or over the whole range with a single polynomial) are common techniques for smoothing. The difference between the linear interpolation and a smoothed representation is of an aesthetic nature. The smooth curve looks better, but the linear interpolation is more truthful. It does not suggest a more accurate analysis than was actually achieved. Furthermore, when cost and turn-around time are of great concern, the linear interpolation is cheaper and faster than smoothing. Thus, for technical communication

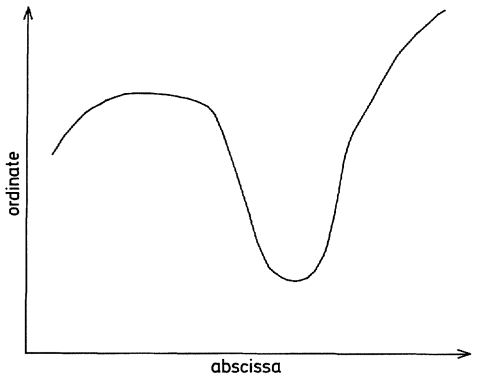


Fig. 6.34. Smooth curve representation of a function of one variable

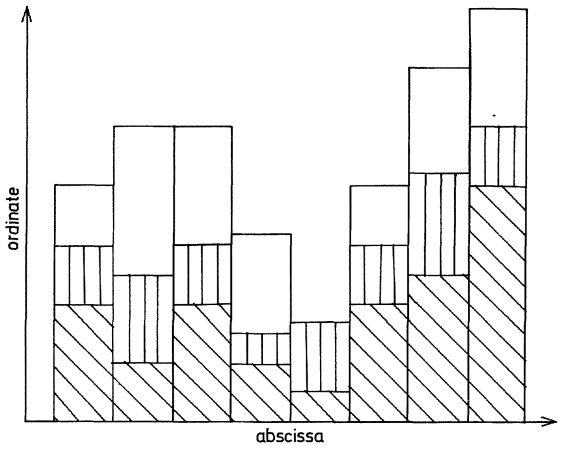


Fig. 6.35. Simultaneous representation of several continuous functions having the same measure

Fig. 6.33 would be preferable, while for commercial presentations or advertisements the smooth curve will probably be the better choice.

6.3.2.2 Representations of Several Functions in One Diagram

The representation of more than one function of only one variable does not pose great problems. We may distinguish between the different curves by using different line styles or colors, by adding markers, or by shading, hatching, or coloring the area below a curve. Figure 6.35 shows a histogram. In this instance, three quantities (with different hatching) are shown added together to form each total. This possibility applies only to quantities having the same measure (for example, production costs of different components of an object, adding up to the total production costs). Pseudo-perspective representations (as shown in Fig. 6.36) may be used for quantities without a common measure, such as cost, weight, efficiency, and manufacturing time. While hatching (or shading, or coloring) is not necessary in this case, it certainly improves the readability of such diagrams. Continuous functions of one variable are often dis-

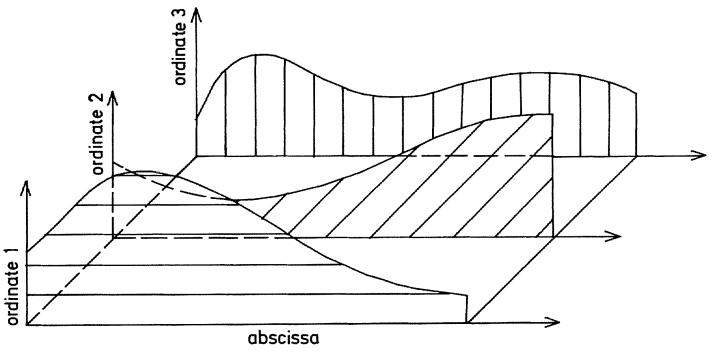


Fig. 6.36. Pseudo-perspective view of several functions with different measures

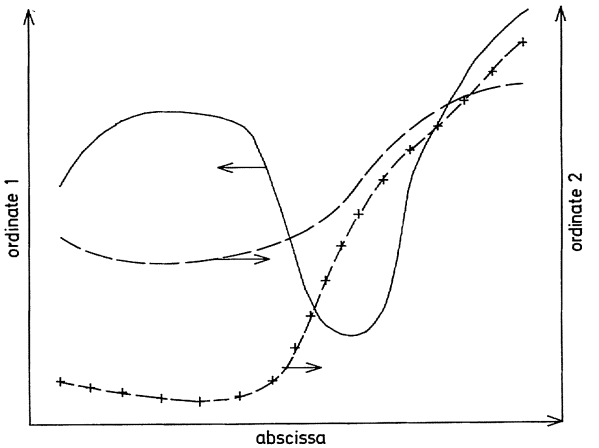


Fig. 6.37. A diagram with two ordinates for functions of different measures

tinguished by different line styles (line widths, line colors) or by adding markers to the curves. More than one ordinate axis may be added to the diagram but to use more than two (or possibly three) is not recommended (see Fig. 6.37).

6.3.3 Functions of Two Variables

With the growing capability of numerical computer programs, the detail analysis of design objects has shifted from one-dimensional to two-dimensional or even three-dimensional problems. Graphical representations, however, are at first glance limited to the two dimensions of a display screen or a plotter. Thus, while the representation of a function of one variable is a simple task, representations of functions of more variables may require rather sophisticated methods.

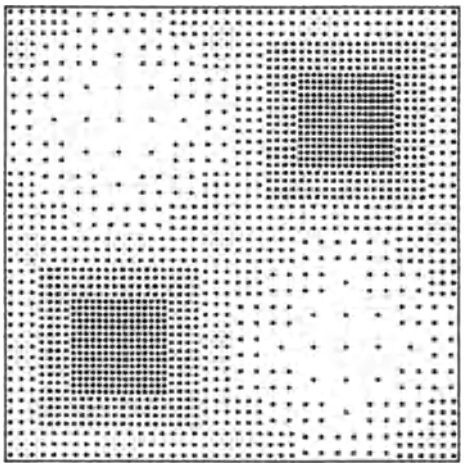


Fig. 6.38. Representation of a function of two variables by regular marker clouds

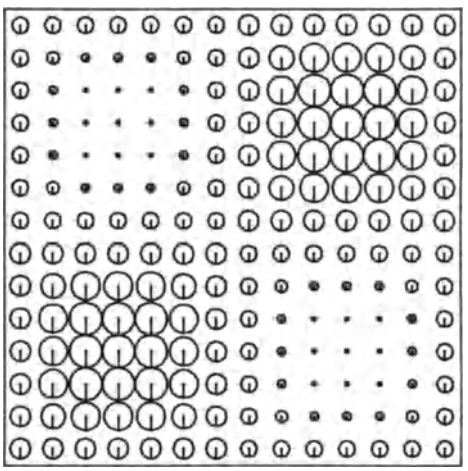


Fig. 6.39. Representation of a function of two variables by markers of different size

6.3.3.1 Marker Clouds

In order to present a single scalar function of two variables one may map the value of this function onto the reciprocal distance of markers. Thus the apparent density of the displayed markers indicates the variations of the function. As an example, the function

$$\sin(x) \cdot \sin(y)$$

over a 2π range in either direction is shown in Fig. 6.38. In this instance, the markers are distributed regularly within small rectangles for which some representative value was computed. Another option is to use a pseudo-random number generator to make the representation look smoother. This type of representation is often chosen to represent densities of physical quantities (such as densities of a certain material in a mix-

ture). If color is available, the random marker cloud technique may be used successfully to represent several densities in one diagram, with the corresponding markers displayed in different colors. In principle, black and white markers of different shape could also do the same job. But different colors make the perception of the separate functions much easier. In many fluid dynamics programs, markers are used to depict the movement of material through time. At the start of the computation, the material densities are mapped onto marker populations. These markers are then viewed as "marked particles" which move according to the computed velocity field. Markers of different size (Fig. 6.39) or shape or color can each be associated with a function value.

The applicability of the marker cloud technique depends upon the hardware which is used for visualization. The technique is unsuited for drum or table plotters because it may require many hours to draw so many markers. The technique is useful for electronic displays (particularly raster displays) as well as raster plotters (electrostatic, laser or ink jet) and microfilm plotters.

6.3.3.2 Hatching, Shading, and Coloring

A different technique for representing a single scalar function of two variables is based on hatching, shading, or coloring areas. The hatching technique can be used on vector-oriented hardware (e.g. line plotters). For vector displays, hatching may soon exceed the refresh capacity and lead to flicker. On raster-oriented hardware (plotters as well as displays), shading is preferred because it can produce a smooth-looking picture (as opposed to hatching which is always restricted to discontinuous steps in a representation). Figure 6.40 shows the same function as Fig. 6.38 with finer raster and 16 levels of shading. Color is even better suited for showing both discontinuous and smooth variations of a function.

The shading technique can even be used with the most primitive plotting device – the line printer. Appropriate choice of characters (from blank to M overprinted

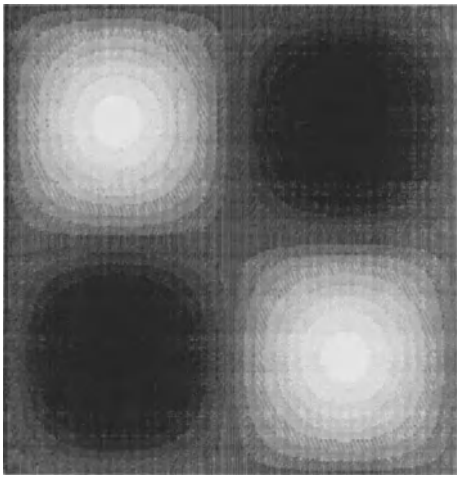


Fig. 6.40. Representation of a function of two variables by shading

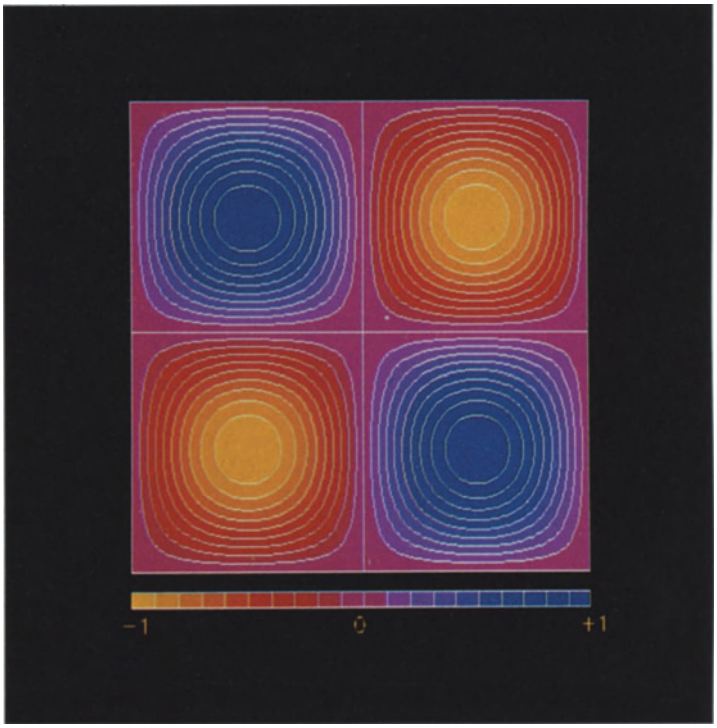


Fig. 6.41. Representation of a function of two variables by pseudo-coloring

with W) can produce the impression of a coarse black-and-white raster display. However, in a CAD environment suitable plotters and displays are commonly available so that the use of a printer as a plotter is less important.

Hatching, shading, and coloring are less well suited to representing more than one function in a single picture. A combination of coloring, marker clouds, and contour plots, however, could be used to display a small number of functions in a single picture. But care should be taken not to overload one picture with too much information.

Colored images for the representation of computerized results of structural mechanics programs were already being published in the early 1970s [CHRI74]. But the breakthrough of color is yet to come, although the color graphics market is developing rapidly [GANZ81]. Considering the advantages of color data presentations, this is surprising. A major problem with color is the cost of reproduction, which is significantly higher than for black-and-white line drawings or gray-shaded pictures. In any case, color pictures require a more delicate treatment. Figure 6.41 uses color to visualize the same two-dimensional function which appears in Fig. 6.38 through 6.42. Even though color pictures can now be produced more cheaply and much more easily than in the past (due to progress in color display and the use of instant color photography), the reproduction problem is likely to persist for some time.

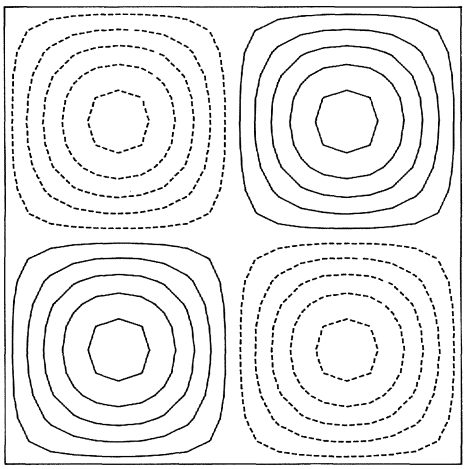


Fig. 6.42. Representation of a function of two variables by contour lines

6.3.3.3 Contour Plotting

A wide-spread technique for representing a scalar function of two variables is the contour plot (or isoline plot). Figure 6.42 is a contour plot of the same function as in Fig. 6.38. The advantage of the contour plot is that it is suited for vector-oriented hardware (line plotters, vector tubes). The hardware may even have an influence on the algorithm that is used to produce contour plots. Let us assume that the function is defined as a two-dimensional array Z(I,J) on a raster of points identified by indices $0<I<M$, $0<J<N$. The most simple algorithm scans one rectangle (formed by four adjacent points) after another, and determines whether a section on the contour line passes through each rectangle. The sections of the contour line are then plotted piecewise as they are found. As a consequence, a vector-oriented plotter may spend an unacceptable amount of time to produce the lines (not a problem on raster-oriented hardware). Thus, for vector-oriented plotters, an algorithm should be used that follows a contour line throughout the whole data field in order to plot it without interruption. Contour following is also required if the line is to be smoothed. Care must be taken to assure that adjacent isolines will not cross each other after smoothing. Because of this problem, simple linear interpolations (as in Fig. 6.42) are often preferred over smoothed curves.

The example used above is particularly easy because the data which represent the surface are arranged in a regular rectangular pattern. A slightly more general problem has the same degree of complexity: if the positions x and y of the data points do not form a rectangle but may be arranged in two-dimensional arrays X(I,J) and Y(I,J), then very simple algorithms may be used. The same is true for data resulting from finite element analyses as long as information about the topology of the finite element net is available (which nodes form an element? Which elements are based on each node?). In all of these cases the "neighborhood" of the data points is either explicitly or implicitly known. The problem becomes more difficult when only a vector of data

X(I), Y(I), Z(I) is given. The problem now resembles the task of constructing a solid body based on information about the vertices alone, without any knowledge about the topology of faces and edges. Nevertheless, contour-following algorithms have been developed for this task (as in [PATT78]). Such algorithms are often based on implicit assumptions of the following type:

- the surface does not overlap itself;
- the projection of the surface onto the xy plane is convex or has a predefined boundary; and
- the surface is spanned by triangles between the data points, the edges of the triangles representing shortest connections between the projections of the data points in the xy plane.

Whether or not isolines are helpful for our perception of the behavior of a function of two variables depends very much on the function itself. Isolines are well suited to indicate a small number of pronounced structures like

- peaks, or
- wave fronts.

They become less suitable if the function shows no such pronounced structures, or too many of them. It is impossible to perceive hills and valleys from isolines alone. They give no indication of the direction of slope (see, for instance, Fig. 6.17). The missing information should be supplied by additional aids to visualization, such as:

- annotations;
- different line styles at different levels (in Fig. 6.42, lines corresponding to positive or negative values are solid or dashed, respectively);
- different markers added to the lines; or
- area-oriented techniques (marker clouds, shading, coloring).

6.3.3.4 Pseudo-Perspective View

Another way to represent two-dimensional functions applies techniques which are known from the representation of curved surfaces. The function values are mapped onto a third spatial coordinate and the resulting surface in space can be treated in just the same way as a real curved surface in space. The same techniques for hidden-line removal or smoothing are applicable. Figure 6.43 shows the same function used previously in pseudo-perspective view. Pseudo-perspective views are very helpful for data presentation because:

- they are equally applicable for line-oriented and area-oriented hardware; and
- they provide an excellent means of perceiving the general shape of the function, even providing gross quantitative information.

Problems arise when the function values pass through several decades, or when hills in the front part of the view hide relevant information (as in Fig. 6.43, where a hill in the background is hidden). Showing the logarithm of the function instead of the function itself may help in the former case; viewing from different vantage points may help in the latter.

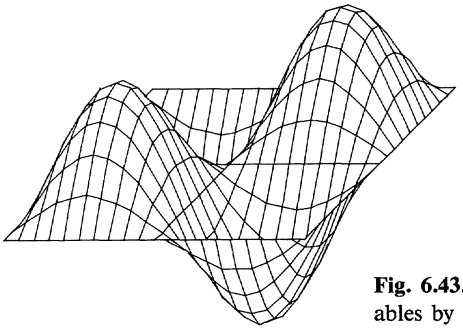


Fig. 6.43. Representation of a function of two variables by a pseudo-perspective view

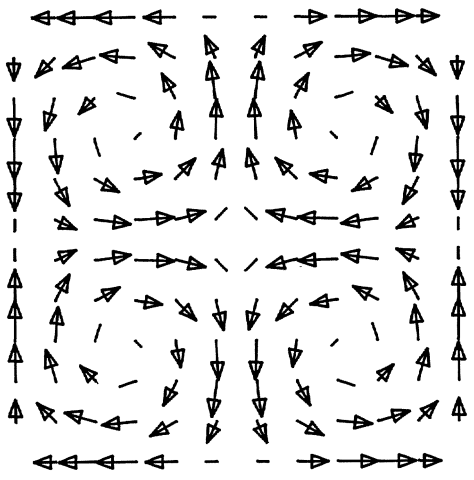


Fig. 6.44. Two-dimensional vector plot

An excellent method for visualizing a two-dimensional function is to produce detailed three-dimensional representation, as by milling the surface on a NC machine. Other techniques for generating a three-dimensional visualization are as yet in an experimental stage. Numerically generated holograms (to be viewed under laser light) and the use of two displays, with light polarization by means of filters (for viewing through polarized glasses and a half-transparent mirror) are approaching a stage that may make them attractive to the CAD community.

6.3.3.5 Vector Plots

The above mentioned techniques are applicable to the representation of scalar functions. Vector functions (such as velocity field) may be represented as vector plots (see Fig. 6.44). The vectores are represented as starting at the points of interest and pointing in the vector direction. The arrow length is given in proportion to the vector length (or in the case of extreme variations, in proportion to its logarithm). Care must be taken to avoid overlapping of the vectors, which would obscure the information. Two

vector fields may be represented in the same plot by using different arrow heads or different colors.

6.3.3.6 Two-Dimensional Functions on Curved Surfaces

Often the need arises to represent a two-dimensional function on the curved surface of a body. A typical example is the distribution of temperature or stress on the surface of a technical component. Pseudo-perspective views are not applicable in this case; hatching is not advisable (hatching lines would have to be curved); but other techniques (marker clouds, shading, coloring, isolines, and vector plots) are applicable. Obviously, different techniques should be used to represent the three-dimensional shape of the surface and the behavior of the function. If curved lines are used to represent the surface, shading or coloring is well suited for function representation. If the surface is represented by different shades (or colors), isolines are best suited to depict the function.

In summary, none of the techniques described here is the best in all cases. The available hardware has a significant influence upon the best choice; but for rapid and successful design analysis, there is no substitute for providing several of these techniques in an interactive graphics environment, so that the user can select among the various options according to his needs.

6.3.4 Functions of More than Two Variables

Three-dimensional functions (stresses, temperatures, etc.) result from static three-dimensional analysis or two-dimensional transient analysis (where time is the third dimension). When we represent a three-dimensional function on a two-dimensional display surface, we actually lose two dimensions (the dimension of the function value itself and one space or time dimension). Thus, we must look for techniques to simulate these two missing dimensions. Three methods can be used to overcome this problem:

- Clouds of markers in 3D space with varying size or color (according to the function value) can be projected from different view points (possibly with some scissoring applied). Human perception must reconstruct the full three-dimensional information from these views. As an alternative to several static projections, interactive display techniques might be provided to let the domain of interest be rotated under user control. As a compromise, the image may rotate or wobble in a predefined way; this technique is applicable for computer-generated movies and videotapes. 3D vector fields can be represented in this manner.
- Stereoscopic projection of marker clouds requires the overlay of two projections from appropriate viewpoints. Because of the problems associated with the separation of the two images (each one must be seen with one eye only) the technique has not become very popular. Wearing special glasses to separate light according to color or polarization is mostly too much of a burden for the user.

- The most successful way to represent three-dimensional functions is the use of techniques for two-dimensional functions (see above), allowing time to simulate the third dimension. Computer-generated movies or video tapes are an excellent tool for representing three-dimensional behavior.

6.3.5 Graphic Editing

The editing operations for data presentation usually comprise the following functions:

- Generation of coordinate axes:
  Care must be taken to assure the consistency of the displayed axes with the corresponding data display. In particular, when linear transformations (shift, rotate, scale) are applied to an already generated data display, the associated axes must be subjected to the same transformations.
- Annotations:
  Descriptive text must be added to most data displays.
- Viewing transformations and picture combinations:
  It quite often happens that data curves from different sources must be combined into a single diagram: analysis data have to be shown together with scaled drawings or sketches of the objects to which they relate. Thus, graphics for data presentation and graphics for geometric modeling need to be interfaced in an appropriate way. A suitable method is the use of a standardized data schema for both applications.

Figure 6.45 shows the concept of an editing system which has been developed and used successfully at Karlsruhe [KUEH79], [LEIN80]. It is based on two data formats:

- the PLOTCP format for data vectors and matrices [ZIMM75]; and
- the AGF-plotfile format for graphic information [ENDE80], [ENDE81], [ENDE78].

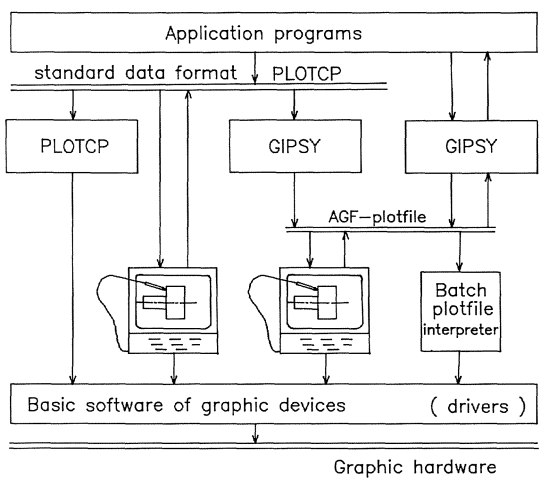


Fig. 6.45. Data format interfaces in a data presentation system

Graphic information can be produced from:

- the geometrical description of objects (3D or 2D) in the graphical programming language GIPSY [ENDE80], [ENDE81] or from interactive display terminals;
- the results of various simulations; or
- the results taken from corresponding experiments. Both data sources (experiments and simulation) can either generate standard (PLOTCP) data files or else generate graphic data without utilizing the intermediate interface.

Batch and interactive processors are available to combine information from these sources, to edit it, and to display it on various hardware. Since the output format of the graphic editors is again the standard AGF-plotfile, the edited pictures can sometimes be used in editing other pictures, for instance, by being inserted into them.

6.4 Summary

Numerical methods play an important role in CAD. An essential area of CAD is the treatment of geometry. The repertoire of analytical geometry and the numerical methods provided by mathematics for dealing with curves, surfaces, and solid bodies is implemented in today's CAD systems. On the other hand, CAD has itself pushed the development of these methods forward as there is an unsatisfied need for speeding up the geometry-handling algorithms and to extend their range of applicability.

Finite element methods are most widely used for structural analyses. For some time during the 1970s, computer-aided design was almost synonymous with finite element analysis. Today, finite element programs with various degrees of sophistication are commercially available. Computer centers offer their facilities to organizations which have less frequent need for such analyses. Static analysis based on linear theory of elasticity is most common. It can be mastered by the design engineer after a moderate amount of training. Dynamic analyses and non-linear analyses require a high degree of expertise, and are preferably performed by small groups of specialists in cooperation with the designer.

Finite element analysis is spreading in other domains: thermal analysis and fluid mechanics. The fundamental advantage of finite element methods is their flexibility with respect to geometry. But the conversion of the geometrical description of the object into the form required by the analysis programs – and the presentation of the results – produce considerable pre- and post-processing costs. Finite difference methods exhibit other advantages. They are easier to implement, and are often more efficient for regular geometries. But they lack the geometrical flexibility. Boundary integral equation methods and spectral methods, though rapidly developing, remain specialized and very efficient tools for particular cases; so far, they do not appear to have become widespread in the design environment.

Simulation methods have been developed for many classes of problems in the past. The most pronounced basis for distinction among these methods is whether they deal with continuous or discontinuous changes of state. Queueing problems are typically treated with discontinuous simulation. Continuous simulation problems always have

a mathematical representation as a set of ordinary differential equations. For applications that call for both continuous and discontinuous simulation, combined methods have been developed. A number of simulation systems are commercially available, or may be employed at computer service centers by a remote user.

Optimization methods can help to support the synthesis part of design. In most cases, design optimization requires the solution of a non-linear constrained problem. A practical difficulty is the need for the formal statement of a measure-of-merit function and of the constraints. Many design requirements cannot be formally stated; or it might simply consume too much time or manpower to achieve a formal statement. Another problem is the large number of parameters influencing a design. For this reason, the application of optimization methods is more promising in an early design stage – on the basis of a crude model with only a few design parameters – than in final design, where the number of parameters becomes prohibitive because of the many details to be considered.

Numerical analysis methods inherently produce huge amounts of numerical data, which must be brought into a comprehensible form. Graphical representation techniques are well suited for this purpose. Functions of one variable are generally represented as diagrams. For functions of two variables, various techniques are possible; the usefulness of these techniques depends both on the behavior of the function and on the available graphical hardware. Computer-generated movies or videofilm, combined with color coding of the displayed information, is best suited for the visualization of complicated multi-dimensional results.

6.5 Bibliography

[AOKI71] M. Aoki: Introduction to Optimization Techniques. New York, Macmillan (1971).

[AYRE73] D. J. Ayres: Elastic-Plastic and Creep Analysis Via the MARC Finite-Element Computer Program. In: St. J. Fenyves et al., Numerical and Computer Methods in Structural Mechanics. New York, Academic Press (1973), pp. 247–263.

[BARR81] A. H. Barr: Superquadrics and Angle-Preserving Transformations. IEEE Comp. Graph. and Applic. 1, 1 (1981), pp. 11–23.

[BATH73] K. J. Bathe, E. L. Wilson, E. Peterson: SAP IV: A Structural Analysis Program for Static and Dynamic Response of Linear Systems. Report No. EERC-73-11, Berkeley, Univ. California Press (1973).

[BOBL76] P. A. Bobllier, B. C. Kahan, A. R. Probst: Simulations with GPSS and GPSS V. Englewood Cliffs, Prentice-Hall (1976).

[BOEH84] W. Böhm: A survey of curve and surface methods in CADG. CADG 1, 1 (1984).

[CATM78] E. Catmull, J. Clark: Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. Computer Aided Design 10, 6 (1978), pp. 350–355.

[CHRI74] H. N. Christiansen: Application of Continuous Tone Computer Generated Images in Structural Mechanics. In: W. Pilkey, K. Saczalski, H. Schaeffer, Structural Mechanics Computer Programs. Charlottesville, Univ. Virginia Press (1974), pp. 1003–1015.

[COHE80] E. Cohen, T. Lyche, R. F Riesenfeld: Discrete B-Splines and Subdivision Techniques in Computer Aided Design and Computer Graphics. Computer Aided Graphics and Image Processing 14, 2 (1980), pp. 87–111.

[COOL65] J. W. Cooley, J. W. Tukey: An Algorithm for the Machine Computation of Complex Fourier Series. Math. Comput. 19 (1965), pp. 297–301.

[COON67] S. A. Coons: Surfaces for Computer-Aided Design of Space Forms. Report Project MAC TR-41, Massachusetts Institute of Technology (1967).

[CSMP72] "CSMP-III": Continuous System Modelling Program III (CSMP-III) Program Reference Manual. IBM Form SH19-7001-2. Don Mills, IBM Canada Ltd (1972).

[CZAP76] H: Czap: Nichtlineare Optimierungsmethoden. In: H. Noltemeier (ed.), Computergestützte Planungssysteme. Würzburg, Physica-Verlag (1976), pp. 93–110.

[DAHL70] O.-J. Dahl, B. Myhrhaug, K. Nygaard: SIMULA 67 Common Base Language. Report S22, Oslo-Bergen-Trømso, NORSK Norwegian Computer Center (1970).

[DESI69] B. M. E. deSilva: The Application of Nonlinear Programming to the Automated Minimum Weight Design of Rotating Disks. In: R. Fletcher (ed.), Optimization. London, Academic Press (1969), pp. 115–150.

[DOO_78] D. Doo, M. A. Sabin: Behaviour of Recursive Division Surfaces Near Extraordinary Points. Computer Aided Design 10, 6 (1978), pp. 356–360.

[ENCA75] J. Encarnação: Computer Graphics. München, Oldenbourg (1975).

[ENDE78] G. Enderle, I. Giese, M. Krause, H.-P. Meinzer: The AGF-Plotfile – Towards a Standardization for Storage and Transportation of Graphics Information. Computer Graphics 12, 4 (1978), pp. 92–113.

[ENDE80] G. Enderle, K.-H. Bechler, F. Katz, K. Leinemann, W. Olbrich, E. G. Schlechtendahl, K. Stölting: GIPSY-Handbuch. Report KfK 2878, Kernforschungszentrum Karlsruhe (1980).

[ENDE81] G. Enderle, K.-H. Bechler, H. Grimme, W. Hieber, F. Katz: GIPSY-Handbuch Band II. Report KfK 3216, Kernforschungszentrum Karlsruhe (1981).

[FAUX83] I. D. Faux, M. J. Pratt: Computational geometry for design and manufacture. New York, John Wiley and Sons (1983).

[FENY73] St. J. Fenyves, N. Perrone, A. R. Robinson, W. C. Schnobrich: Numerical and Computer Methods in Structural Mechanics. New York, Academic Press (1973).

[FLET69] R. Fletcher (ed.): Optimization. London, Academic Press (1969).

[FORR68] A. R. Forrest: Curves and surfaces for computer aided design. University of Cambridge (1968).

[FORR72] A. R. Forrest: On Coons' and Other Methods for the Representation of Curved Surfaces. Computer Graphics and Image Processing 1, 4 (1972), pp. 341–359.

[GALL75] R. H. Gallagher, O. C. Zienkiewicz, M. Morandi-Cecchi, C. Taylor: Finite Elements in Fluids, Vols. 1, 2 and 3. Chichester, J. Wiley and Sons (1975/1975/1978).

[GANZ81] R. Ganz, H.-J. Dohrmann: Farbgraphische Ausgabesysteme. Zeitschrift für wirtschaftliche Fertigung 76, 5 (1981), pp. 223–239.

[GILO78] W. K. Giloi: Interactive Computer Graphics. New Jersey, Prentice-Hall (1978).

[GLOW79] R. Glowinski, E. Y. Rodin, O. C. Zienkiewicz: Energy Methods in Finite Element Analysis. Chichester, J. Wiley and Sons (1979).

[GOLD73] D. G. Golden, J. D. Schoffler: GLS – A Combined Continuous and Discrete Simulation Language. SIMULATION 20 (1973), pp. 1–8.

[GORD71] W. J. Gordon: Blending-Function Methods for Bivariate and Multivariate Interpolation and Approximation. SIAM J. Num. Anal. 8, 1 (1971), pp. 158–177.

[GORD74] W. J. Gordon, R. F. Riesenfeld: B-Spline Curves and Surfaces. In: R. Barnhill, R. F. Riesenfeld (eds.), Computer Aided Geometric Design. New York, Academic Press (1974), pp. 95–126.

[GORD75] G. Gordon: The Application of GPSS V to Discrete System Simulation. Englewood Cliffs, Prentice-Hall (1975).

[GORD78] G. Gordon: System Simulation. Englewood Cliffs, Prentice-Hall (1978).

[GOTT77] D. Gottlieb, St. A. Orszàg: Numerical Analysis of Spectral Methods: Theory and Applications. Philadelphia, Soc. for Industrial and Applied Mathematics (1977).

[GROS57] G. Grosche: Projective Geometrie, Teil 1 and 2. Leipzig, Teubner Verlagsgesellschaft (1957).

[HECK78] R. Heckler, H. P. Schwefel: Superimposing Direct Search Methods for Parameter Optimization onto Dynamic Simulation Models. In: H. J. Highland, N. R. Nielsen, L. R. Hull, 1978 Winter Simulation Conference. IEEE 78CH1415-9. New York, Institute of Electrical and Electronic Engineers (1978), pp. 173–181.

[HESS67] J. L. Hess, A. M. O. Smith: Calculation of Potential Flow about Arbitrary Bodies. Progr. Areonautical Sciences 8 (1967), pp. 1–138.

[HEUS70] G. Heusener: Optimierung natriumgekühlter schneller Brutreaktoren mit Methoden der nichtlinearen Programmierung. Report KfK 1238, Kernforschungszentrum Karlsruhe (1970).

[HIGH79] H. J. Highland, N. R. Nielsen, L. R. Hull (eds.): 1978 Winter Simulation Conference. IEEE 78CH1415-9. New York, Institute of Electrical and Electronic Engineers (1979).

[HORN81] C. Hornung: An Approach to a Calculation Minimized Hidden-Line Algorithm. In: J. Encarnação (ed.), EUROGRAPHICS '81. Amsterdam, North-Holland (1981), pp. 31–42.

[HUGH79] T. J. R. Hughes: Finite Element Methods for Fluids and Fluid-Structure Interaction. Preprints 1st Int. Seminar on Fluid Structure Interaction in LWR Systems. Report ISSN 0172-0465. Berlin, Bundesanstalt für Materialprüfung (1979), pp. 19–28.

[JOHN86] R. H. Johnson: Solid Modeling: A State-of-the-Art Report, 2nd ed., ISBN 0-444-87994-3. Amsterdam, North Holland (1986).

[KLEM86] K. Klement: Parametric rational curves of second degree. Extension of the surfaces of second degree. Jahresbericht der Fachgruppe GRIS, Fachbereich Informatik, TH Darmstadt (1986).

[KLEM87] Mathematische Grundlagen für CAD Systeme, K. Klement: Seminarunterlagen des ZGDV, Darmstadt (1987).

[KRIE80] R. Krieg, B. Göller, G. Hailfinger: Transient, Three-Dimensional Potential Flow Problems and Dynamic Response of Surrounding Structures. J. of Comp. Phys. 34, 2 (1980), pp. 164–183.

[KUEH79] D. Kühl, K. Leinemann: Erfahrungen mit dem Einsatz des AGF-PLOTFILE-Formats. Angewandte Informatik 9 (1979), pp. 401–405.

[KUEN67] H. P. Künzi, H. G. Tzschach, C. A. Zehnder: Numerische Methoden der mathematischen Optimierung. Stuttgart, Teubner (1967).

[LEIN80] K. Leinemann, P. Royl, W. Zimmerer: Integration graphischer Systeme für Darstellungszwecke. KfK-Nachrichten 1–2, Kernforschungszentrum Karlsruhe (1980), pp. 75–80.

[LIET55] W. Lietzmann: Anschauliche Topologie. München, Oldenbourg (1955).

[LUEN73] D. G. Luenberger: Introduction to Linear and Nonlinear Programing. Reading, Addison-Wesley (1973).

[MACN72] R. H. MacNeal: NASTRAN. In: Three-Dimensional Continuum Computer Programs for Structural Analysis. New York, ASME (1972), pp. 21–22.

[MESC66] H. Meschkowski: Grundlagen der euklidischen Geometrie. BI-Hochschultaschenbuch 105/105a. Mannheim (1966).

[MYER71] R. H. Myers: Response Surface Methodology. Boston, Ally and Bacon, Inc. (1971).

[NELS72] M. F. Nelson: ICES STRUDL II. In: Three-Dimensional Continuum Computer Programs for Structural Analysis. New York, ASME (1972), pp. 23–24.

[NEWM79] W. M. Newman, R. F. Sproull: Principles of Interactive Computer Graphics. London, McGraw-Hill (1979).

[NOOR79] A. K. Noor, H. G. McComb Jr. (eds.): Trends in Computerized Structural Analysis and Synthesis. Computers and Structures 10 (1979), pp. 1–430.

[NOWA73] H. Nowacki: Optimization in Pre-Contract Ship Design. Conf. Computer Applications in the Automation of Shipyard Operation and Ship Design. Tokyo (1973), pp. 327–338.

[NOWA80] H. Nowacki: Modelling of Design Decisions for CAD. In: J. Encarnação, Computer Aided Design: Modelling, Systems Engineering, CAD-Systems. Lecture Notes in Computer Science 89, Heidelberg, Springer-Verlag (1980), pp. 177–223.

[NOWC80] H. Nowacki: Curve and Surface Generation and Fairing. In: J. Encarnação (ed.), Computer Aided Design: Modelling, Systems Engineering, CAD-Systems. Lecture Notes in Computer Science 89, Heidelberg, Springer-Verlag (1980), pp. 137–176.

[PATT78] M. R. Patterson: CONTUR: A Subroutine to Draw Contour Lines for Randomly Located Data. Report ORNL/CSD/TM-59, Oak Ridge, Union Carbide Corp., Nuclear Division (1978).

[PEAR77] "PEARL": Full PEARL Language Description. Report KfK-CAD 130, Kernforschungszentrum Karlsruhe (1977).

[PILK74] W. Pilkey, K. Saczalski, H. Schaeffer: Structural Mechanics Computer Programs, Surveys, Assessments, and Availability. Charlottesville, Univ. Virginia Press (1974).

[PRIT75] A. A. B. Pritsker, R. E. Young: Simulation with GASP-PL/1. New York, John Wiley and Sons (1975).

[PUGH61] A. L. Pugh III: DYNAMO User's Manual. Cambridge, Mass., MIT Press (1961).

[REQU82] A. A. G. Requicha, H. B. Voelcker: Solid Modeling: A Historical Summary and Contemporary Assessment. Computer Graphics and Applications 2, 2 (1982), pp. 9–24.

[RIES81] R. F. Riesenfeld: Homogeneous Coordinates and Projective Planes in Computer Graphics. IEEE Computer Graphics and Applications 1, 1 (1981), pp. 50–55.

[RUOF74] G. Ruoff, E. Stein: The Development of General Purpose Programs and Systems. In: W. Pilkey, K. Saczalski, H. Schaeffer, Structural Mechanics Computer Programs. Charlottesville, Univ. Virginia Press (1974), pp. 703–719.

[RUSH67] B. C. Rush, J. Bracken, G. P. McCormick: A Nonlinear Programming Model for Launch Vehicle Design and Costing. Oper. Res. 15 (1967), pp. 185–210.

[SCHL70] E. G. Schlechtendahl: DYSYS, A Dynamic System Simulator for Continuous and Discrete Changes of State. Report KfK 1209, Kernforschungszentrum Karlsruhe (1970).

[SCHL81] E. G. Schlechtendahl: Graphische Datenverarbeitung in der Kerntechnik. In: J. Encarnação, W. Straßer (eds.), Geräteunabhängige graphische Systeme. München, Oldenbourg (1981).

[SCHR72] E. Schrem: ASKA. In: Three-Dimensional Continuum Computer Programs for Structural Analysis. New York, ASME (1972), pp. 31–34.

[SCHU76] R. Schuster: System und Sprache zur Behandlung graphischer Information im rechnergestützten Entwurf. Report KfK 2305, Kernforschungszentrum Karlsruhe (1976).

[SCHU78] U. Schumann (ed.): Computers, Fast Elliptic Solvers, and Applications. London, Advance Publ. (1978).

[SCHW77] H. P. Schwefel: Numerische Optimierung von Computer-Modellen mittels Evolutionsstrategie. Basel, Birkhäuser (1977).

[SHAN75] R. E. Shannon: Systems Simulation: The Art and Science. Englewood Cliffs, Prentice-Hall (1975).

[SUTH74] I. E. Sutherland, et al.: A Characterization of Ten Hidden-Surface Algorithms. ACM Computing Surveys 8 (1974), pp. 1–55.

[SVAN81] K. Svanberg: Optimization of Geometry in Truss Design. Computer Methods in Appl. Mechanics and Engineering (1981), pp. 63–80.

[SWAN00] J. A. Swanson: ANSYS – Engineering ANalysis SYStem Users' Manual. Elizabeth, Swanson Analysis Systems, Inc. (no date).

[SYN__66] W. M. Syn, R. N. Linebarger: DSL/90 – A Digital Simulation Program for Continuous System Modeling. Spring Joint Computer Conference (1966).

[TILL83] W. Tiller: Rational B-splines for curve and surface representations. San Diego, Calif. Structural Dynamics Research Corporation (1983).

[TILO80] R. B. Tilove: Set Membership Classification: A Unified Approach to Geometric Intersection Problems. IEEE Transactions on Computers C-29, 10 (1980), pp. 874–883.

[VOEL78] H. B. Voelcker, A. A. G. Requicha: Boundary Evaluation Procedures for Objects Defined via Constructive Solid Geometry. Tech. Memo No. 26. Production Automation Project, University of Rochester (1978).

[WILD67] D. J. Wilde, C. S. Beightler: Foundations of Optimization. Englewood Cliffs, Prentice-Hall (1967).

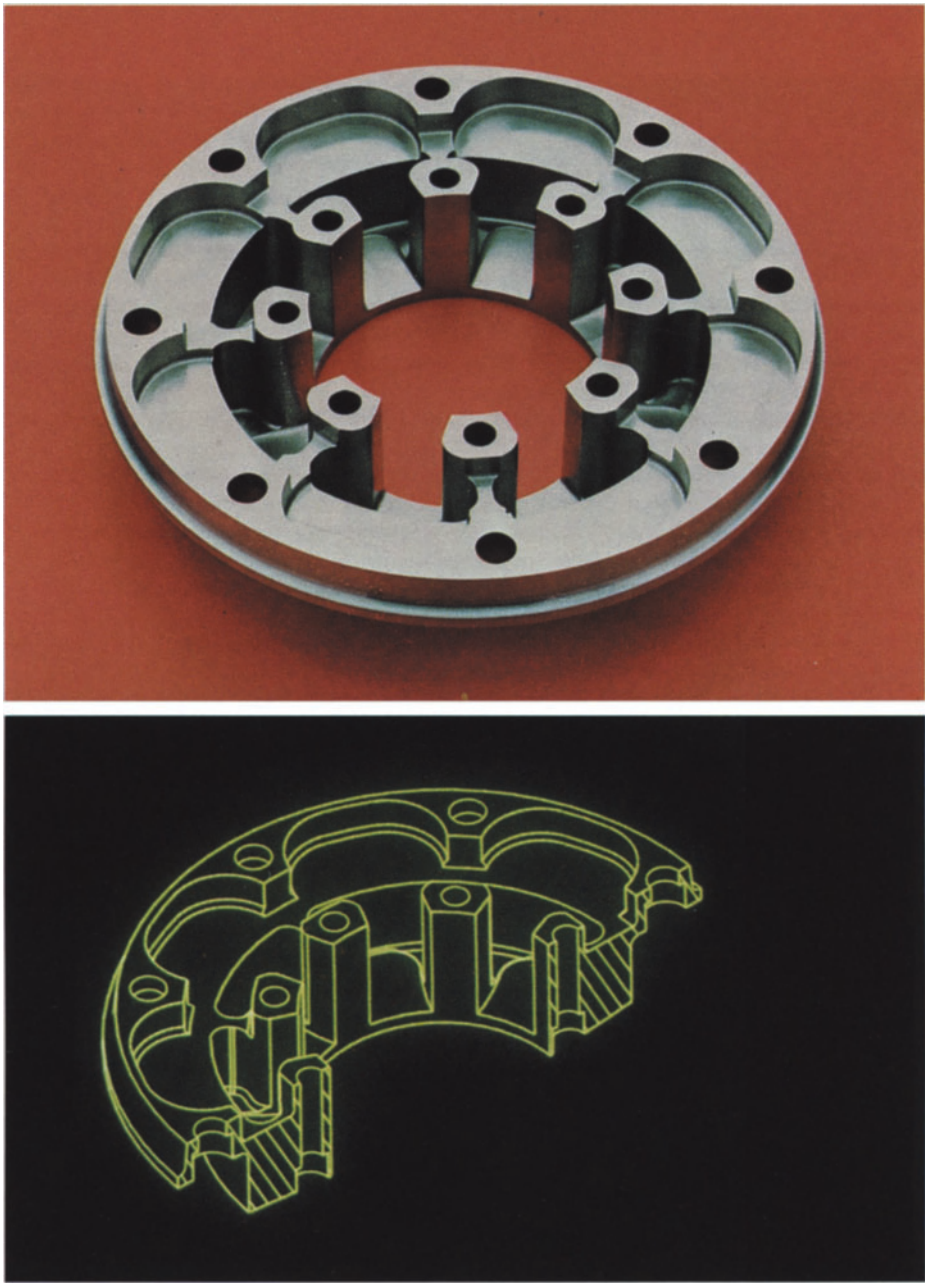
[ZIEN77] O. C. Zienkiewicz: The Finite Element Method, 3rd edition. London, McGraw-Hill (1977).

[ZIEN79] O. C. Zienkiewicz, D. W. Kelly, P. Bettes: Marriage a la Mode – The Best of Both Worlds (Finite Elements and Boundary Integrals). In: R. Glowinski, E. Y. Rodin, O. C. Zienkiewicz, Energy Methods in Finite Element Analysis. Chichester, John Wiley and Sons (1979), pp. 81–107.

[ZIMM75] W. Zimmerer: PLOTCP – Ein FORTRAN IV-Programm zur Erzeugung von Calcomp Plot-Zeichnungen. Report KfK 2081, Kernforschungszentrum Karlsruhe (1975).

[ZOUT60] G. Zoutendijk: Methods of Feasible Directions. Amsterdam, Elsevier Publ. Co. (1960).

7 CAD Data Transfer



Two different presentation modes for the solid model of a transmission (courtesy of Evans & Sutherland, Salt Lake City, USA)

7.1 Introduction

In the 1980's a new issue became important in CAD: the transfer of CAD data [ENCA86]. This resulted from two developments:

- CAD systems had been introduced into the industrial practice to the extent that significant amounts of design data were now stored in CAD data bases. It turned out that in all but the smallest design environments no single CAD system could cover the design needs adequately. Many industrial products require, e.g., both mechanical and electrical design or both design of solid mechanical engineering parts and sculptured surfaces. Thus different CAD systems had to be used for designing different aspects of the same industrial product. This situation has not changed until to date and is unlikely to change in the future, as CAD systems will continue to exhibit differences in functional performance and efficiency depending on the emphasis which their vendors place on the different application areas.

There is an obvious need to communicate design results between the different systems involved in order to ensure consistency. Even more so, for the design of larger products (an automobile, for instance) the design is not restricted to one company (the car manufacturer itself), but many suppliers have to design their parts such that they fit into the final product. Hence, there is a need for communicating design data between different companies. The need for communicating between design processes being executed in separate design environments was already mentioned in Sects. 3.1 and 4.1.2.5.

Communication of design data is also required for a different reason: the completed design must be communicated to other processes (see Fig. 3.3) such as:

- analysis,
- production planning,
- manufacturing (NC, robotics),
- quality assurance,
- maintenance, and
- marketing.

The term CIM (computer-integrated manufacturing) has become a buzz-word in the late 1980s for the attempts to connect properly the various computer support systems used in these various areas into a well integrated system in which all information, once it is produced, becomes immediately available at all places where it is needed.

7.2 The Principle of Neutral Files

The basic idea of CAD data transfer is the concept of a neutral file. "Neutral" means that the file is written in a format that is independent from the many different formats utilized by the various CAD system vendors. This concept reduces the amount of processor development from an effort that is proportional to the square of the number

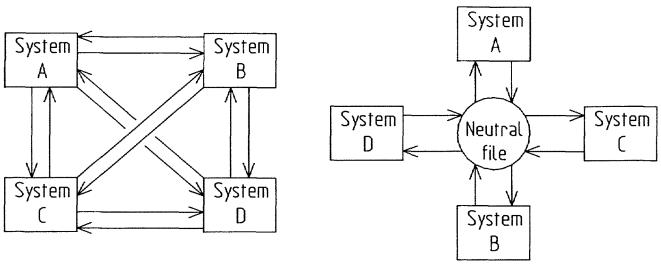


Fig. 7.1. The number of processors required for CAD data transfer grows quadrically fro bilateral exchange and only linearly with the neutral file solution

of systems involved to one that depends only linearly on the number of systems (see Fig. 7.1). The number of pre-processors (programs that write the CAD database contents to a sequential file) and post-processors (programs that interpret the file and reconstruct the CAD database contents) is reduced from $N \cdot (N-1)/2$ pairs of such processors to only N pairs (if N is the number of CAD systems)!

This line of argument does not say that a set of bilateral pre- and post-processors (using special interface formats for every pair of CAD systems) should never be considered as a practical solution. In fact, if we have a closed design environment with only a small number of systems involved (2, 3, or perhaps even 4) then the special solution may even be more efficient. But it is a solution that does not lend itself easily to future growth.

7.3 History of CAD Data Exchange Standards and Neutral Format Proposals

Industrial application of computer-aided design originally concentrated mainly on drafting (two-dimensional representations) and on approximate representation of three-dimensional objects with wireframe models. In some areas (in the automotive industry, aircraft and spacecraft design, and in shipbuilding) modeling techniques for sculptured surfaces grew up rapidly. Solid modeling became important more recently, in particular in the mechanical products industry and in plant layout, as collision analysis and transfer of CAD information to manufacturing were introduced into the industrial practice.

The history of CAD data exchange parallels this development. Originally, transfer of drawings was satisfactory. It is not surprising that the most widely used standard was called IGES = Initial Graphics Exchange Specification, with emphasis on the graphics aspect [IGES81]. Two-dimensional and three-dimensional line geometry, and to some extent also non-geometrical information can be transferred with IGES 2.0 (which never became an official standard as its predecessor IGES 1.0 and its successor IGES 3.0, but which was used as a reference by all implementers). IGES-based CAD data transfer was never very satisfactory, but became indispensable due to the lack of anything better for most practical applications. Another standard for transfer of

similar information is SET [SET_85]. Transfer of surface information has also become feasible. It is now covered in IGES [IGES83], SET, and the VDA-FS [VDA_86] standard. Transfer of solid model data, however, is still a matter of research and development.

It has become evident in the past years that there is a need for an integrated mechanism for transfer of all product information. The international standard STEP which is presently being developed by the international standardization group ISO/TC184/SC4/WG1 is to provide this integrated facility. Important specifications of neutral file representations for CAD data:

name		origin	publisher	status	year
Y14.26M-81		USA	IGES/ANSI	Standard replaced by IGES 3.0	1981
IGES	Vers.2.0	USA	NBS	Specification	1983
VDA_FS		Germany	DIN	Standard	1983
PDDI		USA	NBS	Proposal	1984
IGES	Vers.3.0	USA	NBS	Standard	1985
CAD*I	Vers.2.1	Europe	ESPRIT	Specification	1985
SET	Vers.1.1	France	AFNOR	Standard	1986
CAD*I	Vers.3.2	Europe	ESPRIT	Specification	1987
SET	Vers.2.0	France	AFNOR	Proposal	1987
PDES		USA	NBS	Proposal	in progress
STEP		intern.	ISO	Proposal	in progress

7.4 IGES

7.4.1 Development and Content

The development of IGES was originated in 1979 by the US National Bureau of Standards (NBS) as an initially small effort which was funded by the US Air Force [LIEW85]. After two critical reviews the IGES draft was first published in 1980. It contained geometry, graphical data, and annotations for drafting applications. The growing interest and the rapidly increasing number of volunteers soon allowed other technical areas to be covered. The first official standard was published in five parts in 1981 [IGES81]. The fifth part was based on work done by McDonnell Douglas Automation for CAM-I.

The elementary building blocks of IGES are called "entities". Thirty-four different types of entities (with 41 sub-types) were defined in IGES 1.0. They were classified as

- geometry entities,
- annotation entities, and
- structure entities.

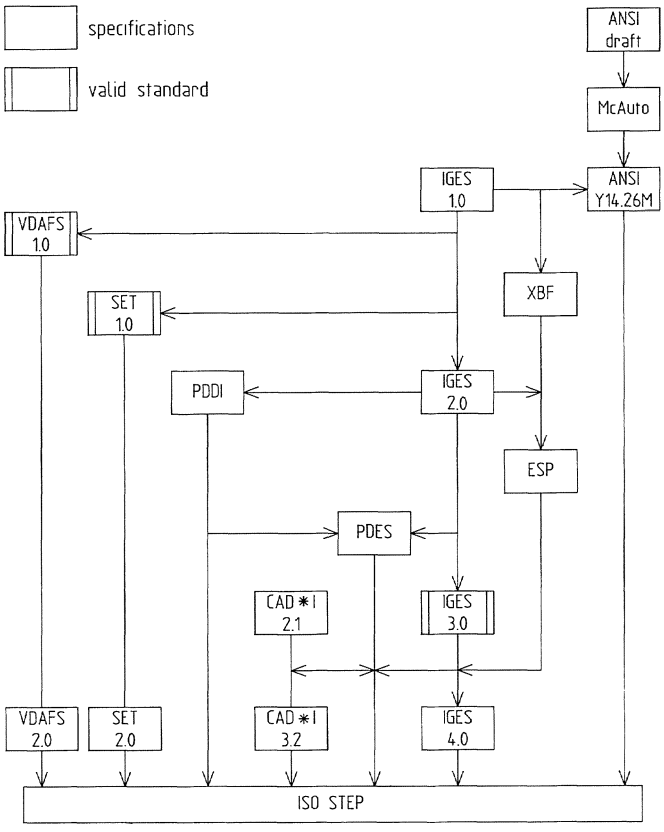


Fig. 7.2. History and interdependence of neutral file proposals for CAD data transfer

In the 1983 revision of IGES [IGES83], entities from the finite element analysis and the electrical systems applications were added. IGES 3.0 was expanded to include over 50 entity types in the areas of

- geometry,
- annotation,
- piping and electronics schematics,
- network subfiguring,
- finite element modeling, and
- external file handling.

IGES 4.0 was issued in 1989 and (among other enhancements) contains solid modeling representations of the constructive solid geometry type. Boundary representations are planned for IGES 5.0.

START SECTION	iges file data source = cd / 2000								s	1
	cd / 2000 part name =								s	2
	igestest								s	3
GLOBAL SECTION	.;7Hcd / 2000.4higes.8hcdc v1.4.3h1.0.60.10.48.10.96.7hcd / 2000.1.00000.2.								g	1
	2hmm.1.0..13h831019.112333..100000e-003..100000e+40;								g	2
DIRECTORY ENTRY SECTION	124	1	1	0	0	0	0	0000000	d	1
	124	0	0	1	0	0	0matrix		1d	2
	212	2	1	1	0	0	1	0000001	d	3
	212	0	0	2	1	0	0gnote		1d	4
	116	4	1	1	0	0	1	0000000	d	5
	116	0	0	1	0	0	0point		1d	6
	100	5	1	1	0	0	1	0000000	d	7
	100	0	0	1	0	0	0circ arc		1d	8
PARAMETER DATA SECTION	124,1.0,0..0..0..0..1.0.0..0..0..0..1.0.0..0.0;								1p	1
	212,1.33,115.253,3.175,1.1.57080.0..0.0,117.883,208.053,0..33h								3p	2
	iges - entities: punkt,kreis,note,0;								3p	3
	116,175.940,144.372,0..0.0.0;								5p	4
	100,0..176.122,143.646,201.122,143.646,201.122,143.646,0.0;								7p	5
TERMINATE SECTION	s 2g 4d 7p 5								t	1

Fig. 7.3. A fixed format IGES file

7.4.2 Format

An IGES file is a sequence of 80-character records written in fixed format or (since IGES 3.0) in a compressed format. The whole file consists of 5 main sections (see Fig. 7.3):

- the *start section* contains a text describing the contents of the file;
- the *global section* contains data to be used by the post-processor for the interpretration of the file (such as the delimiters and precision of real number representations);
- the *directory entry section* consists of two records per entity written in fixed format with data that does not depend on the entity type (version identification, attributes, pointers);
- the *parameter data section* contains the parameters for all entities in free format;
- the *terminate section* is a single record containing the numbers of each record in each section.

The directory entry section and the parameter data section were combined into a single *data section* in the compressed format provided by IGES 3.0 (Fig. 7.4).

7.4.3 Experiences

IGES has become a useful tool in practical CAD. In fact, it has become so indispensable that a CAD system which cannot interface with IGES files will be in serious difficulty on the market. However, the history of IGES application is full of trouble. The main problem areas were the following [GRGL86], [WEIS85]:

FLAG SECTION		c	
START SECTION	iges file data source = cd/2000	s	1
	cd/2000 part name =	s	2
	igestest	s	3
GLOBAL SECTION	.;7Hcd/2000.4higes.8hcdc v1.4.3h1.0.60.10.48.10.96.7hcd/2000.1.00000.2.	g	1
	2hmm.1.0..13h831019.112333..100000e-003..100000e+40;	g	2
DATA SECTION	d1@1_124@14_1		
	124.1.0.0..0..0..0..1.0.0..0..0..0..1.0.0..0.;		
	d2@1_212@4_1@9_1@14_2		
	212.1.33.115.253.3.175.1.1.57080.0..0.0.117.883.208.053.0..33h		
	iges - entities: punkt.kreis.note.0;		
	d3@1_116@4_1@14_1		
	116.175.940.144.372.0..0.0.0;		
	d4@1_100@4_1@14_1		
	100.0..176.122.143.646.201.122.143.646.201.122.143.646.0.0;		
TERMINATE SECTION	s 2g 4d 7p 5	t	1

Fig. 7.4. A compressed format IGES file

- Data types:
  The data structures which the different CAD systems use to store CAD data are normally different from the ones that underlie IGES. Hence, the implementers of pre- and post-processors are left with the task of mapping from and to their data representations. This often boils down to an approximation. Proper transfer of data between systems will now depend on whether the implementers of the processors use the same kind of mapping or approximation. As far as geometry is concerned the common understanding all over the world is greatly supported by the fact that mathematics is a precise discipline. Other types of data, especially text, annotations, and dimensioning are based more on tradition than on formal theories. Hence, it is not surprising that entities of these kinds cause the most problems in CAD data transfer.
- Processors:
  A thorough investigation of IGES processors in 1983 [BOOZ83] indicated that only 27% of the potentially applicable IGES entities were supported by the processors of 12 leading CAD systems. One reason for this development was that most CAD system vendors were under high pressure from their major customers to implement those entities which these customers needed. The result was that the intersection of all the subsets of IGES entities which were supported by all systems became rather small. Instead of leaving the selection of the supported subset of IGES entities to the choice of the implementer, in 1987 allowable subsets were defined, e.g., by the German automotive industry as a whole. The VDA-IS (IS = IGES subset) defines two upward-compatible subsets of geometric entities and also two upward-compatible subsets of annotation entities (see Fig. 7.5). Thus, any IGES processor can now be classified by the subsets which it supports *entirely*.

The quality of the processors, especially in the early years, was rather poor. Files had syntactical errors, illegal or inconsistent pointers, incorrect numerical values, or

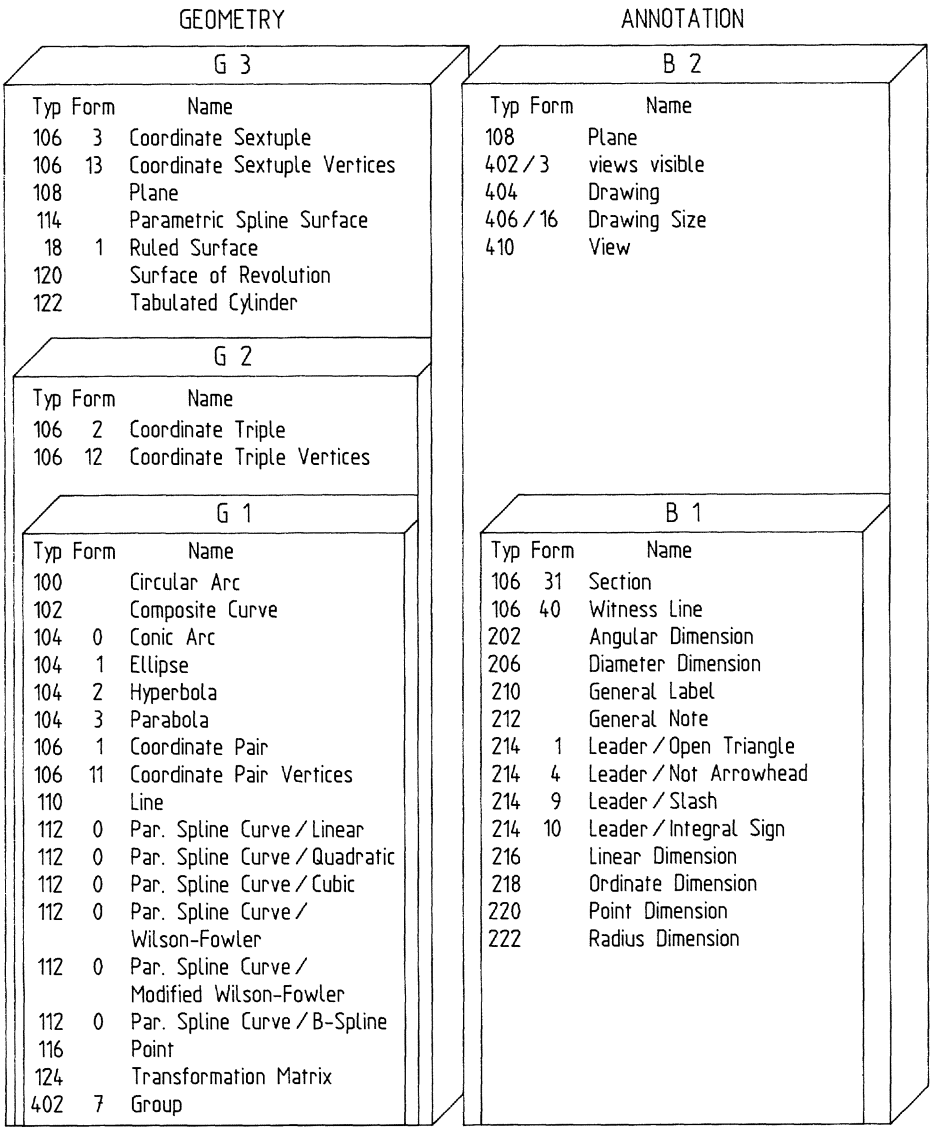


Fig. 7.5. The IGES subsets defined by VDA

private extensions which were useful for data exchange in and out of the same system but not between different systems. Testing IGES processors became a discipline in its own right [GRAB86]. The principal test methods are (see Fig. 7.6):

- The cycle test:
  CAD database contents are converted into an IGES file. This file is then used to reconstruct (a twin of) the same database in the originating system. The two databases are then compared.

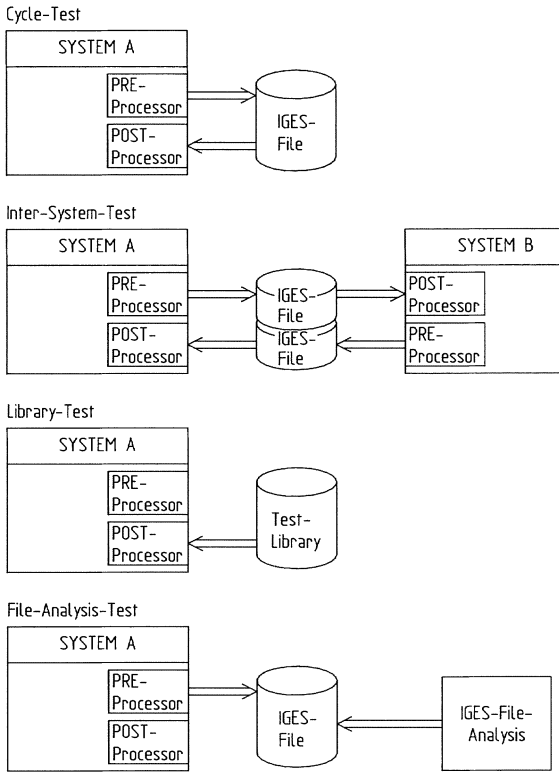


Fig. 7.6. Test methods for testing pre- and post-processors

- The inter-system test:
  The database contents are transferred into another system and back into the original one. Thus, two pairs of pre- and post-processors are involved. Then the two databases are compared.
- The library test:
  An IGES file which is known to be correct is converted into the CAD system's database. This type of test validates the post-processor only.
- The file analysis:
  Special IGES file analyzer software has been developed which checks an IGES file for most types of errors. This test validates the pre-processor only.

7.5 SET

Inadequacies of the IGES file format and the IGES processors led Aérospatiale to develop a new standard format for approximately the same application domain as IGES: Standard d'Échange et de Transfert (SET) [SET_85]. The data model is similar to IGES but the format is very much different. SET has become widely used in the European Airbus industry and has started to spread into other domains. File

size reductions by a factor of 80 and processing time gains by a factor of three have been quoted compared to IGES [CLAU85].

The following differences in the SET approach relative to IGES are worth mentioning:

- the goal to represent in the SET file the internal representation of the sending CAD system as accurately as possible. If any transformations are to be made they are to be done by the receiving system;
- the block structure (in two levels). Certain information is valid only in a block or sub-block. The semantic meaning of data is influenced by the block or sub-block type where it occurs;
- the directory concept. Information that will be required in many places is grouped into a directory and referenced where needed instead of being repeatedly represented on the file; and
- the free format instead of the card image format.

7.6 VDA-FS

In contrast to IGES and SET, VDA-FS [VDA_86] has a more restricted scope: it is specifically aimed at sculptured surface applications. One reason for its development was the limitation of IGES free-form surface representations to third-degree polynomials. The VDA-FS entities are:

- point,
- point sequence,
- point vector sequence,
- curve (polynomial representation), and
- surface (polynomial representation, topologically rectangular patches).

With BEGINSET and ENDSET, a grouping mechanism is provided in VDA-FS.

Recently, an updated version 2.0 of VDA-FS has been proposed. The principal extensions are: circle and circular arc, curves which are defined in the parametric space of a surface (instead of world coordinates) – the curves-on-surface. Other capabilities are composite surfaces built from regular or irregular patches of surfaces, a general grouping mechanism, and transformation matrices which define transformed instances of already defined geometry.

7.7 Proposals for Solid Model Transfer

The development of solid model interfaces is strongly influenced by CAM-I (Computer-Aided Manufacturing – International Inc., see Fig. 7.2). CAM-I gave a contract to McDonnell Douglas Automation (McAuto) to prepare a file specification for solid models. That effort was completed in 1979. Part of that specification was cast

in the IGES format and published as "Section 5 – Basic shape description" of ANSI-Y14.26M (IGES1.0) [IGES81].

7.7.1 IGES Section 5 – Basic Shape Description

Due to the fact that the basic shape description was a result of the McAuto work, section 5 was completely independent of the first four sections of IGES 1.0. It contained its own geometry definition for B-rep solid models, though most of the basic geometry was already defined in the first four chapters of ANSI-Y14.26M. The only link was the use of the same basic file format.

The basic shape description relied on a relational table format which was represented in the IGES format by the Relation Entity (entity number 500) and the Domain Structure Entity (entity number 502). Any structure for basic shape description is defined by using either of those two entities.

A model was composed of three classes of entities:

- geometric entities (curves, surfaces),
- topological entities (vertex, edge, loop, etc.), and
- miscellaneous entities (group).

There are no indications so far that this basic shape description has ever been used for solid model transfer.

7.7.2 Experimental Boundary File (XBF)

In parallel with the work at McAuto, CAM-I launched a project to develop a neutral subroutine interface to a modeler. The "Applications Interface Specification" (AIS) was developed by Ian Braid of Shape Data Ltd. and finished in 1980. It consisted of about 150 subroutine calls, which allow access to both CSG and B-rep modelers. The AIS has since been restructured and enhanced a number of times. The latest version was released in January 1986.

With the experience of these two specifications CAM-I decided to produce a new specification, the "Experimental Boundary File" (XBF). The first version of XBF was published in 1981 and a revised version in 1982 [CAMI82]. XBF uses the IGES format as a physical file format. The topological and CSG entities are defined as new IGES associativities (entity number 402 with form numbers 100–110) and the geometric entities as new IGES entities (entity numbers 700–744).

XBF supports both B-rep and CSG models. The B-rep model is based on the topological structure consisting of vertex, edge, path, face, shell and object with associated geometry. That structure allows for backpointers to entities on the next higher level in the topological structure, e.g., back from the path to the face. A path is an ordered list of edges. If the path is closed and bounds a region of a surface, it corresponds to the loop. However, the specification allows for a generalization of that structure by introducing dummy faces with no associated geometry. This is a means to represent special cases of solid models such as sheet objects with associated thickness or graphs of piping systems with associated cross section.

XBF provides for CSG models the Boolean operations Union, Intersection and Complement and a set of CSG primitives:

- Cuboid,
- Circular and elliptical cylinder,
- Circular and elliptical cone,
- Sphere,
- Ellipsoid,
- Pyramid,
- Prism,
- Torus, and
- Halfspace.

Similar to the "Basic Shape Description", XBF contains its own set of geometric entities ranging from analytical to parametric geometry. The objective was to avoid one of the weaknesses of the original IGES by allowing for symbolic referencing instead of comparing numeric values. For that reason the geometry is defined as far as possible by some basic building blocks such as Point, Direction, and Local Coordinate System.

An implementation of XBF was done at Lucas Industries Ltd. to prove its viability together with the viability of the AIS. Cycle tests have been performed with, e.g., the MBB test part (see Fig. 7.10). Already, these first tests showed that the efficiency of the translation of an XBF file was rather poor. The post-processor spent most of the time searching through the pointer structure in the PD and the DE sections of the IGES file. In addition, the file format was rather space-consuming because it was initially tailored to fulfill the needs of IGES. The solid model data used, e.g., only six of the 20 fields in a directory entry.

7.7.3 The IGES Experimental Solids Proposal (ESP)

In 1983 the IGES community felt the need to enhance the IGES specification with the capabilities to transfer solid models. The ESP was specified on the basis of XBF. It supports B-rep and CSG models [ESP__84].

The ESP lost the generality of the topological structure defined in the XBF, i.e., it does not provide for sheet and graph objects. As a new feature, the Face-set and the Edge-set entities were added to define collections of faces and edges and attach additional information to them.

The curves and surfaces geometry is (except for some minor differences) the same as in XBF.

On the CSG level some primitives of lesser significance such as Elliptical Cone were removed and some new ones such as Solid of Linear Extrusion were added.

Those entities were embedded in the IGES physical file format as separate entities and no longer as associativities as in XBF.

In 1984 the ESP was frozen for a testing phase of two years. After that period it turned out that no testing of the B-rep part had been done. The CSG part has been successfully tested in an exchange between PADL-2 at Ford Motor Company, GMSolid at General Motors and TRUCE at General Electric.

7.7.4 IGES 4.0

After that successful testing the CSG part of ESP was used for the enhancement of IGES 4.0 [IGES86]. The following CSG entities are included in IGES 4.0:

Primitives:
- Block,
- Right Angular Wedge,
- Right Angular Cylinder,
- Right Circular Cone Frustum,
- Sphere,
- Torus,
- Solid of Revolution,
- Solid of Linear Extrusion,
- Ellipsoid,
- Boolean Tree,
- Solid Instance, and
- Solid Assembly.

7.7.5 The Product Data Definition Interface (PDDI)

The PDDI effort initiated by the US Air Force is intended to provide a complete computer-based product-definition interface between design and manufacture. Its approach is somewhat similar to that of the XBF and the AIS in that way that it supports both a static file and a "working form" as access medium for application programs. PDDI comprises entities for geometry, topology, tolerances, part features and part control information. As it is the result of a military project it is not available for commercial use; however, it has occasionally been demonstrated in public [WILS85].

7.7.6 SET Solids Proposal

In 1987, Aérospatiale published a proposal for transfer of solid models to be included in the French SET standard. The proposal covers both CSG and B-rep models [SET__87].

7.8 CAD*I

In 1984 the Commission of the European Communities began to fund Project 322 of ESPRIT (European Strategic Program for the Research and development in Information Technologies). The five years project, called CAD-Interfaces (CAD*I), was to develop a set of standard CAD interfaces and to contribute to national and international standardization efforts [BEY__85], [BEY__86]. The project team consisted of twelve partners from six European countries.

Two of the project's working groups were concerned with the specification of neutral file formats for CAD applications: geometry and finite element data.

This section describes the overall concepts underlying the specification of a neutral file for exchange of CAD models between CAD systems or a CAD system and other application areas [SCHL86], [SCHL87].

Formal languages are used in CAD*I to specify the reference schema for CAD data structures. The high level data specification language (HDSL) has similarities with Pascal record declarations and has evolved from an early version of a specification language in PDES (see below). For the file format a Backus-Naur form is used; it can be generated automatically from the HDSL representation.

Entities are the basic data structures of the schema. They are defined by structural collections of one or more data attributes. An attribute provides details of part of an entity description. There are mainly two types of attributes, predefined and composite. Predefined attribute types include integer, real, logical, and string.

An essential feature of entities is that they may be referenced. All relationships between entities are expressed by references. The semantic difference between data and references to data is an essential part of the specification. Whether a relationship between two entities A and B is expressed as a reference from A to B, or from B to A, or as a separate link which refers to both A and B is determined by a certain anticipated behavior of the entities. Attributes which may or may not be associated with an entity are attached to the entity as "property". Rules for the binding of these references have been established in accordance with Sect. 4.2.2. A new feature of the CAD*I reference model is the "scope". It introduces the concept of a block structure into data structure specifications, in a similar way to that used in structured programming languages. The principle allows the "localized" definition of entities within some enclosing entity (see Fig. 7.7). A scoped entity enables a large collection of data to be addressed at a higher level as a single unit with a single name (see Sect. 3.3.1.3).

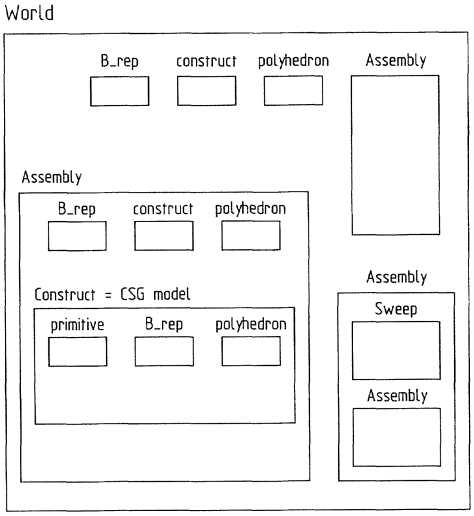


Fig. 7.7. The scope aspect of the CAD*I reference schema

The main significance of this is in terms of functional behavior of the data structure. Deletion of an entity which has other entities in its scope means that all the enclosed entities are deleted at once. Similarly, passing an entity with a scope to a modeling function means that this function has access to all the entities in that scope. Updating an entity means that any data inside the scope of that entity may be changed, but no data outside that scope. References are not allowed to entities defined within the scope of another entity. Consequently, entities which need to be "shared" (multiply referenced) by a number of scoped entities, need to be defined within a scope common to all these entities.

The concepts of entity, reference, scope, and property, together with the specification of how some basic operations interact with them provide the means for not only specifying static aspects of CAD data structures but also (some of) their behavior when used interactively by a CAD system operator. These operations are:

- enter (the scope of) an entity in order to be able to operate on the entities in the scope of that entity;
- interrogate the visible environment with respect to what entities and properties of which type are available;
- identify entities in the visible environment by textual means (i.e., by typing in the user-defined name of that entity);
- identify entities in the visible environment by non-textual means (i.e., by picking the corresponding graphical representation on the display screen);
- identify properties (or relations) by identifying the type of property and the related entity (entities);
- create a new entity;
- create a new property (or relation);
- delete an existing entity;
- delete an existing property and relation;
- modify the values of the attributes which constitute an entity or property result;
- perform modeling operations by invoking CAD system modeling functions and passing entities from the visible environment to these functions (the modeling functions change the database contents while the evaluating functions leave it unchanged);
- evaluate the visible model by invoking evaluation CAD system functions and passing entities from the visible environment to these functions;
- perform linear transformations (these are a special and most often used modeling operation).

The essential constituents of the CAD*I reference model are:

- a hierarchical grouping of components in assemblies, which again may be contained in assemblies up to the level of "world" (which represents a complete database);
- different geometric representations of the components (wire-frame models, surface models, solid models which include constructive solid geometry, boundary representation, and hybrid combinations);
- instancing and placement of entities allow the repeated use of a single template model for any number of occurrences;

- mechanisms for various kinds of external references either to data that existed already in the receiving CAD system, or to libraries (typically containing standard parts);
- parametric models:
  The term "parametric models" expresses a capability of certain CAD systems which allows the CAD system user to assign values to elementary data of some predefined type with the consequence that this will influence the CAD model in a certain way. A typical example is the length of some dimension in a model that is defined not as a constant but as a variable which has some value. Assignment of a new value of this length will cause a redefinition of all geometry that depends on this variable. The reference model allows the use of either constant values for predefined types, or references to entities of predefined type.

  A macro is similar to a parametric model in that it contains references to entities which may be redefined by the CAD user with the effect of creating a new geometry dependent on such changes. However, while parametric models always have values assigned to their variables at the time of sending, and may constitute an integral part of some larger CAD model, macros have to be invoked explicitly by the user. The result of such invocation has to be built into the overall model by the user. This result will then no longer depend on the macro but will exist on its own in the CAD database. A routine behaves similar to the macro, but it contains executable code instead of a parametric data structure;

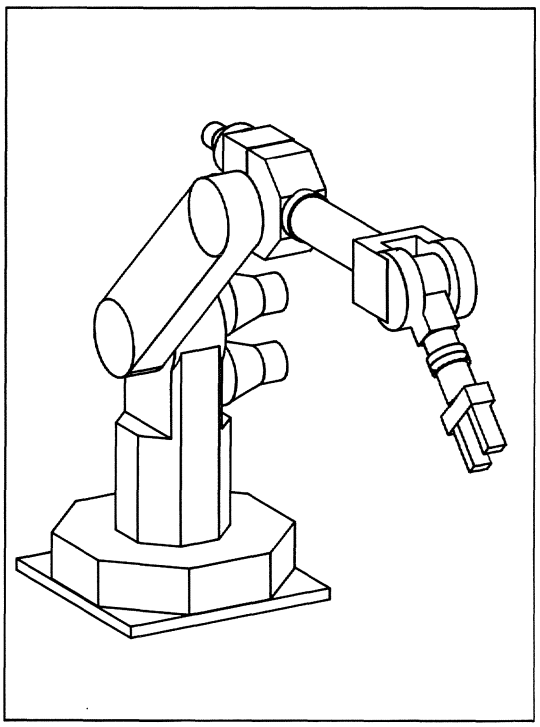


Fig. 7.8. Example of a CSG model transfer from Euclid to Technovision via Bravo (robot model)

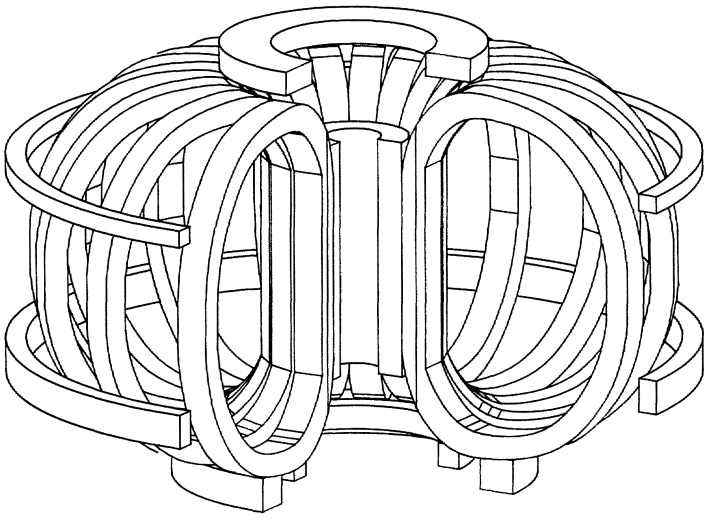


Fig. 7.9. Example of a CSG model after a cycle test with Bravo (Fusion reactor)

- user records:
  These provide a standard way for escaping from the restrictions of the schema when needed. User records are not semantically known in the reference model. They can be accommodated there as a means of storing information that is to be associated with entities, but that is to be interpreted only by special programs. User records may contain data of predefined type only. The record structure may be controlled by a special user-type record;
- a general grouping mechanism for information to be grouped together under various aspects; and
- a standard mechanism to add to the geometrical model test information that may be used in the receiving system for checking the accuracy of CAD data transfer.

The physical file represents a strictly sequential process that can build up the CAD data structure in the receiving system from a single parse of the neutral file. The consequence of this principle is that no forward references are allowed on the file. Both architectural reasons and efficiency considerations have influenced this choice.

The file format supports the block structure that corresponds to the scope aspect of entities. The purpose of the block structure is to localize, on the file, all information which fully defines a certain entity.

All CAD*I neutral files follow the envelope concept. The beginning and the end of the neutral file is clearly marked inside a larger CAD*I metafile. The metafile may carry additional information such as letters, FORTRAN source code, even CAD information according to some other standard such as IGES, VDA-FS, or SET. No conflict arises between these different files even when they are transported as one sequential file on magnetic tape or computer network.

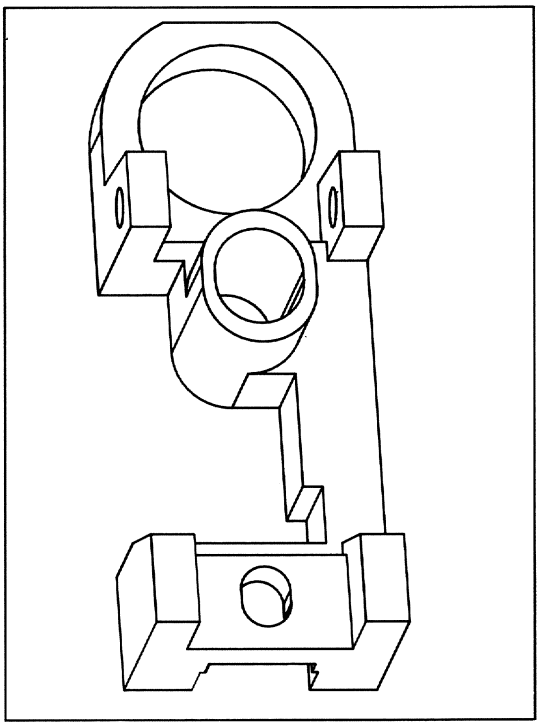


Fig. 7.10. Example of a B-rep model after transfer from Romulus to Technovision (MBB part)

Figures 7.8 through 7.11 show examples of solid models that have successfully been transferred between different CAD systems in the CAD*I project.

Details of the CAD*I specification for finite element analysis data are given in [MAAN87].

7.9 PDES

The term PDES (product data exchange specification) stands presently for three different, yet interrelated activities:

- The development of an ANSI standard for product data exchange. This project is being pursued by the IGES/PDES community and aims at replacing IGES at some time in the future by a better standard that is no longer burdened by the need to maintain compatibility with the previous IGES versions.
- The contribution of the US to the development of the international (ISO) standard STEP; these two activities were, so far, maintained in close agreement.
- A project of the US Air Force to develop a product model standard for its needs.

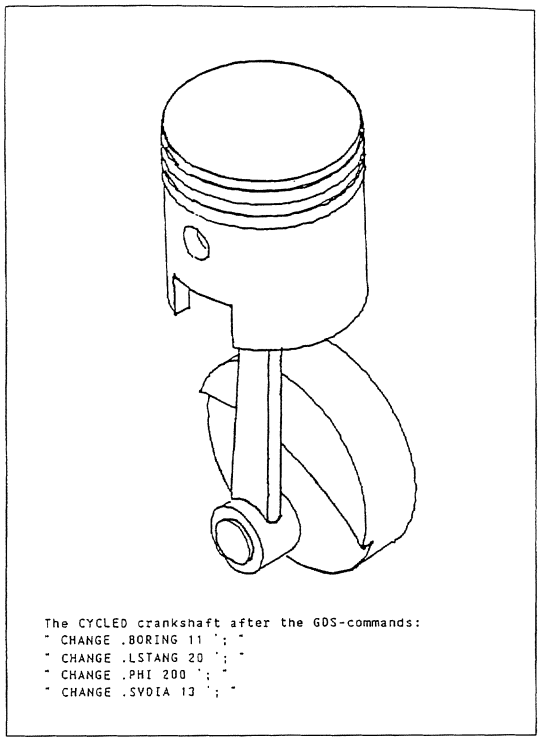


Fig. 7.11. Example of a parametric CSG model which maintained its parametric behavior after a cycle test with GDS

Whether these three activities will end up at a common result or whether conflicting interests will produce separate specifications is one of the most interesting developments in CAD to be watched in the next future.

One of the most important features of the PDES approach is the separation of the whole problem into three layers:

- the application layer, where experts from each application area are supposed to come up with a unified information model of their domain of expertise. This information model is to be represented in a semi-formal graphical form (called IDEF1X) and submitted to the logical layer experts;
- the logical layer, on which other experts shall attempt to combine the various application layer models to eliminate any duplications (e.g., make sure that all application areas use the same concept of "point"). For this level, a formal data specification language EXPRESS has been developed to "express" the underlying schema; and
- the physical layer, dealing with the mapping of the schema onto the syntactical format of a sequential file.

7.10 STEP

In December 1984, Subcommittee 4 of the Technical Committee 184 (Industrial Automation Systems) of the International Standardization Organization was founded upon request by the US and France. ISO/TC184/SC4 is to produce a single international standard. The US delegation plays the leading role in the working group ISO/TC184/SC4/WG1. This standard is intended to replace the previous ones. It shall cover at least the following areas:

Applications:
- AEC (architecture, engineering, construction);
- electrical applications;
- electronic applications;
- finite element applications;
- mechanical products; and
- drafting (annotations, dimensioning, tolerancing, text).

Product model data:
- 2D and 3D curves (analytical and parametric);
- surfaces (analytic and parametric);
- solids (CSG and B-rep);
- tolerance definition;
- form features;
- physical properties;
- product assembly structure;
- material specifications;
- process planning;
- administration and control; and
- presentation.

Product life-cycle data:
- analysis;
- manufacturing (planning, fabrication, assembly);
- quality assurance;
- testing; and
- product support.

Experts from many countries representing many decades of experience in CAD data transfer cooperate in this effort [WILS87]. Preliminary documents intended to evolve into chapters of the future standard have been produced and raised to the level of a first international standard draft proposal in December 1988 (ISO 10303 DP).

Both the CAD*I and the PDES activities have had a significant influence on the STEP development, and have themselves been influenced greatly by the interactions on the international level.

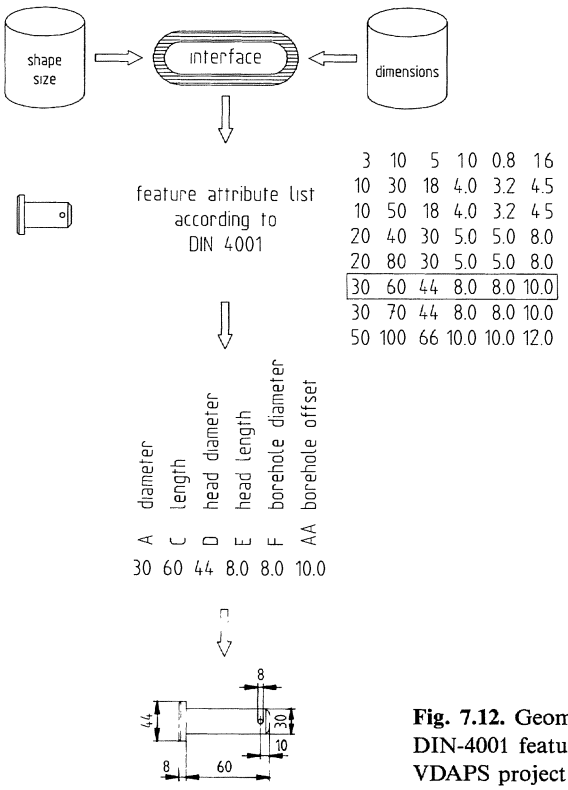


Fig. 7.12. Geometric model generation from DIN-4001 feature attributes according to the VDAPS project

7.11 Standardization of "Standard Parts"

So far we have dealt mainly with CAD models whose geometric representations are to be transferred entirely from one CAD system to another. A significant portion of design, however, is concerned with the use of standard parts (nuts and bolts and the like). Standard parts and their geometrical representations are supposed to be known to the receiving CAD system. Hence, only the identification of the standard part and its location in the model have to be transferred. In Germany, two projects are concerned with the standard part problems.

In a DIN project the contents of all relevant DIN standards are converted from paper representation into machine readable form [GUER86] (DIN-4001). The information about the standard parts are captured in sequential lists of "Sachmerkmale" (feature attributes). As a result of this project, CAD systems conforming with this new standard will guarantee the use of the proper numerical values whenever standard parts are to be chosen or used.

The second project (VDAPS = VDA-Programm-Schnittstelle) is funded by the automotive industry in Germany (VDA) [JONA87]. This project concerns itself with the consistency of the geometric models of standard parts which are generated by the

various CAD systems on the basic of DIN-4001. The two possible representations analyzed were parametric data structures (similar to CAD*I's macros) and executable programs (similar to CAD*I's routines). The routine solution was chosen. A (very large) set of FORTRAN subroutines is presently under development. These subroutines will build geometric models in CAD systems when they are invoked with a set of feature attributes (see Fig. 7.12). On the application level, the subroutines have standardized names and parameter lists. On the implementation level they will call upon a small set of low level modeling routines which are supposed to be available as a programming interface in all relevant CAD systems. This set of low level routines was selected after an analysis of 13 CAD systems which are mainly used in the automotive industry [SCHM86]. The VDAPS specification is to become DIN standard 66304.

The amount of software routines that will have to be developed for this project is impressive: about 15000 FORTRAN subroutines were estimated to be needed in order to implement the 2400 "standard parts" available in the DIN standard system [GUER86].

7.12 Bibliography

[BEY__85] I. Bey, J. Leuridan: ESPRIT Project 322, CAD Interfaces: Interfaces in CAD Systems. Presented at the ESPRIT Technical Week, ESPRIT '85, Status Report. Amsterdam, North-Holland (1985).

[BEY__86] I. Bey, J. Leuridan (eds.): Development of Specifications for Exchange of Product Definition and Analysis Data. ESPRIT Project 322, CAD*I: CAD Interfaces. Presented at the ESPRIT Technical Week, ESPRIT '86. Amsterdam, North-Holland (1986).

[BOOZ83] ANSI: Product Definition Data Interface. Task-I: Evaluation and Verification of ANSI Y14.26M. Booz-Allen and Hamilton Inc. (1983).

[CAMI82] CAM-I: Geometric Modelling Project Boundary File Design XBF-2 (1982).

[CLAU85] D. Claude: Exposé IGES. CAM-I Report M-84-GM-05 (1985).

[ENCA86] J. Encarnação, R. Schuster, E. Vöge: Product Data Interfaces in CAD/CAM Applications. Berlin, Springer-Verlag (1986).

[ESP__84] IGES: E.S.P. Experimental Solids Proposal (1984).

[GRAB86] H. Grabowski, R. Glatz: Testing and Validation of IGES Processors. In: J. Encarnação, R. Schuster, E. Vöge, Product Data Interfaces in CAD/CAM Applications. Berlin, Springer-Verlag (1986), pp. 221–235.

[GRGL86] H. Grabowski, R. Glatz: Schnittstellen zum Austausch produktdefinierender Daten. VDI-Z 128, 10 (1986), pp. 333–343.

[GUER86] G. Gürtler: Sachmerkmale und CAE. DIN-Projekt Normteiledatei. Datenverarbeitung in der Konstruktion '86. CAD und Informatik. VDI-Berichte 610.1 (1986), pp. 23–39.

[IGES81] Digital Representation for Communication of Product Definition. Data Chapter I–IV. ANSI Y14.26M (1981).

[IGES83] NBS: Initial Graphics Exchange Specification Version 2.0 (NSIR82-2631) (1983).

[IGES86] NBS: IGES Mail Ballot RFC's and CO's for Version 4.0. Document (1986).

[JONA87] W. Jonas, W. Schmädeke, R. Schulz: VDAPS: Programmschnittstelle für Normteile wird DIN-Norm. VDI-Z 129, 5 (1987), pp. 66–70.

[LIEW85] M. H. Liewald: Initial Graphics Exchange Specification: Successes and Evolution. Comput. & Graphics 9, 1 (1985), pp. 47–50.

[MAAN87] J. van Maanen, D. Thomas (eds.): Specification for the Exchange of Product Analysis Data, ESPRIT Project 322: CAD*I (CAD Interfaces), Kernforschungszentrum Karlsruhe (1987).

[SCHL86] E. G. Schlechtendahl (ed.): Specification of a CAD*I Neutral File for Solids Version 2.1. ESPRIT Project 322, Springer-Verlag (1986).

[SCHL87] E. G. Schlechtendahl (ed.): Specification of a CAD*I Neutral File for CAD Geometry, Version 3.2. ESPRIT Project 322, Springer-Verlag (1987).

[SCHM86] W. Schmädeke, R. Heydrich, W. Jonas: VDA-Programm-Schnittstelle für Norm- und Wiederholteile in CAD-Systemen. VDI-Z 128, 15/16 (1986), pp. 615–618.

[SET__85] AFNOR: Représentation externe des données de définition de produits, Specification du standard d'échange et de transfert (SET) Version 85-08. Norme Expérimentale Z 68–300 (1985).

[SET__87] Groupe Operational S.E.T. Aérospatiale: SET Solide. (1987)

[VDA__86] DIN: Format zum Austausch geometrischer Informationen. DIN 66301. Beuth-Verlag (1986).

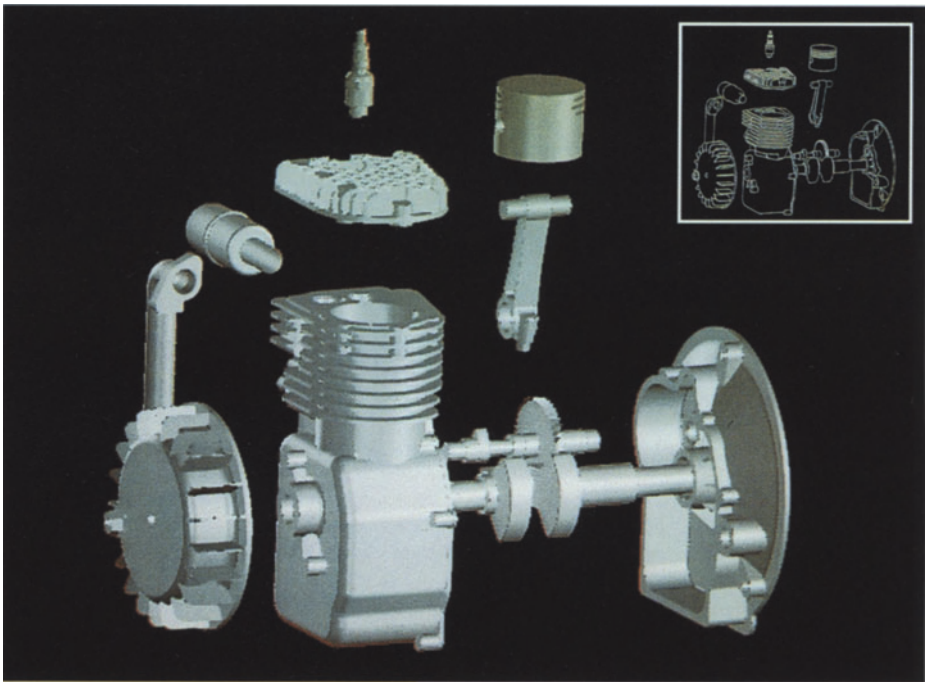
[VDIS86] VDA/VDMA: VDA IGES-Subset (VDAIS). Version 1.0. Entwurf Januar 1987. VDA Frankfurt (1987).

[WEIS85] U. Weissflog: Product Data Exchange; IGES and Alternatives. Datenverarbeitung in der Konstruktion '85. CAD und Informatik. VDI-Berichte 570.5 (1985), pp. 147–160.

[WILS85] P. R. Wilson et al.: Interfaces for Data Transfer Between Solid Modelling Systems. IEEE Comp. Graph. & Appl. (1985), pp. 41–51.

[WILS87] P. R. Wilson: A Short History of CAD Data Transfer Standards. IEEE Comp. Graph. & Appl. (1987), pp. 64–67 (1986).

8 CAD Application Examples



Assembly drawing from the solid model of a motor
(courtesy of Evans & Sutherland, Salt Lake City, USA)

A selection of CAD application examples will always contain some element of arbitrariness. The examples given in this chapter are taken from industrial practice, and represent some of the most common CAD application types. More application examples may be found in the proceedings of the relevant conferences and journals (such as [CINUD4], [LAEN88], [MICA89], [WAR__83], [CAD__], [CAE__]). An even wider spectrum of applications is published not in the computer-oriented literature, but rather in the journals and conference proceedings of the various branches of industry.

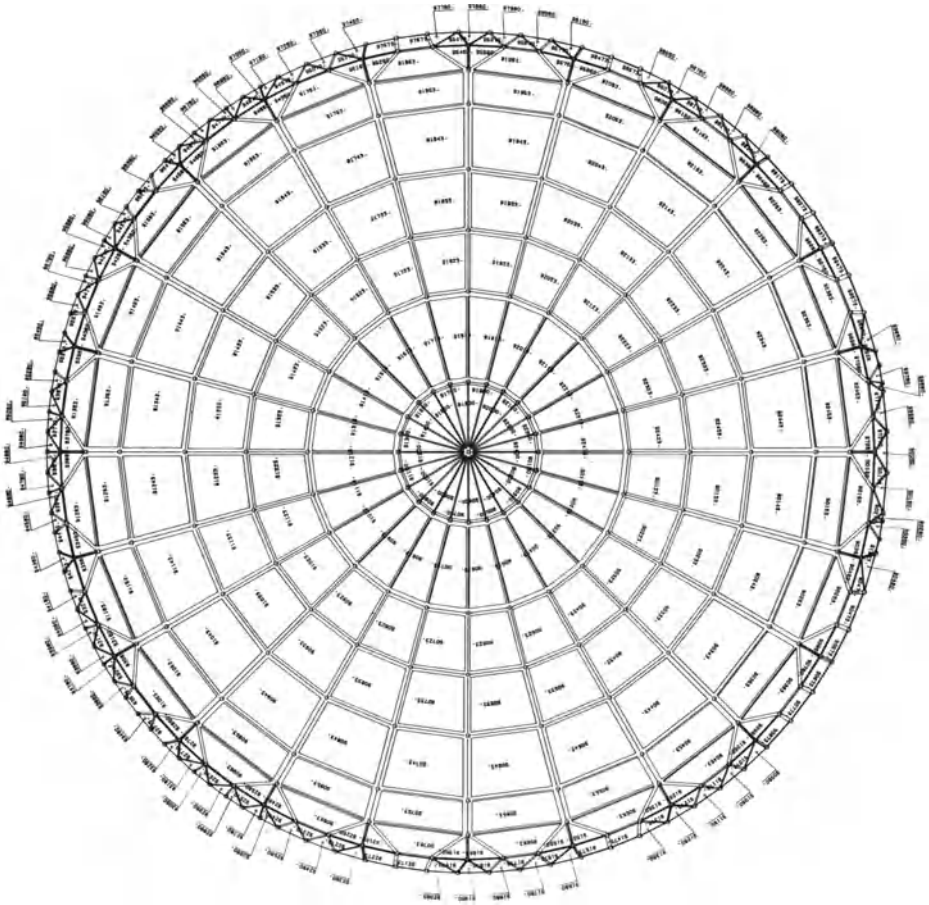


Fig. 8.1. Finite element model of the rear fuselage with pressure bulkhead of an Airbus A310, front view (courtesy of Messerschmitt-Bölkow-Blohm, Hamburg)

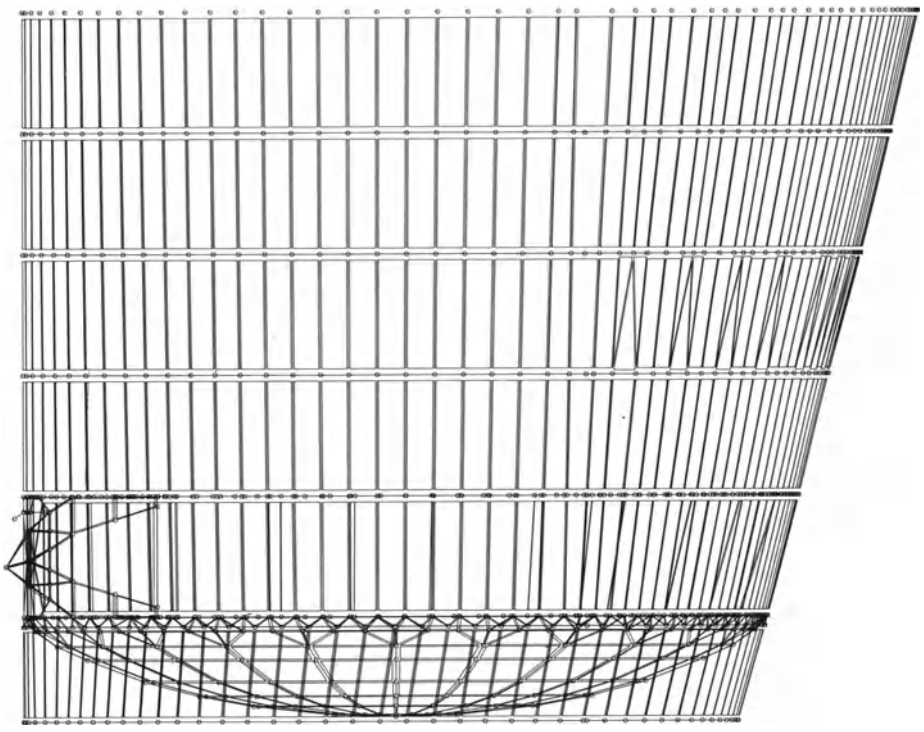


Fig. 8.2. Finite element model of the rear fuselage with pressure bulkhead of an Airbus 310, side view (courtesy of Messerschmitt-Bölkow-Blohm, Hamburg)

8.1 Numerical Analysis and Presentation

Finite element analysis, as the best-known application, is represented by Figs. 8.1 through 8.5. Figures 8.1 and 8.2 (courtesy of Messerschmitt-Bölkow-Blohm, Hamburg) show the finite element discretization of part of the rear fuselage with pressure bulkhead for an Airbus A310, in front view and side view respectively. The gross dimensions are 3550 mm length and 3400 mm in diameter. The model contains 4709 finite elements and a total of 6577 degrees of freedom. Representations of this type are useful for checking the correctness of the topology and geometry of the model. In this case, each finite element is drawn as a polygon, slightly smaller than its actual size; this makes checking for missing elements easier than it would be with elements drawn as full size polygons. The same representation technique is chosen for Fig. 8.3 (courtesy of IKO Software Service, Stuttgart), which shows the finite element model of a destroyer with 22000 degrees of freedom. Checking a model of this complexity requires projections from different viewing points, and is best done in interactive mode at a display terminal.

Figures 8.4 and 8.5 (courtesy of IKO Software Service, Stuttgart) are presentations of results from finite element analyses. Figure 8.4 shows the radial stresses in the

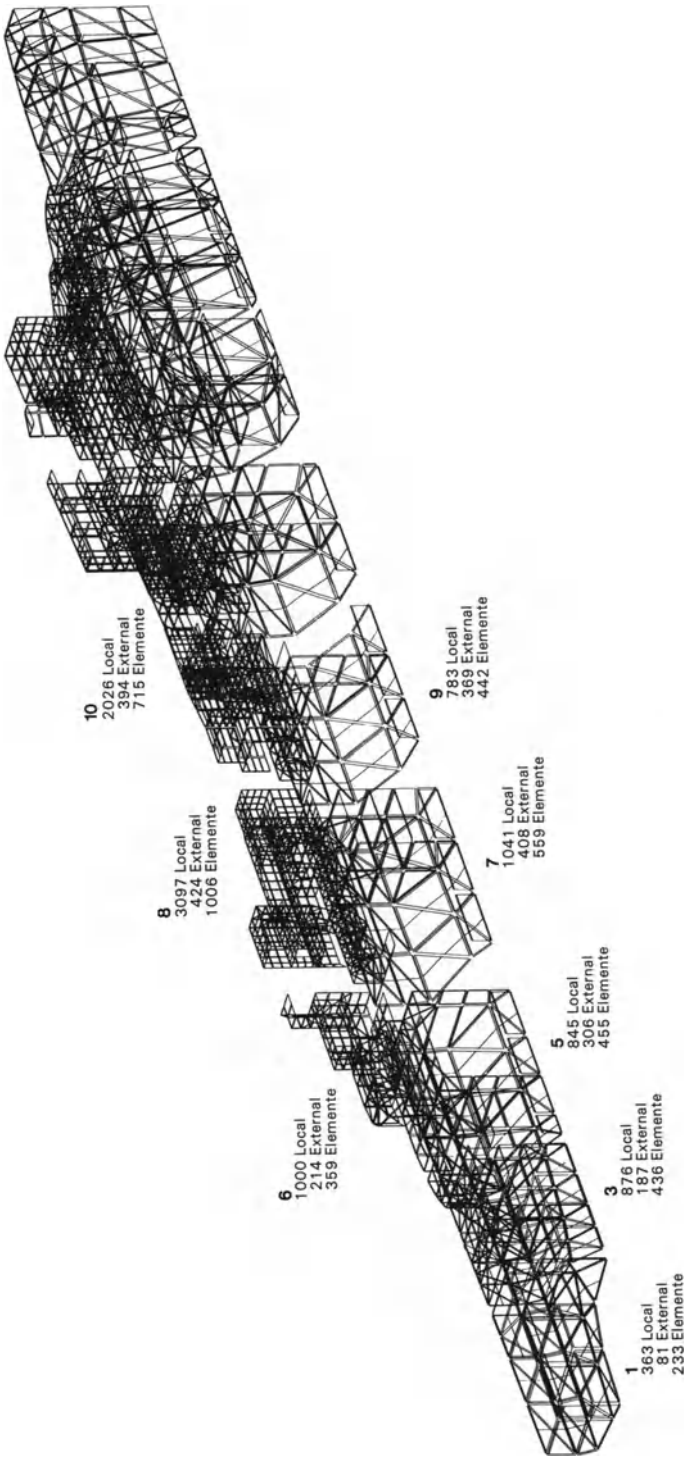


Fig. 8.3. Finite element model of a ship (courtesy of IKO Software Service, Stuttgart)

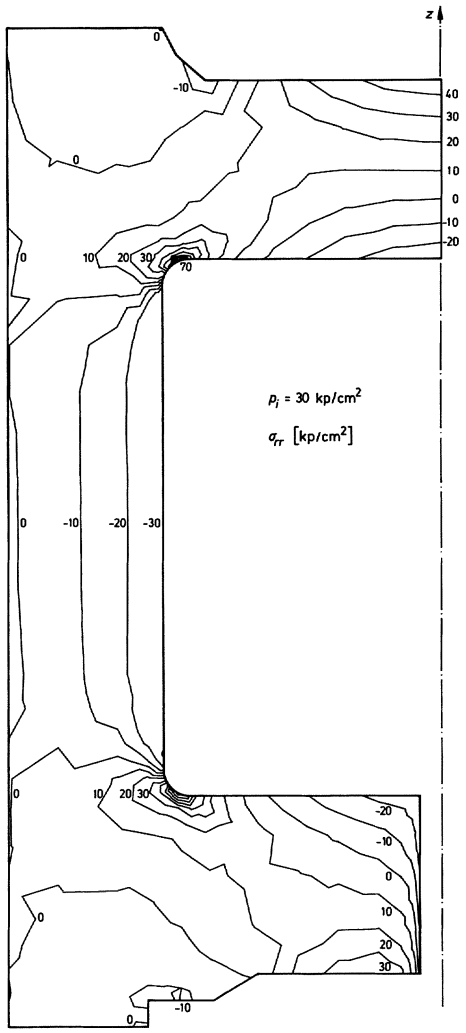


Fig. 8.4. Presentation of finite element analysis results (stresses) for a pre-stressed concrete pressure vessel of a nuclear reactor (courtesy of IKO Software Service, Stuttgart)

flange of a nuclear reactor pressure vessel. The isolines are obviously linear interpolations between computed nodal points. They give a clear indication of the degree of detailing used for the model. The isolines of Fig. 8.5 show stresses in a tube plate of a heat exchanger. From the curves one cannot decide whether the smoothness is a consequence of fine modeling or the result of an a posteriori smoothing (see Sect. 6.3.2.1). In either case, stress concentrations are made evident by the isolines alone; but for evaluation the annotations (levels of stress) are definitely needed.

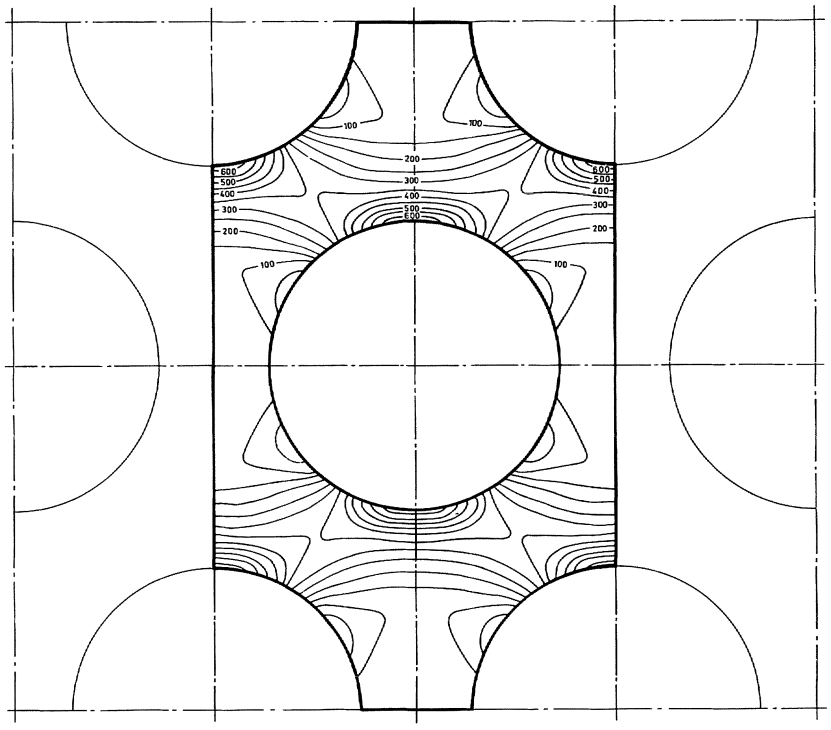


Fig. 8.5. Presentation of finite element analysis results (stresses) for a heat exchanger tube plate (courtesy of IKO Software Service, Stuttgart)

8.2 CAD Application in the Automotive Industry

Car body design starts generally from the data that have been obtained from measuring many thousands of points from the physical model produced by car body stylists. Figure 8.6 displays the essential points of a measured sculptured surface. Based upon these raw data, surface modeling (especially smoothing) techniques are used to develop the surface representations in the CAD model (see Sect. 6.1.3.3). These surface representations are then available for various further applications and display purposes (see Figs. 8.7 and 8.8). Finite element meshes (see Fig. 8.9) may be generated for structural analyses (static, vibration, crash). When production starts, quality assurance data are taken from measuring machines and compared with the design data as shown in Fig. 8.10. Kinematic studies are required for the design and analysis of the car wheels. Figure 8.11 (a rear wheel) and Fig. 8.12 (a front wheel) show different techniques for visualizing the kinematic behavior of the wheels. Both figures utilize the pseudo-coloring technique described in Sect. 6.3.3.2 (see also Fig. 6.41) for visualization.

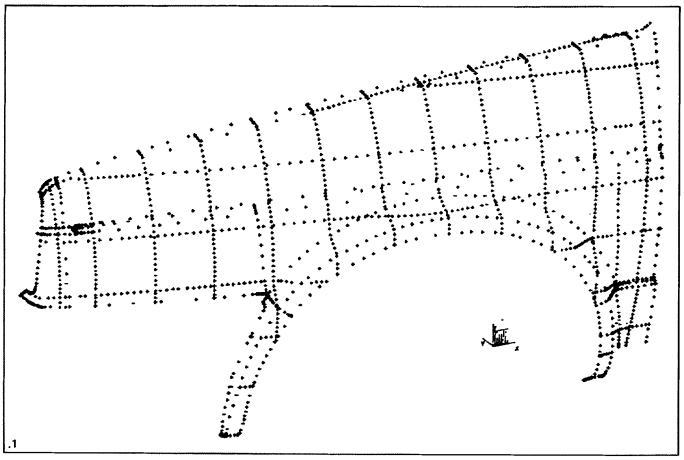


Fig. 8.6. Raster of measured points from a sculptured surface (courtesy of Volkswagenwerk, Wolfsburg)

Fig. 8.7. Shaded picture representation of the front portion of a car (courtesy of Volkswagenwerk, Wolfsburg)

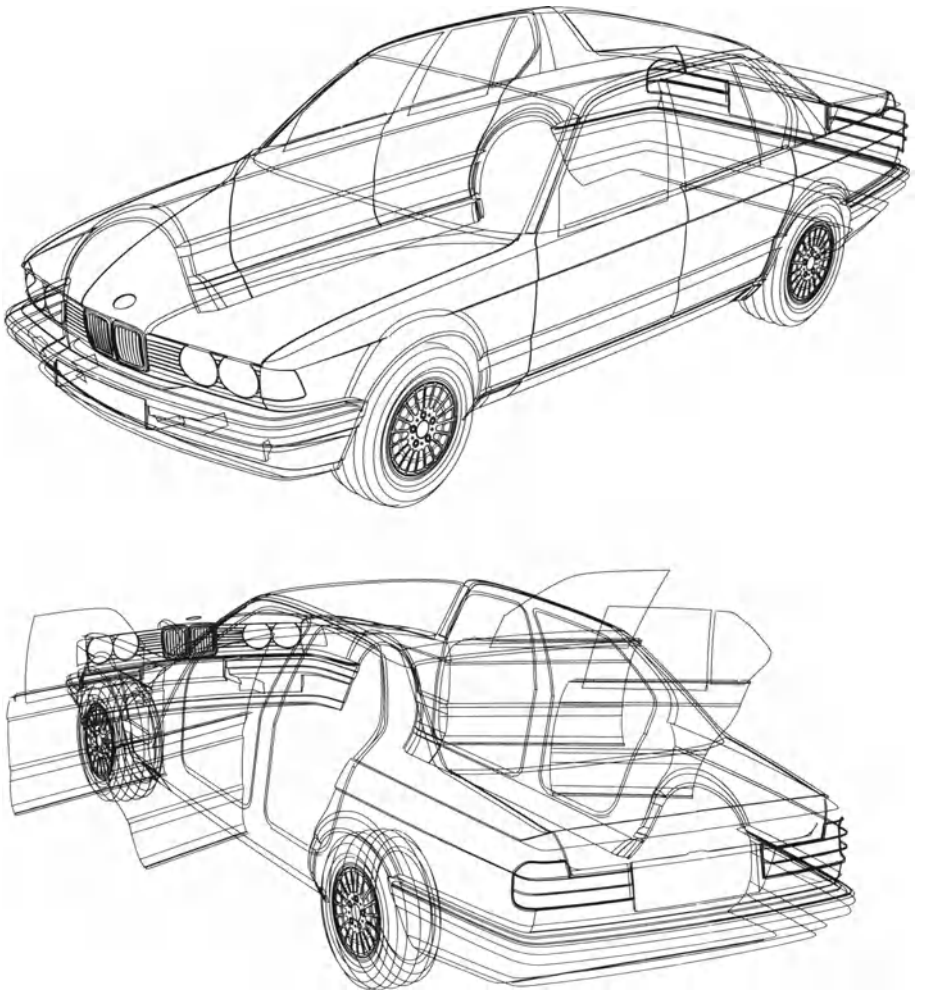


Fig. 8.8. Contour lines representing a car's surface in various projections (courtesy of Bayerische Motorenwerke, München)

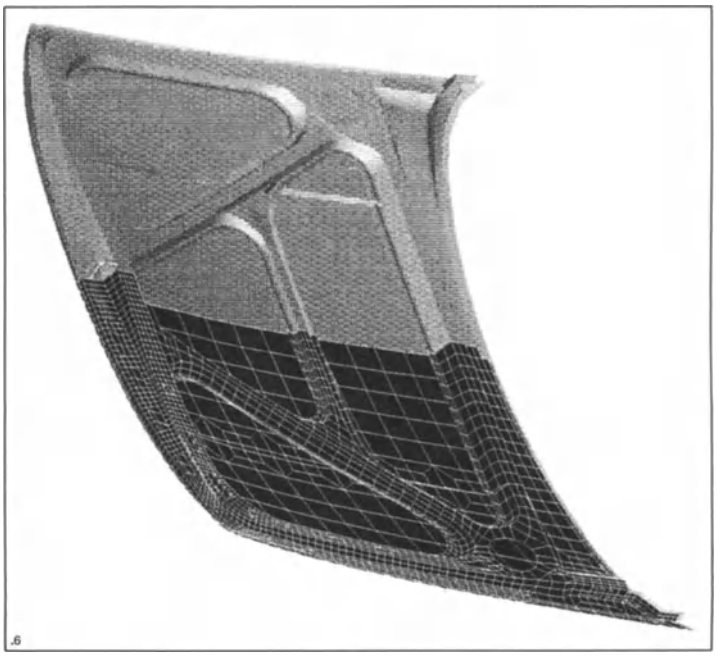


Fig. 8.9. A finite element mesh for structural analysis generated on the basis of a sculptured surface model (courtesy of Volkswagenwerk, Wolfsburg)

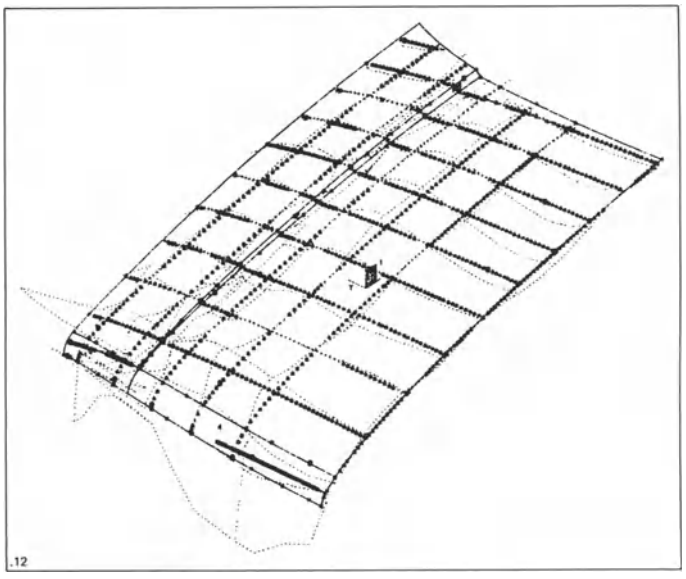


Fig. 8.10. Quality assurance data (measured points) are plotted on top of the design data. The deviation from the design locations of these points is significantly amplified (courtesy of Volkswagenwerk, Wolfsburg)

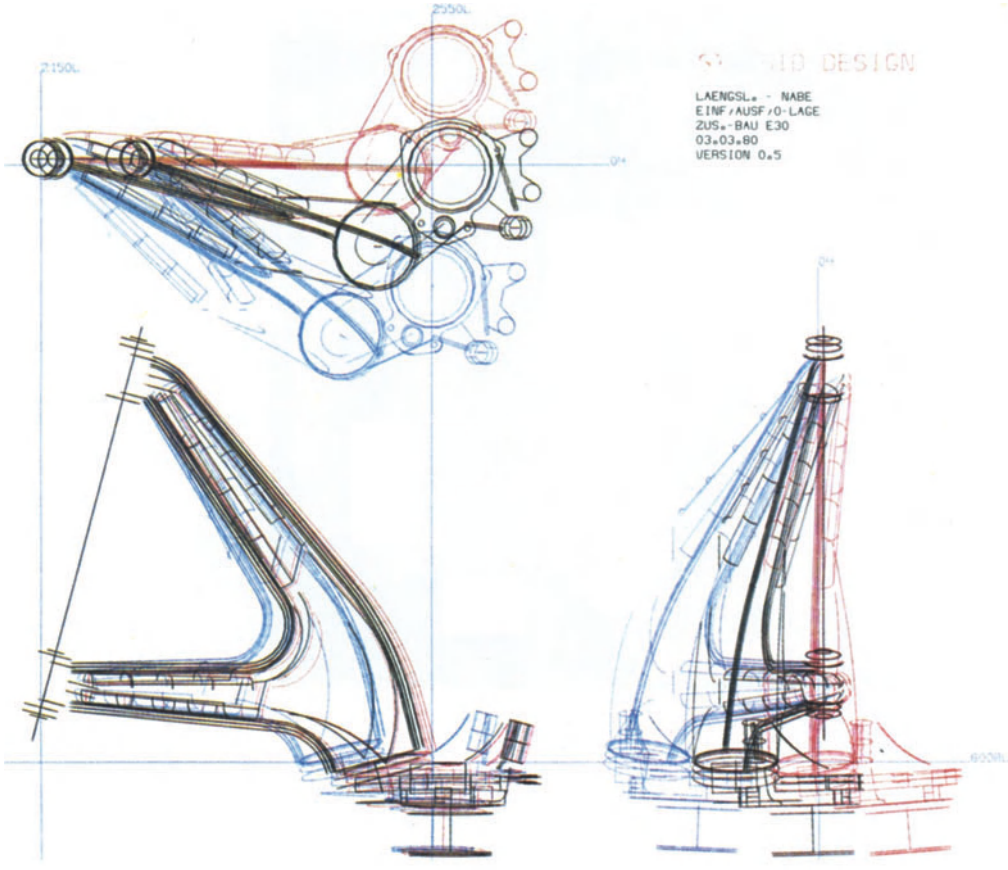


Fig. 8.11. Overlay representation of three positions of a car's rear wheel support (courtesy of Bayerische Motorenwerke, München)

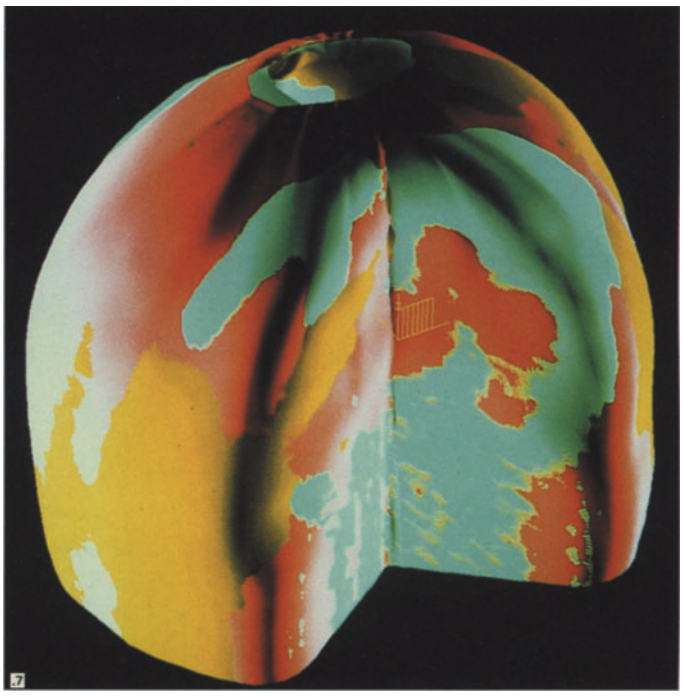


Fig. 8.12. Hull surface for all possible positions of the front car wheels (three different sizes). This hull surface determines the free space required for the wheel (courtesy of Volkswagenwerk, Wolfsburg)

8.3 Functional and Geometrical Layout

Figures 8.13 to 8.20 illustrate steps in computer-aided VLSI design. Two-dimensional functional layout is demonstrated by Figs. 8.13 and 8.14. Figure 8.13 shows a screen during higher level logical circuit design while Fig. 8.14 shows the same design on a more detailed level. Figure 8.15 represents the timing behavior of the circuitry during simulation. Figures 8.16, 8.17, and 8.18 are examples of geometrical layout. Figure 8.16 shows this layout on a block level while Figs. 8.17 and 8.18 already represent the geometrical details of transistors, resistors, and connections. Figure 8.19 shows details from a plot of the mask layout of a VLSI chip and Fig. 8.20 shows a pad area of the chip as seen through a microscope.

More examples of geometrical layout are given in Figs. 8.21 and 8.22. Figure 8.21 represents part of a building layout. Figure 8.22 represents results from an architectural CAD application.

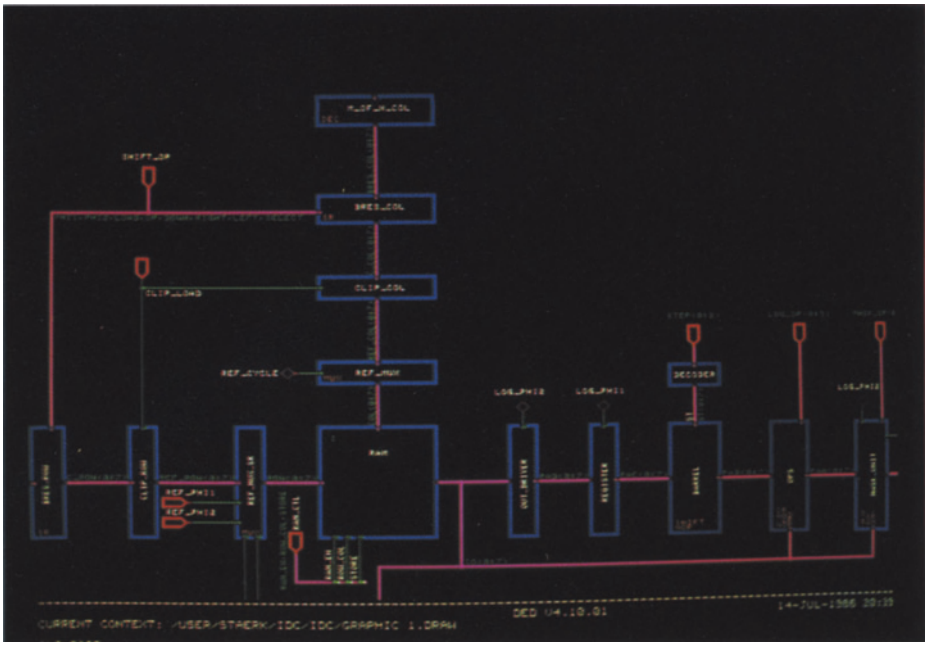


Fig. 8.13. Higher level logical circuit design

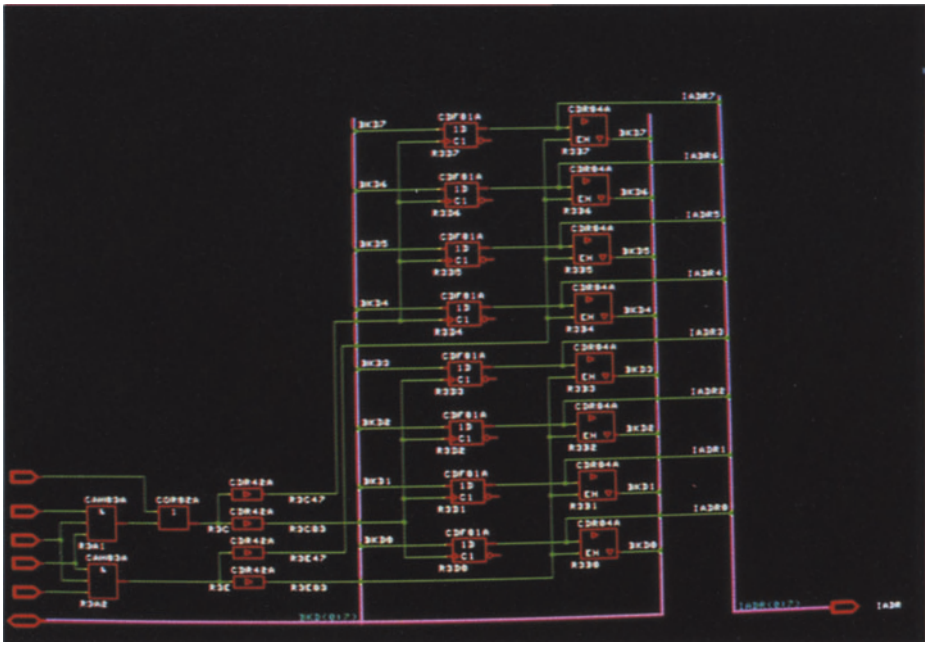


Fig. 8.14. Lower level logical circuit design

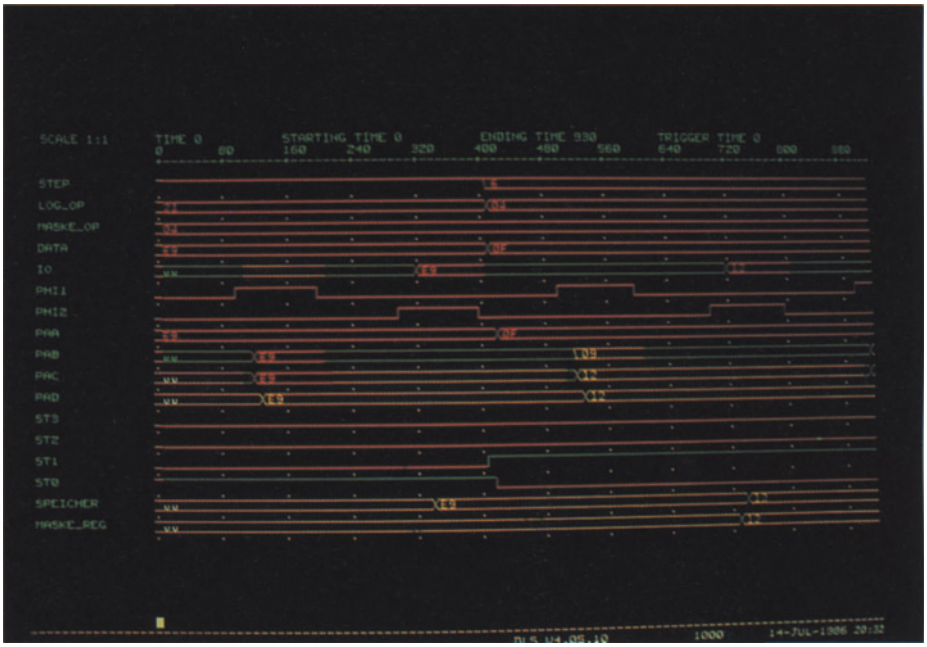


Fig. 8.15. Logical circuit simulation

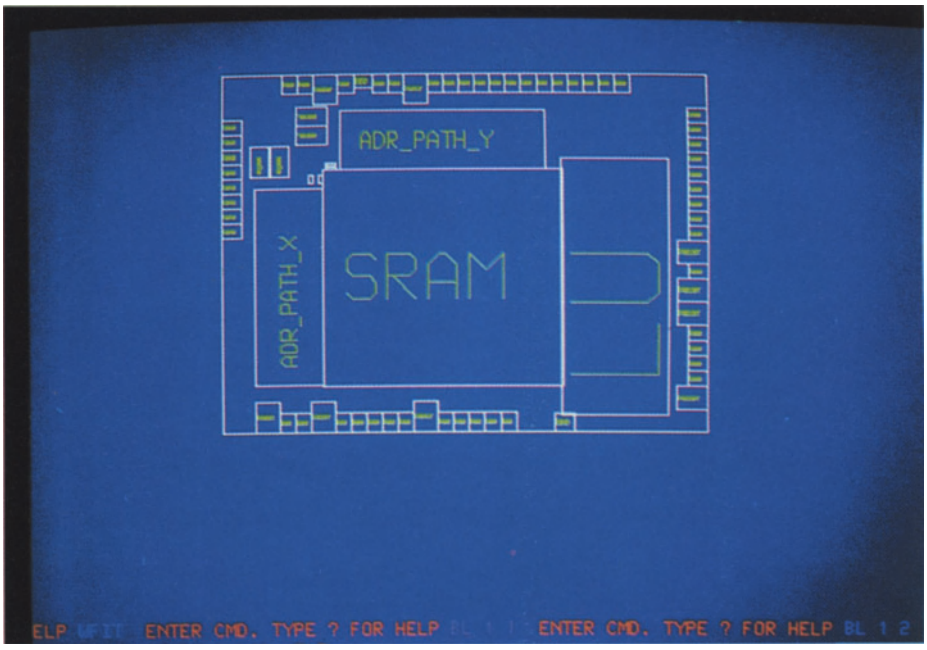


Fig. 8.16. Higher level geometrical layout

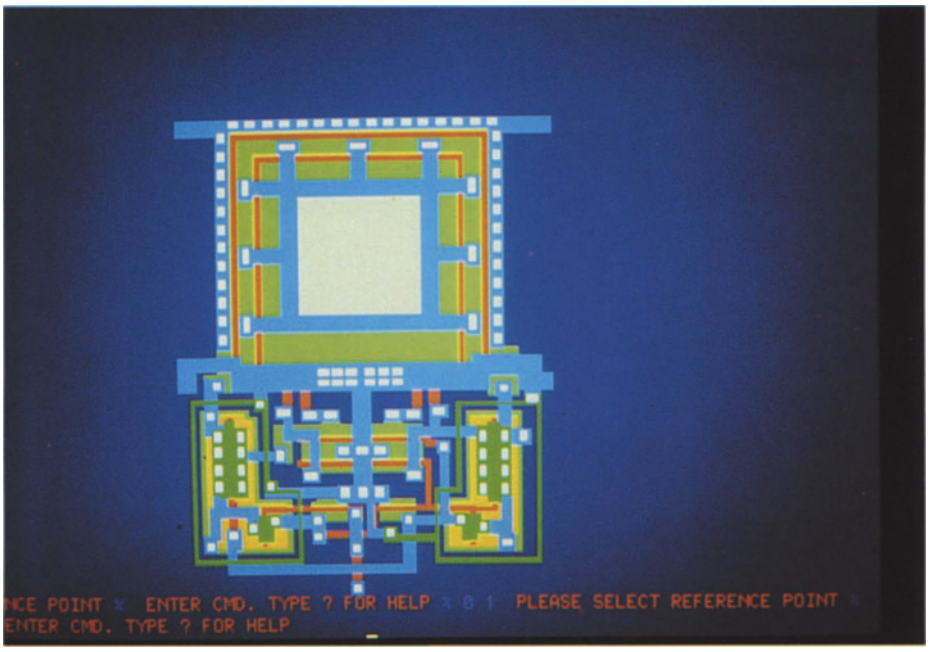


Fig. 8.17. Lower level geometrical layout: PAD area

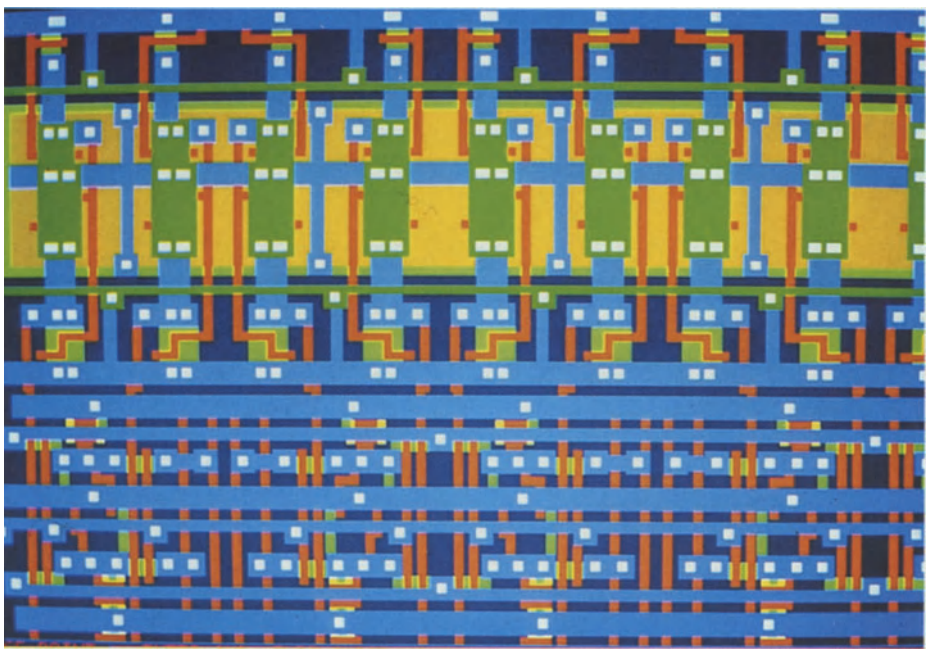


Fig. 8.18. Lower level geometrical layout: SRAM area

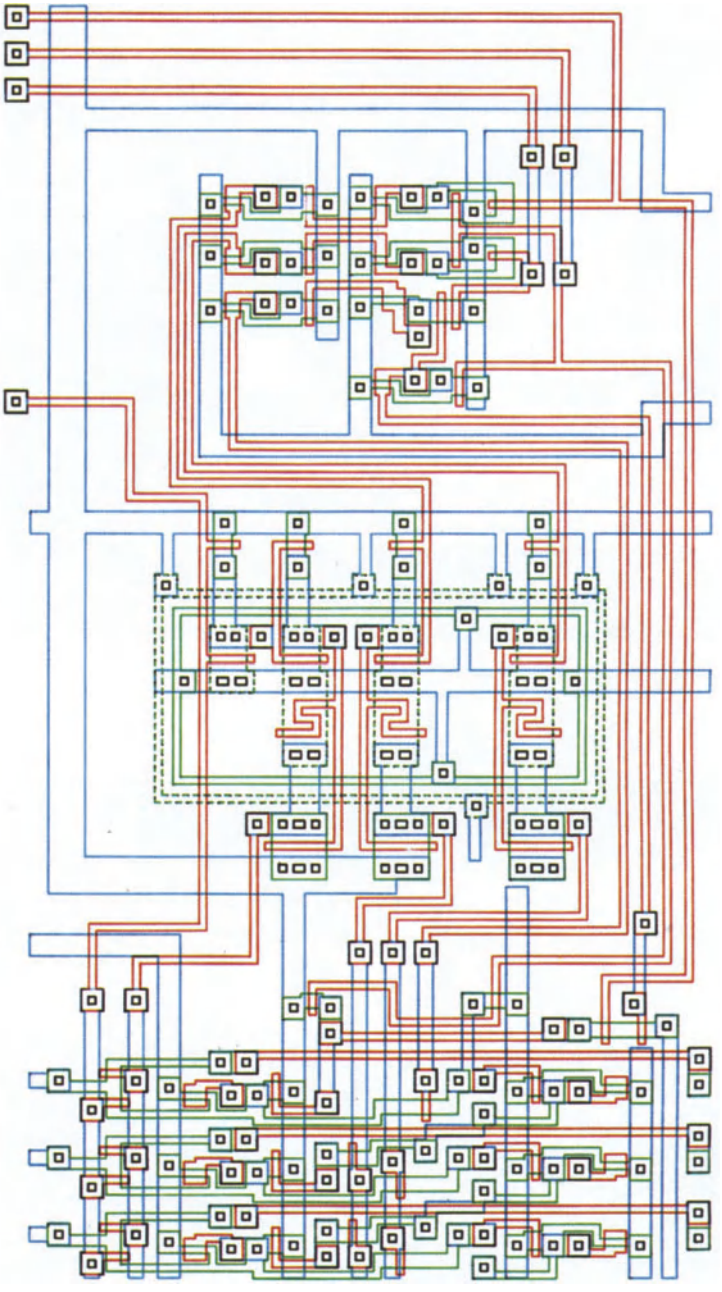


Fig. 8.19. Detail of mask layout

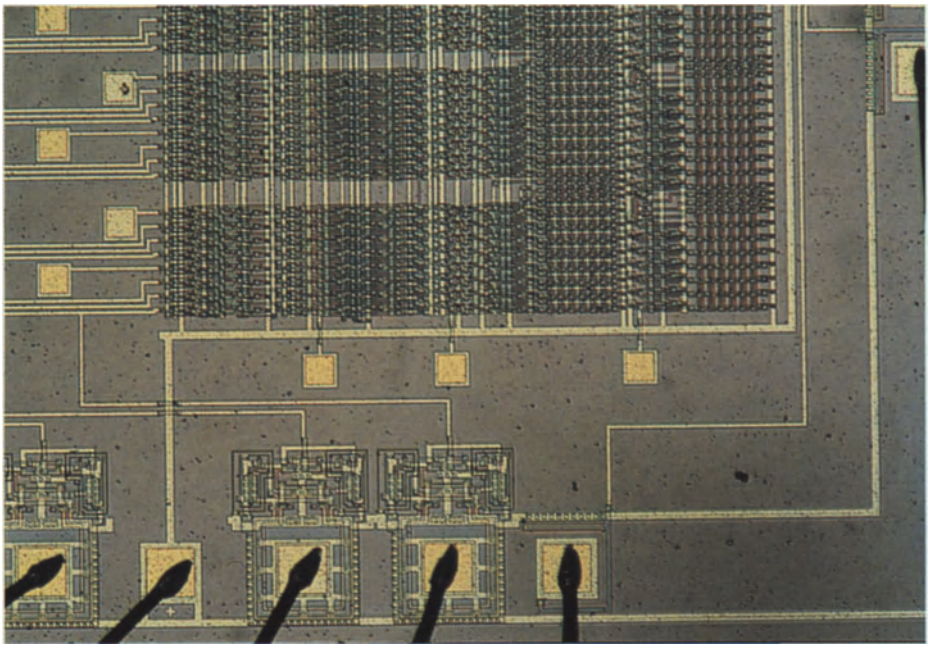


Fig. 8.20. Appearance of a VLSI chip's PAD area after bonding

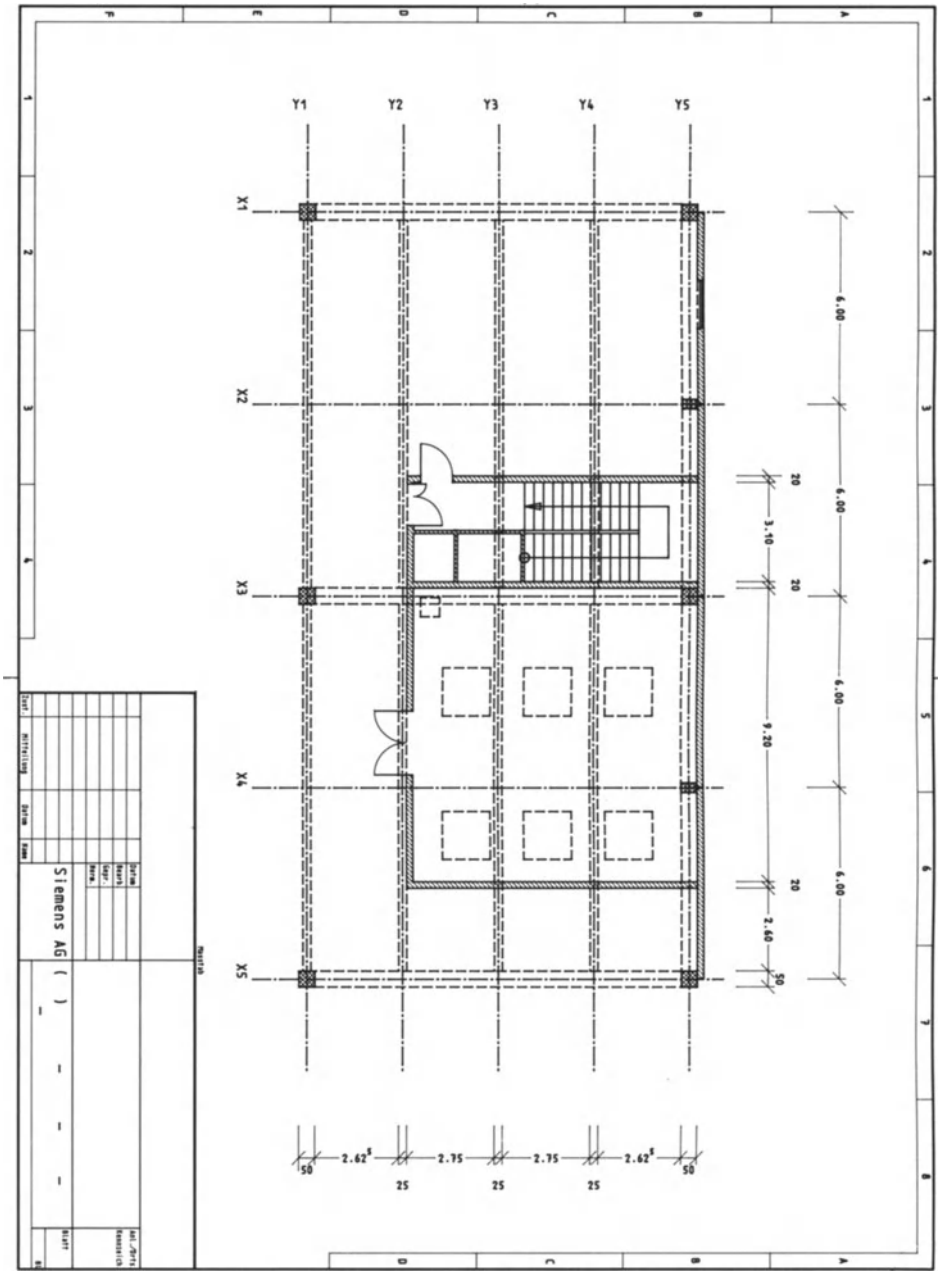


Fig. 8.21. Building layout (courtesy of Siemens, Erlangen)

Fig. 8.22. Architectural CAD applications (courtesy of GMW Computers, Hertshoreshire, UK)

8.4 CAD Application for Fusion Reactor Development

The development of a fusion reactor calls for the combination of all high-tech potential on a world-wide basis. The physical conditions which have to be established in the core of the plasma to sustain a quasi-steady-state fusion process present some of the most demanding challenges for today's technology. World-wide cooperation in the international projects NET (Next European Torus) and ITER (International Tokamak Experimental Reactor) call for the exchange of design data on a large scale. The following examples from the NET development work indicate some of the interactions which take place.

Figure 8.23 is a drawing which was originally produced in one system (Medusa), converted into an IGES format, transmitted via the public package switching network

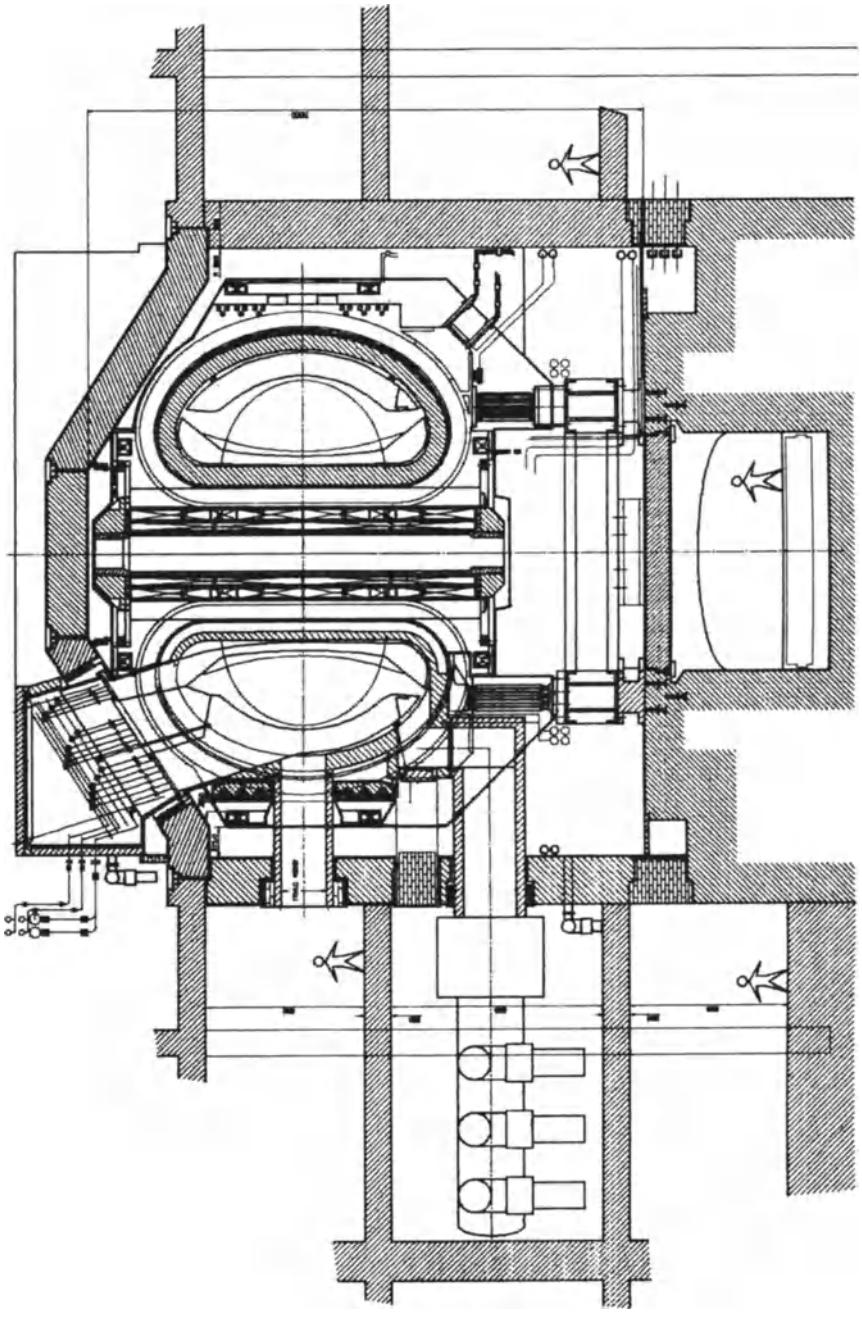


Fig. 8.23. Cross-section through the NET fusion reactor (courtesy of NET-Team, Garching, Germany)

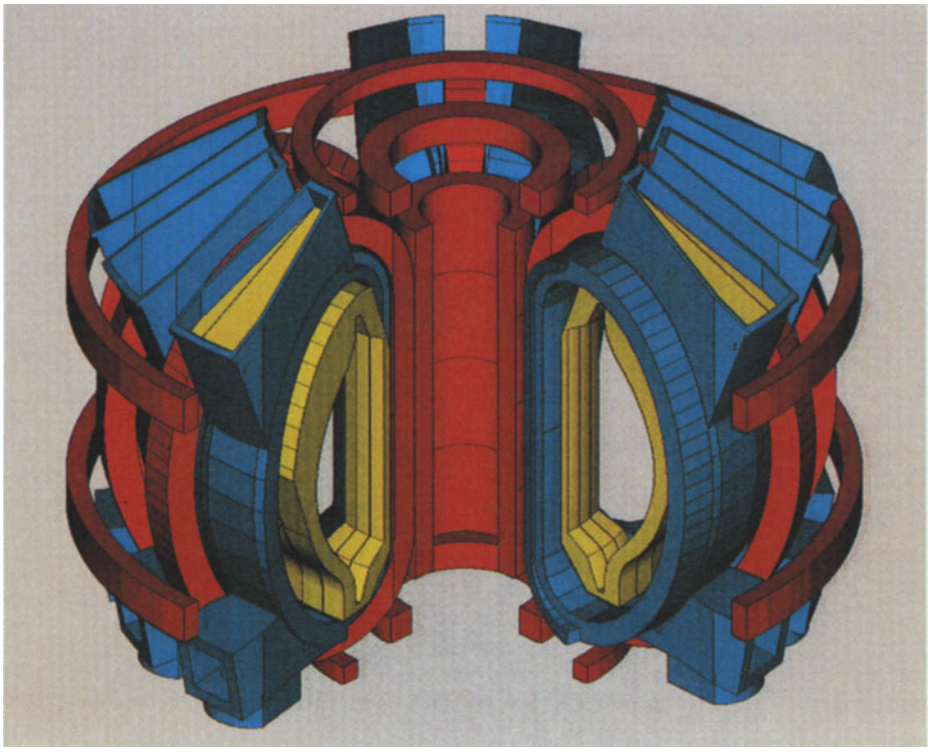
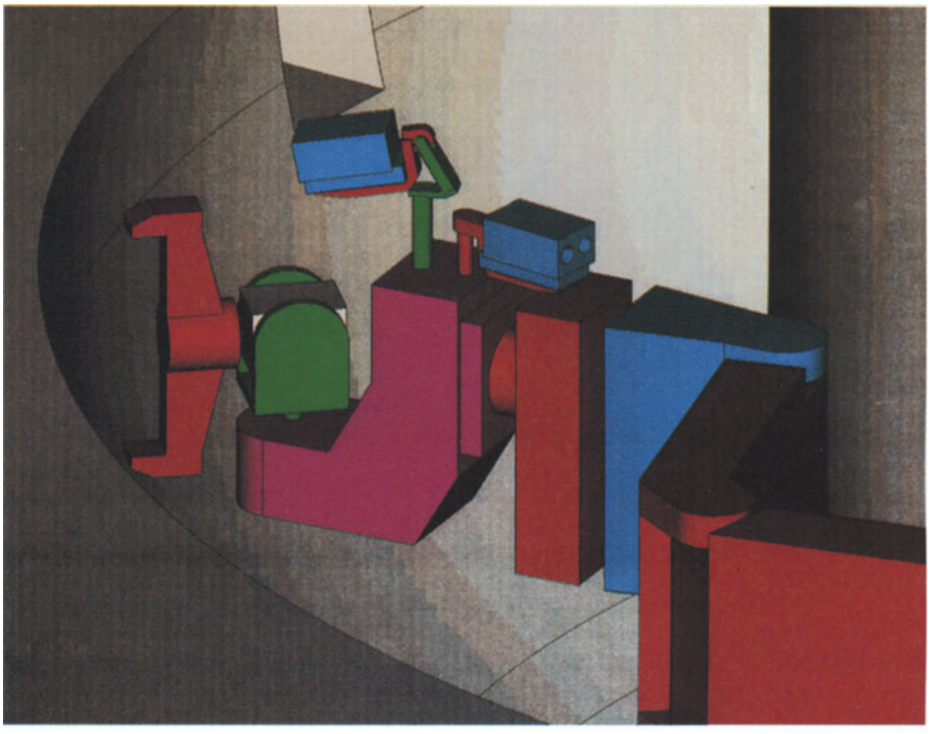
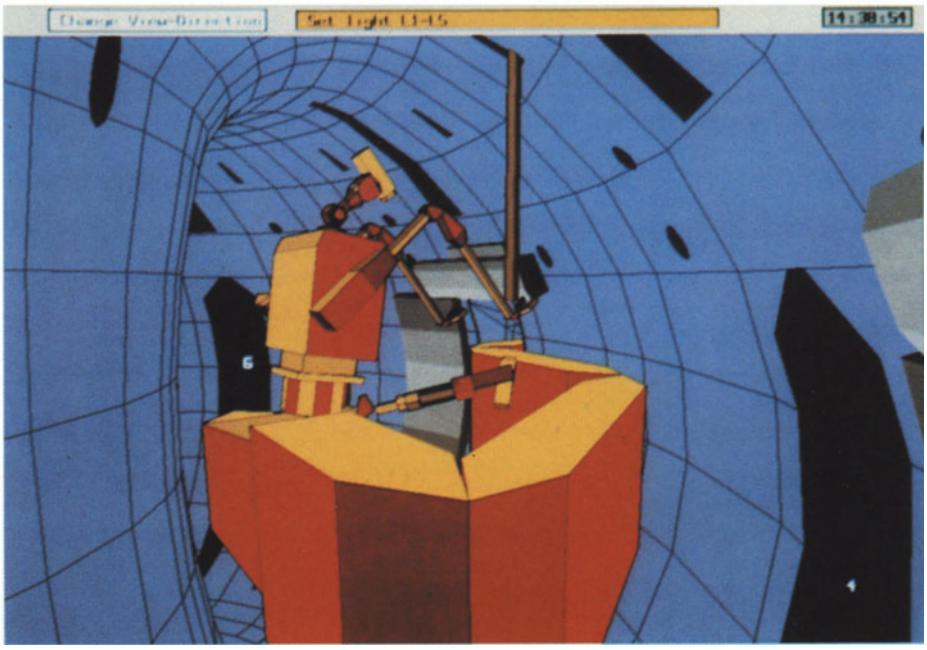


Fig. 8.24. Solid model of NET (courtesy of Kernforschungszentrum, Karlsruhe, Germany)

as a sequential file, converted into the internal representation of another system (Bravo3), and finally plotted. However, in the receiving system the CAD information was not only used for plotting but also for developing a solid model of the design. This solid model is used for investigating whether the various components of the machine can be assembled without collisions (Fig. 8.24). Furthermore, a manipulator had to be designed which will operate inside this machine (with a reach of about 28 m and a load capacity of one ton) fully under remote control. Solid modeling techniques are used to confirm that the design allows the reaching of all necessary working positions without collisions. Figure 8.25 shows the front-end of such a manipulator. Finally, the solid model is transferred via a neutral file format into specialized computer graphics workstations where it is used for on-line visualization of the remote handling actions (especially when no other means are available to the remote-handling operator to monitor his actions) and for real-time collision warning. Figure 8.26 shows the display of such a model for the JET (Joint European Torus) fusion reactor.





8.5 Bibliography

[CAD__] Computer Aided Design, published 6 times a year.

[CAE__] Computer Aided Engineering, published monthly, Penton Publishing.

[CINUD4] Computers in Industry, An International Journal, published four times a year, Elsevier Science Publishers B.V.

[LAEN88] G. L. Lastra, J. L. Encarnação, A. A. G. Requicha (eds.): Applications of Computers to Engineering Design, Manufacturing and Management. Proceedings of the IFIP TC 5 Conference on CAD/CAM Technology Transfer, Mexico City, Mexico. Amsterdam, North-Holland (1988).

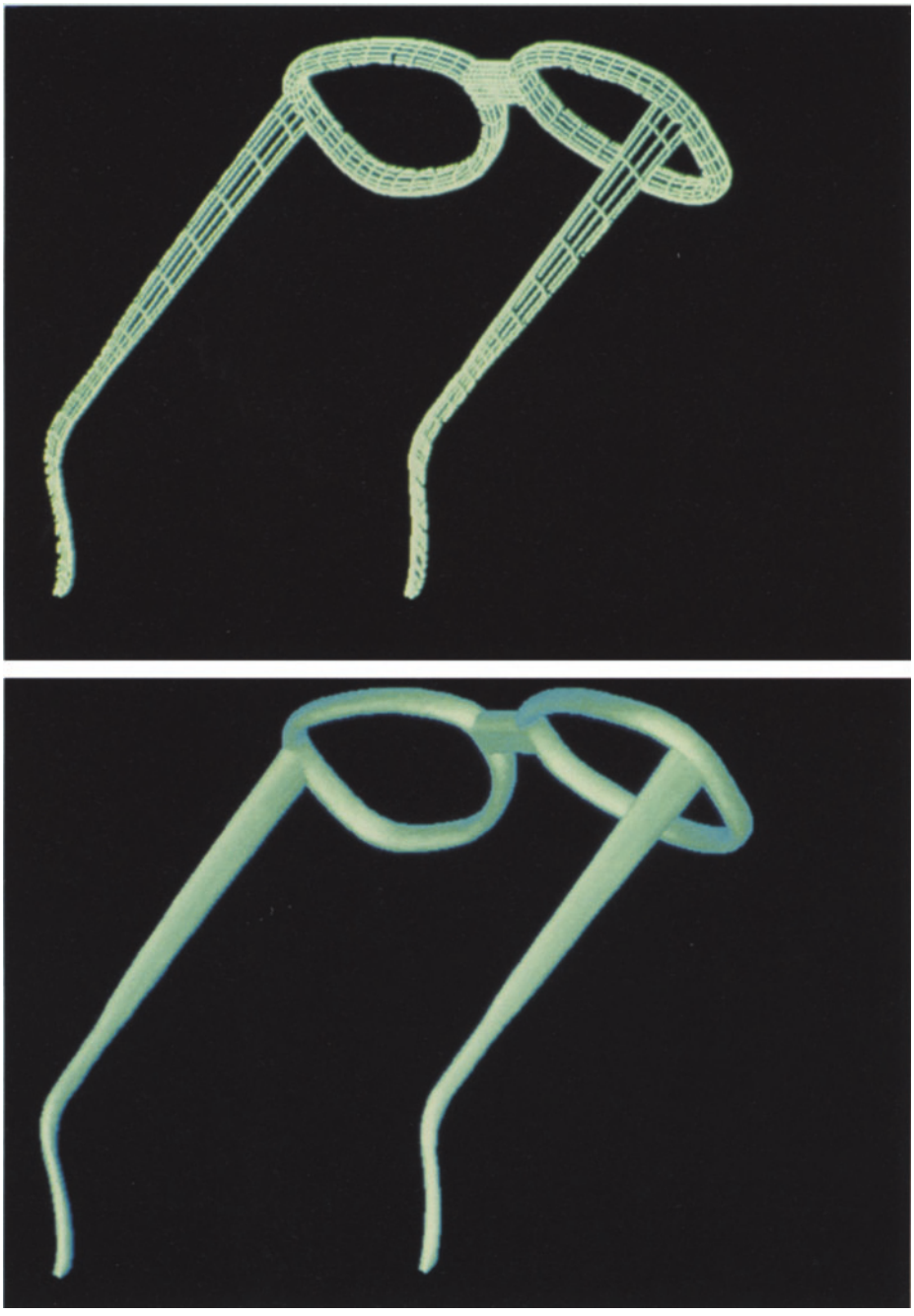
[MICA89] Proceedings of the 8th International Conference on CADCAM, Computer Graphics and Computer Aided Technologies, Porte de Versailles. Paris, Editions Hermès (1989).

[WAR__83] E. A. Warman (ed.): Computer Applications in Production and Engineering. Proceedings of the First International IFIP Conference on Computer Applications in Production and Engineering CAPE'83, Amsterdam, North-Holland (1983).

Fig. 8.25. The front end of a remote-handling manipulator in the NET fusion reactor (courtesy of Kernforschungszentrum, Karlsruhe, Germany)

Fig. 8.26. Real-time display of a solid model for the JET in-vessel manipulator (courtesy of Kernforschungszentrum, Karlsruhe, Germany)

9 Subject Index



Two different presentations of the sculptured model of a spectacle frame
(courtesy of FhG/AGD, Darmstadt, Germany)

10 Author Index

Page numbers in *italics* refer to the bibliography.

Towards computer generated realistic presentation:
liquor poured from a glas into a bowl
(courtesy of Truevision, Indianapolis, USA)