

Some pages of this thesis may have been removed for copyright restrictions.

If you have discovered material in Aston Research Explorer which is unlawful e.g. breaches copyright, (either yours or that of a third party) or any other law, including but not limited to those relating to patent, trademark, confidentiality, data protection, obscenity, defamation, libel, then please read our [Takedown policy](#) and contact the service immediately (openaccess@aston.ac.uk)

**AUTOMATIC REFINEMENT OF CONSTRUCTIVE SOLID
GEOMETRY MODELS**

Esfandyar AFSHARI-ALIABAD

Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

March 1991

© This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's prior, written consent.

The University of Aston in Birmingham

AUTOMATIC REFINEMENT OF CONSTRUCTIVE SOLID GEOMETRY MODELS

Esfandiyar AFSHARI-ALIABAD

Doctor of Philosophy

March 1991

Summary

Geometric information relating to most engineering products is available in the form of orthographic drawings or 2D data files. For many recent computer based applications, such as Computer Integrated Manufacturing (CIM), these data are required in the form of a sophisticated model based on Constructive Solid Geometry (CSG) concepts. A recent novel technique in this area transfers 2D engineering drawings directly into a 3D solid model called "the first approximation". In many cases, however, this does not represent the real object. In this thesis, a new method is proposed and developed to enhance this model.

This method uses the notion of expanding an object in terms of other solid objects, which are either primitives or first approximation models. To achieve this goal, in addition to the prepared subroutine to calculate the first approximation model of input data, two other wireframe models are found for extraction of sub-objects. One is the wireframe representation of input, and the other is the wireframe of the first approximation model. A new fast method is developed for the latter special case wireframe, which is named the "first approximation wireframe model". This method avoids the use of a solid modeller.

Detailed descriptions of algorithms and implementation procedures are given. In these techniques utilisation of dashed line information is also considered in improving the model. Different practical examples are given to illustrate the functioning of the program. Finally, a recursive method is employed to automatically modify the output model towards the real object. Some suggestions for further work are made to increase the domain of objects covered, and provide a commercially usable package. It is concluded that the current method promises the production of accurate models for a large class of objects.

Key Words : Solid modelling, 2D to 3D conversion, CSG, Engineering drawing, Computer graphics

ACKNOWLEDGEMENTS

I would like to express my thanks to Dr. P. Cooley, for his supervision and guidance throughout this project.

I am most grateful to Materials and Energy Research Centre, Teheran, and the Iranian Ministry of Culture and Higher Education for their financial support.

Also I would like to thank Dr. Robert C. Thomson for his editing, and Dr. A. Kargas for his technical assistance.

CONTENTS

Summary.....	2
Acknowledgements.....	3
List of figures	8
List of tables.....	14
1. INTRODUCTION	16
1.1 Introduction.....	16
1.2 Aim of the project	22
1.3 Project's Environment.....	24
1.4 Thesis Plan.....	26
2. GEOMETRIC MODELLING THEORY	29
2.1 Introduction.....	29
2.2 Wireframe Technique.....	32
2.2.1 Data Representation	36
2.2.2 Definitions.....	38
2.3 Surface Modelling.....	38
2.3.1 Parametric Surface Definition	39
2.3.2 Types of Modelling.....	41
2.3.2.1 Transfinite Interpolation.....	41
2.3.2.2 Discrete Approximation	41
2.4 Solid Modelling	42
2.4.1 Constructive Solid Geometry (CSG).....	42
2.4.2 Boundary Representation.....	45
2.4.2.1 Euler Operation.....	48
2.4.2.2 Winged-edge data structure	50
2.4.3 Sweep Representation.....	51
2.4.4 Cell Decomposition.....	54
2.4.4.1 Spatial occupancy enumeration.....	55
3. THEORY OF ENGINEERING DRAWING.....	57
3.1 Introduction.....	57
3.2 Orthographic Projections.....	57
3.2.1 First Angle Orthographic Projection.....	59
3.2.2 Third Angle Orthographic projection.....	61
3.2.3 Orthographic projections of primitives and prismatic objects....	62

3.2.4 Orthographic views of arcs and lines.....	65
3.2.5 Visibility	68
3.3 Engineering Drawing Conventions	68
3.3.1 Line styles	68
3.3.2 Fictitious and unspecified lines	69
3.3.3 Symbolic representation	69
3.4 Views Needed.....	70
3.5 Pictorial Drawing.....	71
3.5.1 Isometric views	72
3.5.2 Oblique Drawing.....	72
3.5.3 Perspective Drawing	73
3.6 Dimensioning and Tolerances	74
4. 2D TO 3D CONVERSION.....	77
4.1 Introduction.....	77
4.2 Review of Published Works.....	78
4.2.1 A System to Generate a Solid Figure from Three Views	80
4.2.1.1 Transformation to three-dimensional points	81
4.2.1.2 Transformation to the three-dimensional lines.....	81
4.2.1.3 Ghost figure elimination.....	82
4.2.1.4 Construction of the faces.....	84
4.2.2 On Automatic Recognition of 3D Structure from 2D Representation.....	84
4.2.2.1 Main components	85
4.2.2.2 Details of the method	85
4.2.3 Recognition of 3D Objects from Orthographic Projections.....	90
4.2.3.1 Reconstruction algorithm.....	90
4.2.4 Inputting Constructive Solid Geometry Representation Directly from 2D Orthographic Engineering Drawings....	101
4.2.5 Automatic Construction of curvilinear Solids from Wireframe Views.....	105
4.2.5.1 Checking and preparing input data.....	105
4.2.5.2 Construction of wireframe	106
4.2.5.3 Surfaces and faces	107
4.2.5.4 Solid generation	107
4.2.6 Discussion.....	110
4.3 Interpretation of Engineering Drawings as Solid Models.....	111
4.3.1 The First Approximation Model	112

4.3.2 Method of creation.....	113
4.3.2.1 Raw data interpretation	114
4.3.2.2 Data analysis.....	114
4.3.2.3 Three-dimensional modelling.....	116
4.3.2.4 Output verification	118
4.3.3 Deficiencies.....	118
5. ENHANCEMENT OF FIRST APPROXIMATION MODEL.....	120
5.1 Introduction.....	120
5.2 Mathematical Description.....	120
5.3 Full Details of Method used.....	124
5.3.1 Scope.....	124
5.3.2 Description	124
5.4 Implementation of the Method.....	129
5.4.1 Modification to the Current First Approximation Model	
Program....	134
5.4.1.1 Changing the program into subroutine	135
5.4.1.2 Modifying edge attributes in data structure.....	140
5.4.1.3 Modification in CYCLES routine.....	141
5.4.1.4 Modifying TRANSF.....	143
5.4.1.5 Modification in relation with primitives	143
5.5 Discussion	146
6. WIREFRAME MODEL GENERATION.....	149
6.1 Introduction.....	149
6.2 Nodes.....	149
6.3 Analysis of Superimposed Edges on Engineering Drawing	152
6.4 Generation of 3D Edges.....	156
6.4.1 Dashed Lines.....	158
6.5 Generation of Isometric View.....	161
6.6 First Approximation Wireframe Model.....	161
6.6.1 Overall description.....	162
6.6.2 2D boundary intersection with horizontal/vertical lines	165
6.6.3 Removing extra lines from FAWM.....	168
7. METHOD OF COMPARISON.....	172
7.1 Introduction.....	172
7.2 Overall Description.....	172

7.3	2D Comparison.....	174
7.4	3D Comparison.....	177
7.5	Mapping of 3D differences to 2D	178
7.6	Extraction of 2D Loops.....	180
7.6.1	Boundary elements	182
7.6.2	Dashed lines.....	183
7.7	Identification of Primitives.....	185
7.8	Use of Recursion to Improve the Accuracy of the Model	186
8.	PRACTICAL RESULTS AND DISCUSSION.....	194
8.1	Programming Requirements	194
8.2	Input/Output	195
8.2.1	Input Options.....	195
8.2.2	Output format	196
8.2.2.1	Text output format	197
8.2.2.2	Graphic screen format	197
8.2.3	IGES output file	199
8.2.4	Error messages.....	199
8.3	Running different examples.....	200
8.3.1	Illustrative example.....	200
8.3.2	Program testing	211
8.4	Discussion	212
9.	SUGGESTIONS FOR FURTHER WORK.....	216
9.1	Discussion	216
9.2	Suggestions for Further Work.....	218
9.2.1	Input section.....	218
9.2.2	Increasing the object range	219
9.2.3	Implementation.....	220
9.2.4	CSG tree optimization's	221
10.	CONCLUSION.....	223
	REFERENCES.....	226
	APPENDICES	232
A.	EXAMPLES	232
B.	CAD/CAM 90 : state of the art	247
C.	A SELECTION OF COMMERCIALY-AVAILABLE SOLID MODELLERS....	254

LIST OF FIGURES

1.1	Traditional design method.....	17
1.2	Integrated system.....	19
1.3	Steps of using engineering drawings in production of 3D model	21
1.4	Expansion tree	23
1.5	Overview of the process.....	25
2.1	Types of modellers.....	31
2.2	An example of an ambiguous wireframe model	32
2.3	An interpretation of the ambiguous model.....	33
2.4	A different interpretation of the same model	33
2.5	Another possible interpretation	34
2.6	Impossible object A	35
2.7	Impossible object B	35
2.8	The wireframe model of the object of figure 2.11	36
2.9	Parametric definition of a curve.....	40
2.10	Parametric representation of a surface	40
2.11	A solid object	44
2.12	CSG element representation for the solid object of figure 2.11.....	44
2.13	(a) Set-theoretic (b) Non-regularized (c) Regularized boolean operations .	45
2.14	The CSG tree	46
2.15	Boundary representation of the solid of figure 2.11	47
2.16	Boundary description (topology).....	49
2.17	Winged-edge representation.....	50
2.18	Translational sweep	51
2.19	Rotational sweep	52
2.20	Variable cross section sweep.....	53
2.21	The effect of varying value of u	54

2.22 Octree representation.....	55
3.1 Orthographic projection.....	58
3.2 First Angle orthographic view	59
3.3 The three planes swung into a two-dimensional position.....	60
3.4 A solid object with direction of different views.....	60
3.5 First Angle orthographic views of the object in figure 3.4	61
3.6 Third Angle orthographic views of last example (by: BOXER)	62
3.7 Orthographic projection of primitives	63
3.8 Orthographic views of a simple prismatic object	64
3.9 Orthographic views of a complex prismatic object.....	64
3.10 Different types of arc's orthographic views.....	66
3.11 Different cases of line's orthographic views	67
3.12 Example of fictitious lines in engineering drawing.....	70
3.13 A simple Isometric view.....	71
3.14 Oblique and Oblique cabinet drawings.....	72
3.15 Two examples of one-point perspective	73
3.16 Two-point perspective	73
3.17 Three-point perspective.....	74
4.1 Input views.....	81
4.2 Ghost figures	82
4.3 Main components of the system	86
4.4 Types of entities and relationships defining the data structure	88
4.5 A uniform-thickness object.....	88
4.6 Flowchart of the algorithm	91
4.7 Ellipse mode	93
4.8 Higher order curve mode.....	94
4.9 Derive radius.....	95

4.10 Two cases to derive radius	95
4.11 Using auxiliary circle to find cylinder axis	96
4.12 Cutting vertex/edge.....	97
4.13 Face-loops.....	98
4.14 Incident faces	100
4.15 Views of an input primitive	102
4.16 Cone and its three principal views.....	103
4.17 Input views and its wireframe representation	106
4.18 Surface construction	107
4.19 Examples of pseudo faces (in bold).....	108
4.20 Segment counts (left: at the beginning, right: before backtracking)	108
4.21 Object "cut-out" from a "raw block"	111
4.22 Boolean tree of two-dimensional base views	113
4.23 Flowchart of the process	114
4.24 Data analysis terminology	115
4.25 Decomposition and merging stages in the analysis process	117
4.26 Primitive identification and 3D modelling steps	117
5.1 The decomposition tree	122
5.2 An example of decomposition tree for an object.....	123
5.3 Wireframe and first approximation wireframe in bold	125
5.4 Orthographic views	126
5.5 (a): 3D edges not on FAM boundary (b) : 2D picture of (a) completed with 2D boundary (in bold)	127
5.6 Overview of the system.....	128
5.7 Structure of modules of the software implementation.....	130
5.8 Summary flowchart of the MAIN program	131
5.9 Modules used in the FRSTAPM subroutine.....	139

5.10	An example of cycles finding	142
5.11	An example of circular pattern	144
5.12	Checking of the boundary for primitives	145
6.1	Relationship of 3D and 2D nodes (first angle orthographic views)	150
6.2	An example for ghost node that has not been produced	151
6.3	Different cases of superimposing of projections.....	152
6.4	Overlapping the pictures of two arcs.....	153
6.5	Distinction of 3D edges by three view projections	154
6.6	3D edges parallel to a main plane, having a straight line projection	154
6.7	3D edges not parallel to any main plane, having a straight line projection...	155
6.8	3D arc parallel to a main plane	155
6.9	Different cases of 3D edges with common start and end points	157
6.10	Example where dashed line information is not used	158
6.11	Solid object for dashed line example.....	159
6.12	Orthographic views of object in figure 6.11	159
6.13	Wireframe model of the example	160
6.14	A sample object.....	162
6.15	Orthographic views of object in figure 6.14	163
6.16	Extrusion of three views.....	164
6.17	Intersection of a line with boundary of a view.....	165
6.18	Example of creating 3D edges of FAWM	167
6.19	Different cases of finding the inside sections of a line	168
6.20	Orthographic views	169
6.21	First approximation wireframe model with extra lines.....	170
6.22	Final result of first approximation wireframe model	170
7.1	Sub-object is completely within the first approximation model.....	173
7.2	Sub-object has common planes with the first approximation model	174

7.3	First approximation wireframe and its orthographic views for the object of figure 6.11	175
7.4	Two-dimensional differences of two models	176
7.5	Three-dimensional differences of two models	177
7.6	Another example of three-dimensional differences extraction.....	178
7.7	Mapping 3D differences into 2D orthographic views	179
7.8	3D differences and their 2D orthographic views.....	181
7.9	Completed orthographic views of sub-objects.....	182
7.10	Dashed line application example.....	183
7.11	Wireframe and first approximation wireframe models.....	184
7.12	Completed boundaries for sub-objects.....	185
7.13	Input first angle orthographic views	187
7.14	Wireframe model for solid object of fig. 7.15	188
7.15	A complex example	188
7.16	Sub-objects created at level 1	189
7.17	Wireframe model of sub-object 2.....	189
7.18	CSG tree for sub-object 2	190
7.19	Decomposition tree.....	191
7.20	Improvement of the model from top to bottom	192
8.1	Frame 1 : Input views for visual check	201
8.2	Frame 2 : Input view and wireframe model	205
8.3	Frame 3 : First approximation wireframe model on top of W.M.	206
8.4	Frame 4 : 3D edge differences and orthographic views of next sub-object..	206
8.5	Frame 1 for sub-object 1	207
8.6	Frame 2 for sub-object 1	209
8.7	Frame 3 for sub-object 1	210
8.8	Frame 5 : Decomposition tree.....	210
8.9	Solid object built by solid modeller	211

8.10 First angle orthographic views of the produced object by solid modeller ...	212
A.1 Example 1	232
A.2 Example 2	233
A.3 Example 3	234
A.4 Example 3 (continued).....	235
A.5 Example 4	236
A.6 Example 4 (continued).....	237
A.7 Example 5	238
A.8 Example 6	239
A.9 Example 7	240
A.10 Example 8	241
A.11 Example 9	242
A.12 Example 10.....	243
A.13 Example 10 (continued)	244
A.14 Example 11.....	245
A.15 Example 11 (continued)	246

LIST OF TABLES

2.1	Data representation of wireframe model	37
-----	--	----

Chapter One

INTRODUCTION

Chapter One

INTRODUCTION

1.1 Introduction

In traditional engineering design, an object is represented by a set of conventional 2D drawings consisting of orthographic views. These drawings are interpreted and the 3D physical model is built by another specialist person. This physical model is tested against desired properties and characteristics. If any of these tests fails then these sequences, including steps which are not mentioned here, will be repeated until a good design is achieved (figure 1.1).

Improvements in the technology of making integrated circuits as dense as possible and reduction in prices within recent years, have encouraged the rapid development of Computer Aided Design and Drafting (CADD). Because of access to fast and cheap memories and processors, and emerging parallel processing techniques, most of the algorithms that needed a large amount of memory and were time consuming are now considered seriously, and new algorithms are under development.

With the new facilities mentioned, the following questions arose:

Do these tedious, costly and time consuming processes really need to be done?

Why not generate a sophisticated computer model and analyse it, and then pass the results directly to the production line?

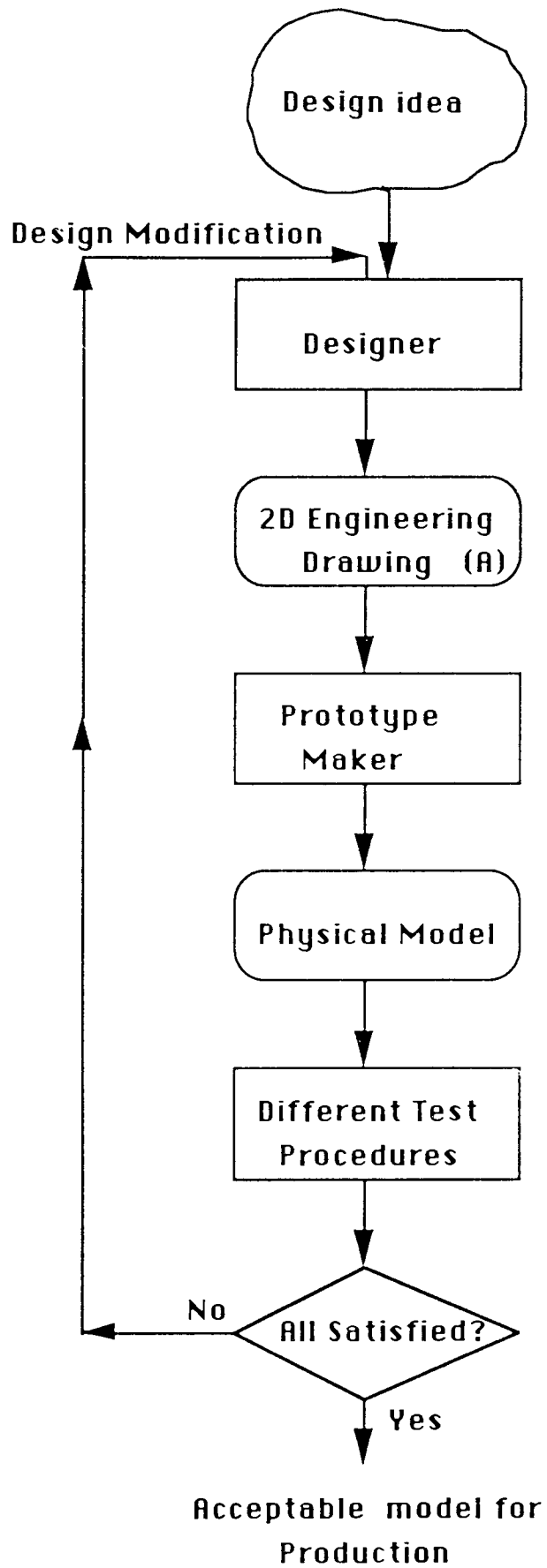


Figure 1.1 Traditional design method

These questions were addressed and considerable progress towards a solution was made. Nowadays, traditional design methods are gradually being replaced by the object's 3D model on computer, called "Solid Model". 3D solid model is the basis for a complete product definition, from concept through production. Many designers prefer to utilise these new cheap tools and facilities for their drafting, 3D design, prototype modelling, different analyses and testing. There is no doubt that the ability to use the same geometric model to generate 2D models for documentation and for analysis and manufacturing simplifies and speeds up the whole design process.

Total benefits will not be achieved by installing a few solid modellers alongside existing CAD systems. As with all manufacturing technology, the higher the level of integration, the greater the benefits. Major suppliers place great emphasis on integration, by utilising networks of multi-tasking workstations. Therefore further steps have been taken towards integrating the whole manufacturing system, known as "CIM".

The integration of design with manufacturing demands a more formalised approach to system purchase and developments. Systems from CAD through CAM to overall production management have to be configured to share common data and communicate with one another. Examples of systems which are covered by this integration are: prototype design, analysis (e.g. mass property calculation, stress and thermal analysis), testing, bill of materials, documentation, process planning, casting, NC machines, Robot control, assembly lines, quality control, accounting, and purchasing (figure 1.2).

Because of the increasingly important role of solid modelling and the trend of industry

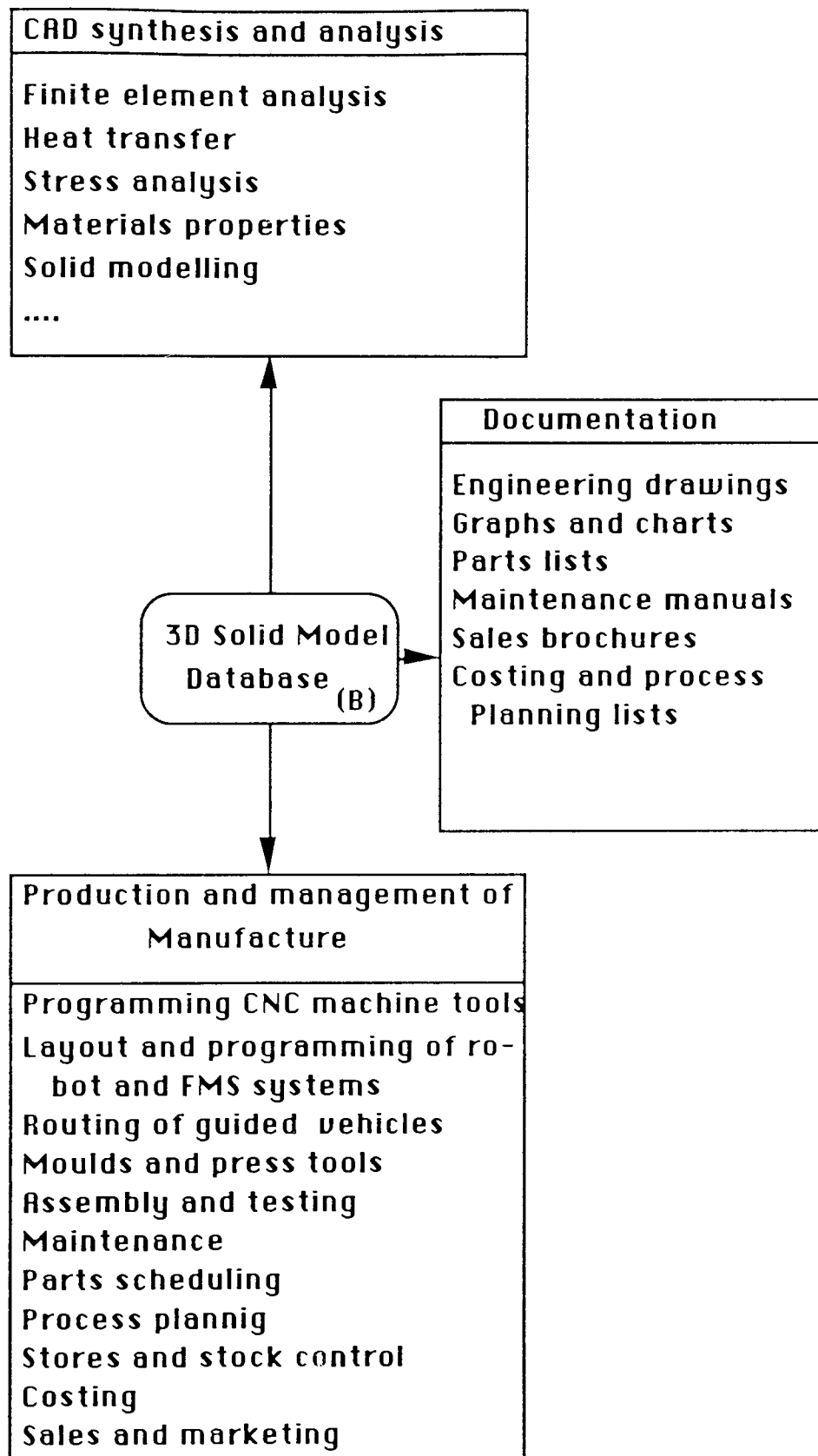


Figure 1.2 Integrated system

in utilising modern and cheap technologies, together with the use of highly efficient software methods and algorithms in faster manufacturing of products, it becomes evident that ordinary methods of design are soon going to be obsolete. With these facts, again these question arise:

What happens to the vast number of parts designed so far, and are they frequently being used in everyday products?

What happens to them, if they are going to be produced or modified under new systems?

Should they all be redrawn and the huge number of existing engineering drawings be ignored?

Therefore, a possible solution to these questions is to find a method to convert automatically all 2D engineering drawings ("A" in figure 1.1) to 3D solid model data ("B" in figure 1.2), utilising the same new technologies [Iwata *et al.*,1988] as shown in figure 1.3. Currently, some sophisticated packages are available that can accept the raster scanned data from 2D drawings and allow interactive graphic editing facilities, and provide 2D vectorised data in form of edge list and node coordinates on computer files. However, transformation from 2D to 3D is still a problem, and one which is not so straightforward since it needs more accurate considerations and some intelligence. This is because, in order to ease the representation of a 3D object in form of 2D drawings, some items of information about the object are missed and are left to the knowledge and experience of the interpreter. Therefore, the 2D to 3D conversion is not a one to one function, and does not necessarily have a unique solution in generating a solid model.

Many people have tried to solve this problem in different ways, and several algorithms have been considered and many techniques developed. But none has yet been successful in presenting a complete automatic algorithm, available in the form of commercial software.

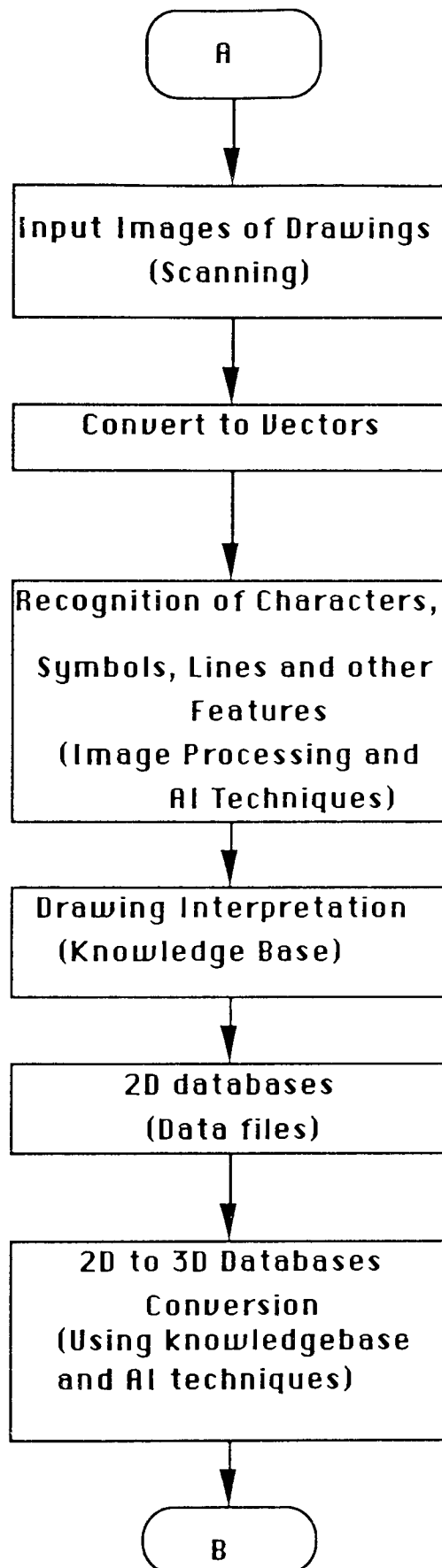


Figure 1.3 Steps of using engineering drawings in production of 3D model

1.2 Aim of the project

To overcome the transformation of 2D engineering drawings into 3D databases, a method has been developed by A. Kargas, P. Cooley and T.H.E. Richards [1988]. The method produces a 3D solid model, called the "First approximation model" (FAM).

Due to the nature of the method used for production, this model does not represent exactly the original object. Therefore some methods had to be implemented to enhance the model towards the real object: this is the aim of this project. To attain the goal the idea of expanding an object, as depicted in figure 1.4, was considered. In this regard the following activities were involved:

- Review of existing techniques for converting 2D databases to 3D databases.
- Study of algorithms and methods which had been used in a recent project to produce the "First approximation model".
- Investigation of the program's source code module by module, then applying necessary modifications to the program to make it a callable subroutine, in order to allow its use as a tool for production of the First approximation model, throughout the project.
- Adding some new primitives to the program.
- Modifying part of the data structure for accepting other information about edges, such as dashed line information.
- Development of algorithms and implementation of the required software to cover the idea proposed in figure 1.4, for which the overview flowchart is shown in figure 1.5.
- Writing programs for demonstrating the algorithm process graphically on the screen, including isometric views of the wireframe model.
- The performance of program has been checked by feeding the output of program, which is an acceptable data file, to a solid modeller, then observing the generated solid on the screen. Also the output orthographic views produced by the solid modeller have

been compared with the initial 2D inputs. The results have been satisfactory and proved the proper working of the idea.

Some methods have also been developed for extracting sub-objects. Input data is used to produce the pseudo wireframe model of an object, and the boundary of the same input is also used for generation of another model called the first approximation wireframe model. Some techniques are used to extract some sub-objects which are either primitive or prismatic, by comparison of the two wireframe models. Extracted sub-objects are altered in the CSG tree, produced by the first approximation model, in such a way as to create the object's model at the root.

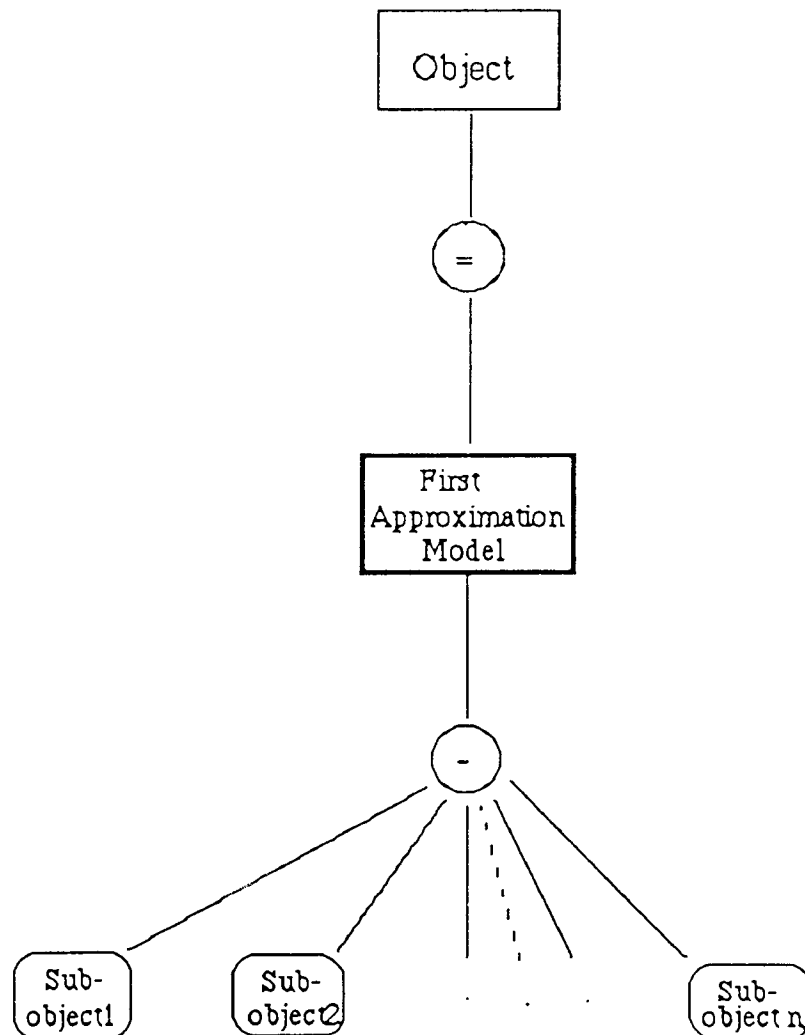


Figure 1.4 Expansion tree

In the proposed algorithm, based on figure 1.4, and assuming that these first approximation models of sub-objects, primitives and prismatic objects (which are FAMs) are all solids, then the object's model at the root of tree produced by this method is also solid. Although the wireframe model has been named and used, the process has relatively little back tracking, as will be seen in future chapters; further, it does not mean that the process is based on wireframe modelling or that the output is not guaranteed to be a solid. The same process is recursively repeated for each nonprimitive sub-object, until an acceptable solution is achieved. Although for most of the examples an accurate model is found by the program, the feedback method used in the previous project [Kargas, 1988] is also considered as a last resort for worst cases. In the feedback method orthographic views of the output model are compared with initial input views for detecting differences.

1.3 Project's Environment

The aim of the project was to improve upon the model created from previous work. Hence, in order to maintain the unity between programs and to save time, it was decided to adopt the same environment.

The program is written in FORTRAN-77 and has been developed on an Apollo Domain workstation (model DN3000). The workstation consists of a 15 inch colour graphic display with 1024 x 800 pixel resolution, a 1.2 Mbyte floppy drive (5.25 inch), a built in hard disk with a capacity of 72 Mbyte, and 4 Mbyte of main memory. An Epson Ex-1000 printer and a mouse are connected to the system. The graphic library available, which has been used in implementation of the program, is called GMR (Graphics Metafile Resource). The operating system running on the machine is AEGIS. The solid modeller which is used as a tool for testing the program's result is BOXER, from PAFEC Ltd. It is available on both Apollo Domain and VAX machines. Some of the figures in this thesis have been created by BOXER or DOGS (another product from

PAFEC Ltd. for 2D drafting) running on VAX. They have been captured from the Macintosh screen, by emulating the Tektronix graphic terminal model 4014, and have been saved in MacPaint file format.

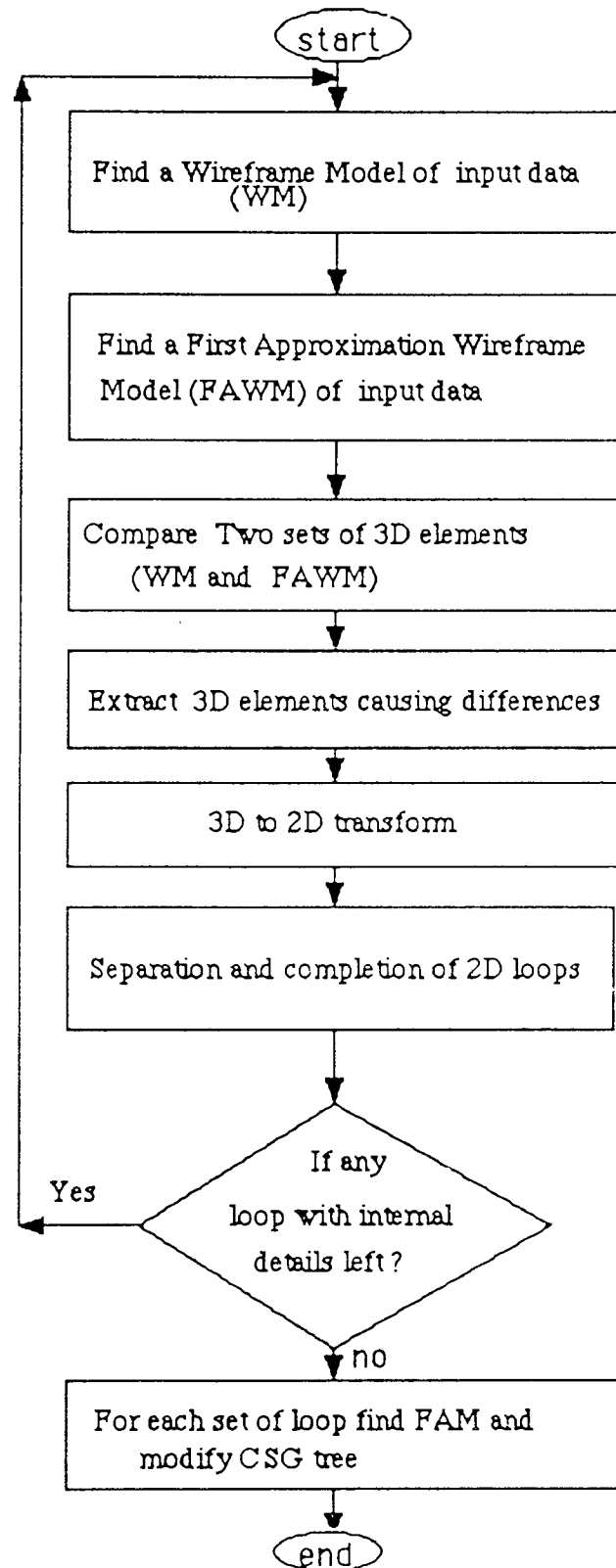


Figure 1.5 Overview of the process

1.4 Thesis Plan

The aim of this project is to convert two-dimensional engineering drawings into three-dimensional solid model, by enhancing the currently produced "first approximation model". The ultimate goal is implementation of a method to do this automatically by computer and generate an output model in the form of CSG for a solid modeller.

In chapter two some basic ideas of geometric modelling and different types of modellers are explained. Advantages and weaknesses of modellers are discussed, and the selection of CSG among them for representing the output model is motivated. Input to the system is a set of 2D orthographic views of an object, using engineering drawing conventions. Some of these concepts are therefore explained in chapter three.

Work done regarding the conversion of 2D models to 3D are briefly reviewed in chapter four; some of the interesting ones are then studied in more detail, with a final comparison. A more recent work, which is a basis for this project, is also explained here and deficiencies of the model are indicated.

The proposed new approach to the problem is prescribed in chapter five. Its basic idea and mathematical description are given, then the full detail of the development method is explained together with pictorial examples. Modifications done to the current "first approximation model" generator in order to make it ready for use as a subroutine are presented, and brief description of the modules are provided. To implement the proposed algorithm, in addition to the first approximation model, the process is divided into three parts: wireframe model generation, first approximation wireframe model generation, and comparison method. A list of the different modules designed for the implementation process, their interrelationship, and summary explanations of each are presented in this chapter. The wireframe model and first approximation wireframe

model generators are described in chapter six. The effect of dashed lines in the 3D edge production process is also considered here.

Description of the implementation process is ended by discussing the method of comparing the two wireframe models in chapter seven. This covers the algorithm for extracting sub-objects. The improvement in accuracy of the output model by using a recursive method is considered. Finally the preparation of BOXER files and decomposition trees, and the role of dashed lines are explained in this chapter.

At the beginning of chapter eight, execution of the program, input and outputs, different options and error messages are explained. Results of a wide range of examples, which have been run by the program and are presented in appendix A, are discussed here. In chapter nine different aspects of the method are discussed; further work is suggested for achieving an advanced package.

Chapter Two

GEOMETRIC MODELLING **THEORY**

Chapter Two

GEOMETRIC MODELLING THEORY

2.1 Introduction

Geometric modelling came into use in the early 1970's. This term refers to the collection of methods used to define the shape and geometric characteristics of an object, and covers fields such as computational geometry and solid modelling. A model is a substitution or representation for a physical thing, such that all relevant questions about the real thing can be referred to this model. A good model is one which is able to provide the answers to questions and acts as much as possible like the original entity. The model should be designed to deal with the specific questions to be asked of it. Many fields of knowledge are used in geometric modelling, for example analytic and descriptive geometry, topology, set theory, numerical analysis, and linear algebra can be mentioned. Different aspects of geometric modelling can be summarized as follows:

Representation : The physical shape of an object is given and presumed to be fixed; a mathematical approximation is computed once.

Design : An artifact must be created to satisfy some functional and/or aesthetic objectives.

Rendering : At any stage an image of the model may be needed so that it can be comprehended.

Computer models are now a viable alternative to physical models, because they can be

cheaper both to construct and to use for experimental work. Thus, products can be manufactured more cheaply and with better qualities.

2D drafting and NC verification were early applications of geometric modelling. Computer graphics, computer aided design and manufacturing (CAD/M), computer aided engineering (CAE), robotics, computer vision, knowledgebase technologies and AI are effective resources that support developments in geometric modelling. Other application areas are in the construction of finite element models, stress analysis, mass properties calculation, process planning, quality checking of part production, VLSI design, animation, flight simulation, reconstruction of objects from visual information, architectural engineering, and so on. For many applications the geometric modelling of a physical object may require the complete description of surface, texture, colour, or it may include only a subset of the object's physical attributes.

Before proceeding further, some terms are explained :

Domain : It is the set of objects that a representational scheme can model. For example, a scheme's domain might include only objects that are plane faced, convex, and simply-connected polyhedra.

Validity : Criteria for specifying validity are variable. In a CAM system designed to produce machinable parts, an object with an internal cavity is invalid. Moreover, there is a great range of what are called "nonsense" objects, which would be invalid in almost any modelling system.

Completeness : It is a measure of the model's ability to respond to a broad range of analysis or, in other words, its ability to answer geometric questions. As an example, wireframe models cannot provide answers about surface normals or volumes. Therefore they are incomplete representations of solids.

Uniqueness : This is a key factor in determining the equality of objects. Most representation schemes for geometric objects are nonunique: first, because

substructures in representation may be permuted (e.g. $A \cup B \cup C = A \cup C \cup B$) and, secondly, because distinct representations may correspond to equivalent objects which are differently positioned.

Conciseness : This refers to the amount of data needed to define an object in a particular representation . The more concise a model is, the more convenient it is to store or transmit, and the less redundant data it contains.

User friendliness : Some measure of the ease of creation of valid models either in interactive mode or batch mode.

Efficacy : How adequate and/or flexible a model representation is to downstream applications as well as general geometric analysis.

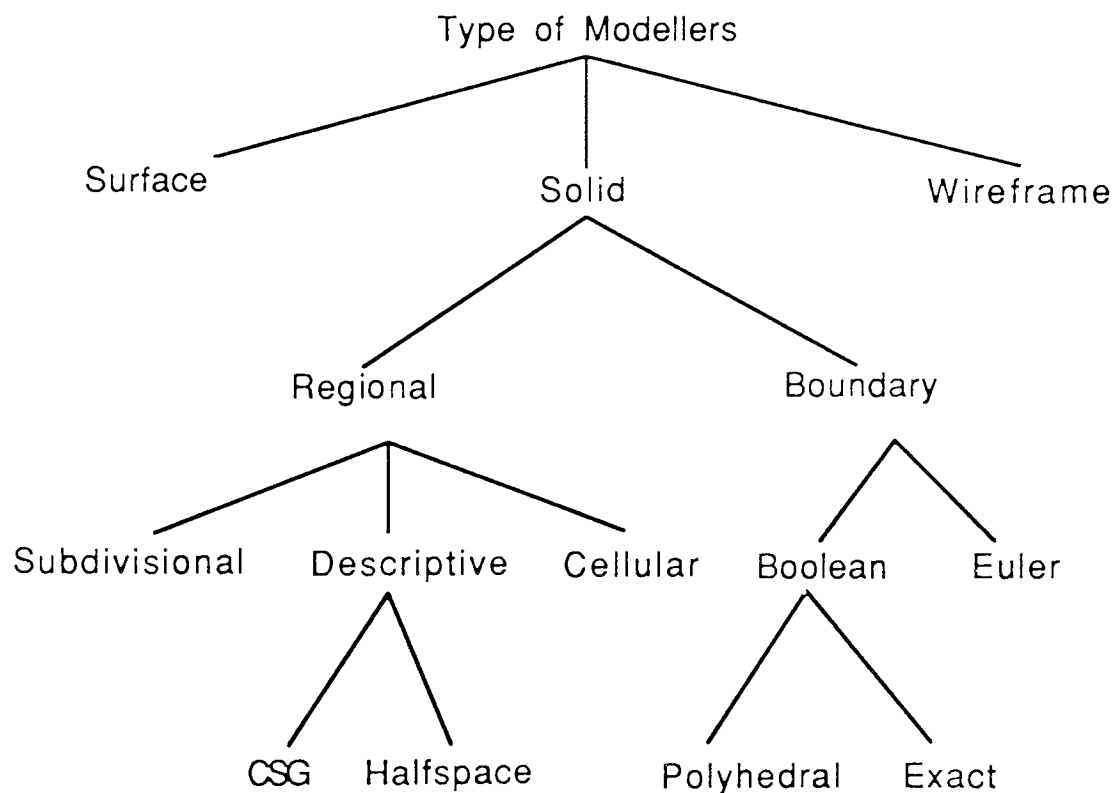


Figure 2.1 Types of modellers

Depending on the application, different models are used. Computer aided geometric modelling systems which have been developed are based on: curves, surfaces, or

solids. Of these, solid modelling seems to be more powerful, and is widely used. Types of modellers [Braid, 1986] can be grouped as shown in figure 2.1.

2.2 Wireframe Technique

This kind of modelling is one of the earliest and easiest ways of representing objects. The model is composed of lines and curves defining the edges of an object and a set of points determining the start and end of these edges, which are called "vertices" or "nodes", as shown in figure 2.8. 2D wireframe models are used for drafting. Current wireframe modelling systems are capable of constructing 3D representations. Since there is no information about surfaces, 3D models may be ambiguous. A typical example of this kind is shown in figure 2.2.

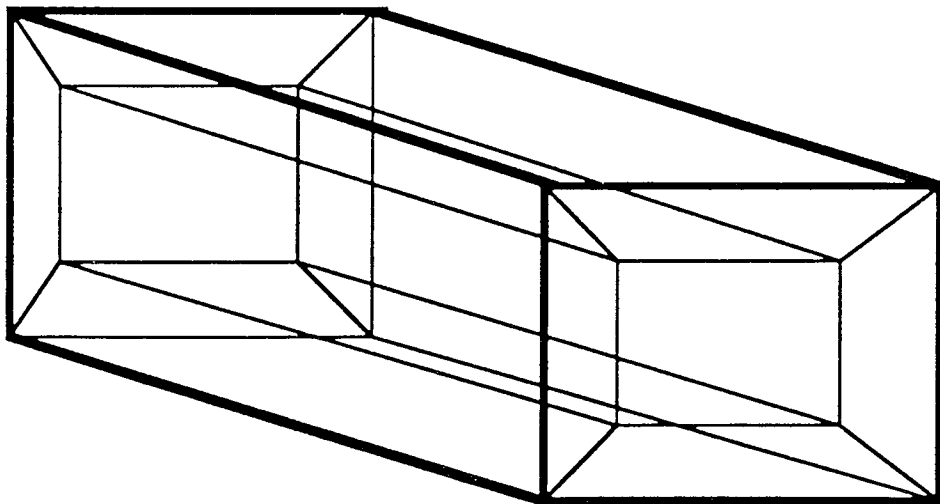


Figure 2.2 An example of an ambiguous wireframe model

Figures 2.3, 2.4 and 2.5 show three different possible interpretations of the ambiguous

wireframe model.

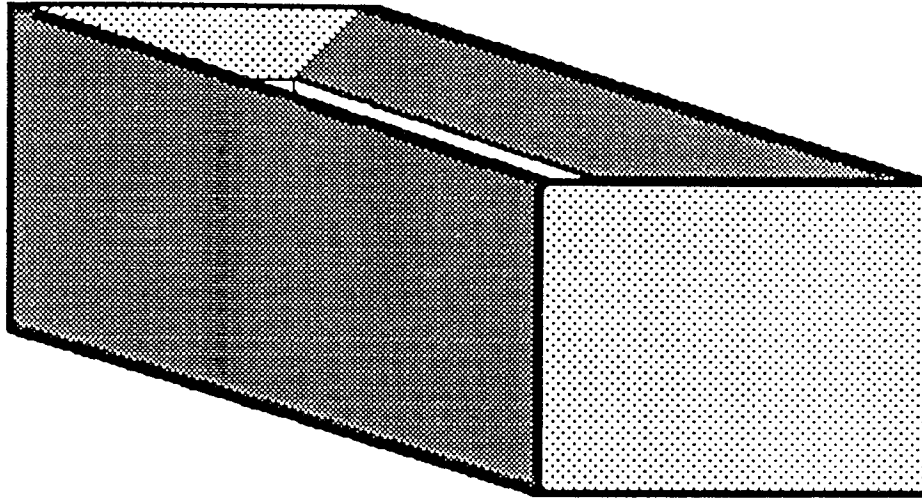


Figure 2.3 An interpretation of the ambiguous model

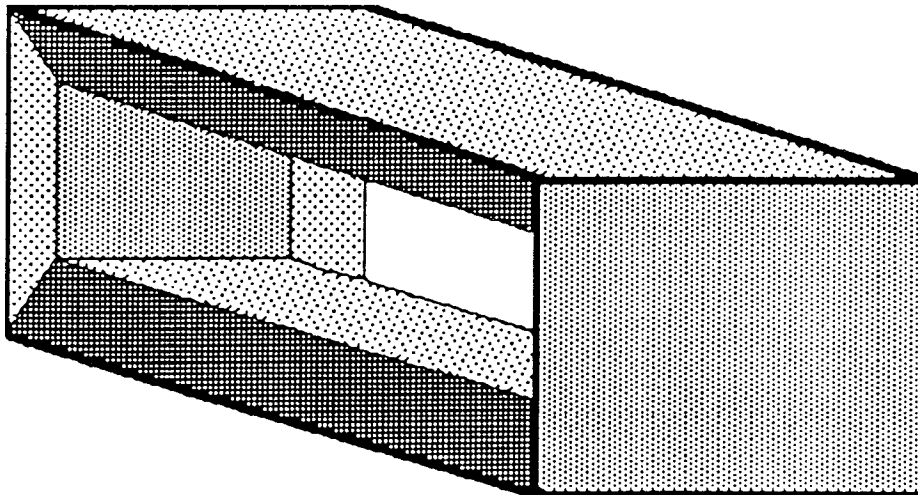


Figure 2.4 A different interpretation of the same model

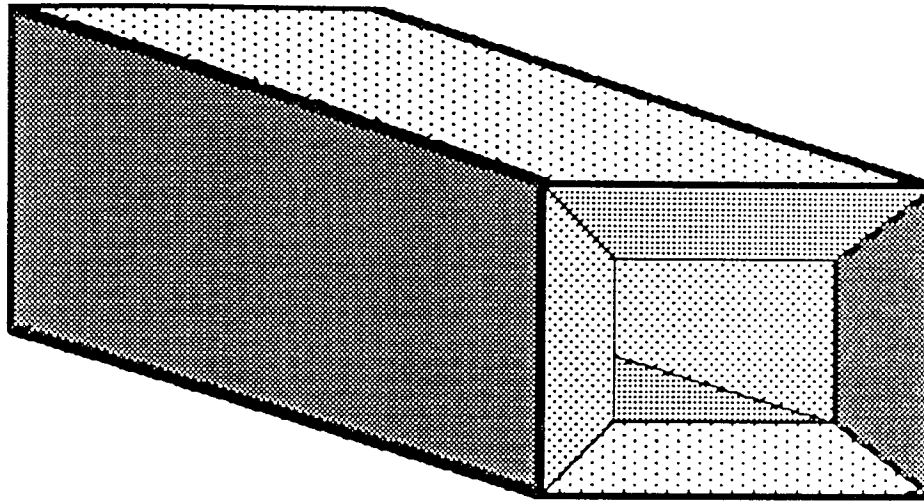


Figure 2.5 Another possible interpretation

In this model there is no way to ensure the integrity of the object. Briefly, problems with 3D wireframe modelling are:

- a) The possibility of creating ambiguous models.
- b) Creating nonsense, incomplete, or impossible objects. Two different examples of this kind are shown in figures 2.6 and 2.7.
- c) The profile lines or silhouette are not usually derivable from the model.
- d) Entering data, and design with this model are very tedious and error prone processes. (figure 2.8).
- e) Holes are difficult to locate and complex shapes are hard to describe (figure 2.8).

Figure 2.8 shows the wireframe model for the solid object of figure 2.11, which has 26 nodes and 39 edges.

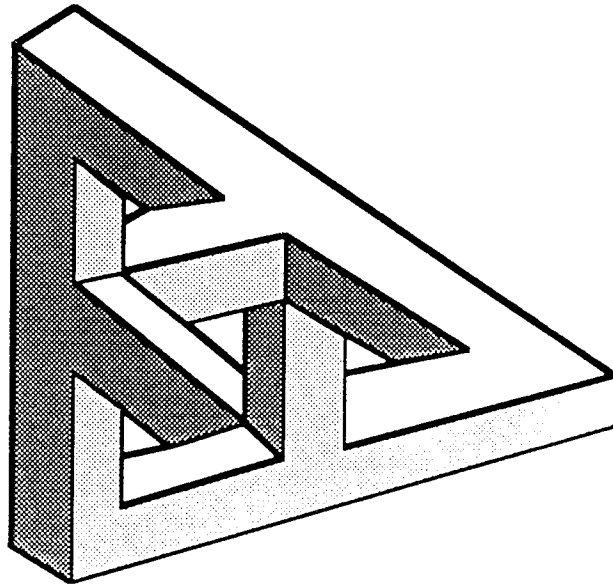


Figure 2.6 Impossible object A

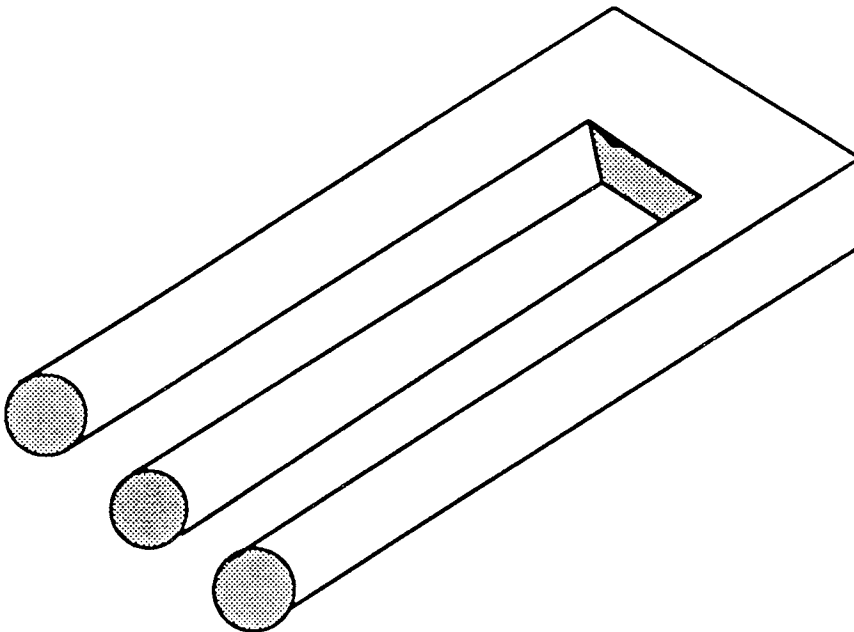


Figure 2.7 Impossible object B

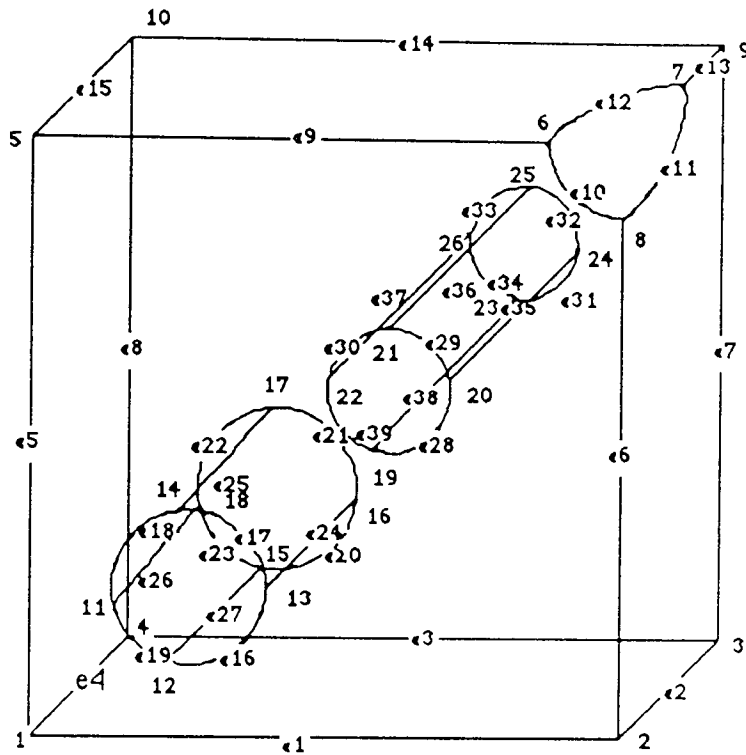


Figure 2.8 The wireframe model of the object of figure 2.11

2.2.1 Data Representation

In order to represent wireframe models in computers it is necessary to store information about the location of vertices in space (geometry), the way that edges are connected together (topology) and other data about the type of edges. Geometry data are maintained by assigning a number to each vertex and storing 2D or 3D coordinates of them into corresponding vectors, depending on the 2D or 3D nature of the wireframe. Topology are shown by keeping the index to a geometry table of start and end vertex of each edge in a separate table. Other information about the type of edge (e.g. line/arc) are also recorded. In the case of arc type other attributes such as centre of arc, its direction (clockwise or anti-clockwise) must be recorded. Normally in this case another table is used for this purpose and only its index is saved in the edge list. Throughout the

project, arcs in 3D wireframes are assumed to be located in flat surfaces parallel to the main planes. Therefore, this plane can be defined as another attribute of arcs in the table (table 2.1). Since arcs are defined by their start and end nodes, therefore arc table is needed to keep the coordinates of their centres, and directions. In this situation for special case of half-circle, the position of circle's plane must also be specified.

Edge no.	Start	End	Type
1	11s	11e	11
2	12s	12e	12
3	13s	13e	13
n	1ns	1ne	1n

(a)- Topology

Vertex no.	X	Y	Z
1	x1	y1	z1
2	x2	y2	z2
3	x3	y3	z3
4	x4	y4	z4
m	Xm	Ym	Zm

(b)- Geometry

Arc no.	Xctr	Yctr	Zctr	Dir	Plane
1	Xc1	Yc1	Zc1	D1	P1	O1
2	Xc2	Yc2	Zc2	D2	P2	O2
k	Xck	Yck	Zck	Dk	Pk	Ok

(c)- Arc table

Table 2.1 Data representation of wireframe model

2.2.2 Definitions

Ghost edges : A ghost edge is an edge drawn on a surface with two adjacent faces on the same surface. The two faces can be merged into one by eliminating the ghost edge. Such ghost edges are not found in a correct wireframe model but may be tolerated.

Tangency edges : A tangency edge separates two faces on two different surfaces, but the surfaces have a first order continuity along the tangency edge. Traditional drafting views do not draw these edges. But in the project, the program expects the presence of such edges.

Silhouette edges : A silhouette edge is a notion associated with a view of the solid. It is the locus on the surface of points where the surface normal is perpendicular to the projection direction. The projection of the silhouette edge is the outline of the surface. A silhouette edge is a ghost edge.

Ghost face : A ghost face is a face that has been created by ghost edges.

2.3 Surface Modelling

The problems of wireframe modelling were largely eliminated by the development of surface modelling. In this method, a surface skin is fitted on top of a wireframe skeleton. When surfaces are explicitly included in the computer model, fewer impossible objects can be produced, and holes can now be located easily and unambiguously. It should be made clear that the term "surface modelling" differs from the "boundary representation", which will be discussed under solid modelling below. Here it refers to the collection of methods and algorithms used to define a surface in the computer. Some application areas of surface modelling systems are in the design and description of car bodies, aeroplane wings, turbine blades, ship hulls, all kinds of ducts, blends, chamfers, and fillets.

There are still some difficulties in using surface modelling to define solids. Designing with surfaces is tedious because there is no higher level of primitives. There is no guarantee that the surfaces bound a solid object and therefore the model may be incomplete. Thus some operations like mass properties calculation cannot be done. The lack of topological information (connectivity) is another deficiency of surface modelling.

2.3.1 Parametric Surface Definition

In this method, three-dimensional surfaces are specified by interpolation or approximation of two or more parametric space curves. A parametric curve is defined by the equation :

$$r=R(u)$$

In this equation r is a position vector of a point belonging to the curve which is defined as below:

$$R(u) = [X(u), Y(u), Z(u)]$$

Then a part of curve can be described by limiting the values of u ($u_1 \leq u \leq u_2$) as illustrated in figure 2.9.

To describe a surface, two parameters are used :

$$r = R(u,v)$$

$$R(u,v) = [X(u,v), Y(u,v), Z(u,v)]$$

The ranges of variation for u and v are bounded by some specified values (normally between 0 and 1). An example of this kind is shown in figure 2.10.

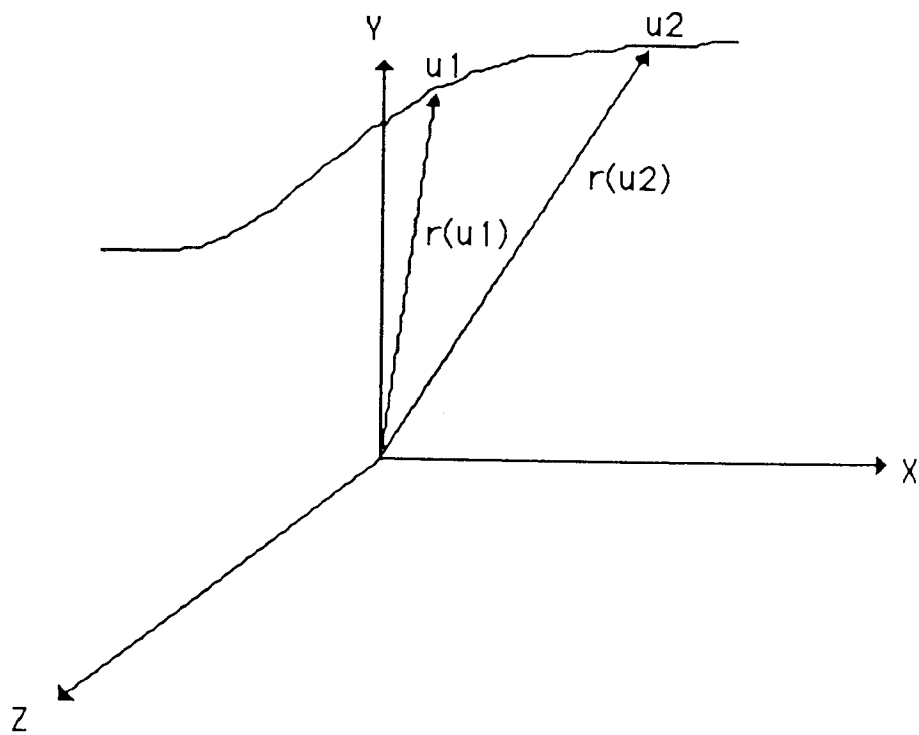


Figure 2.9 Parametric definition of a curve

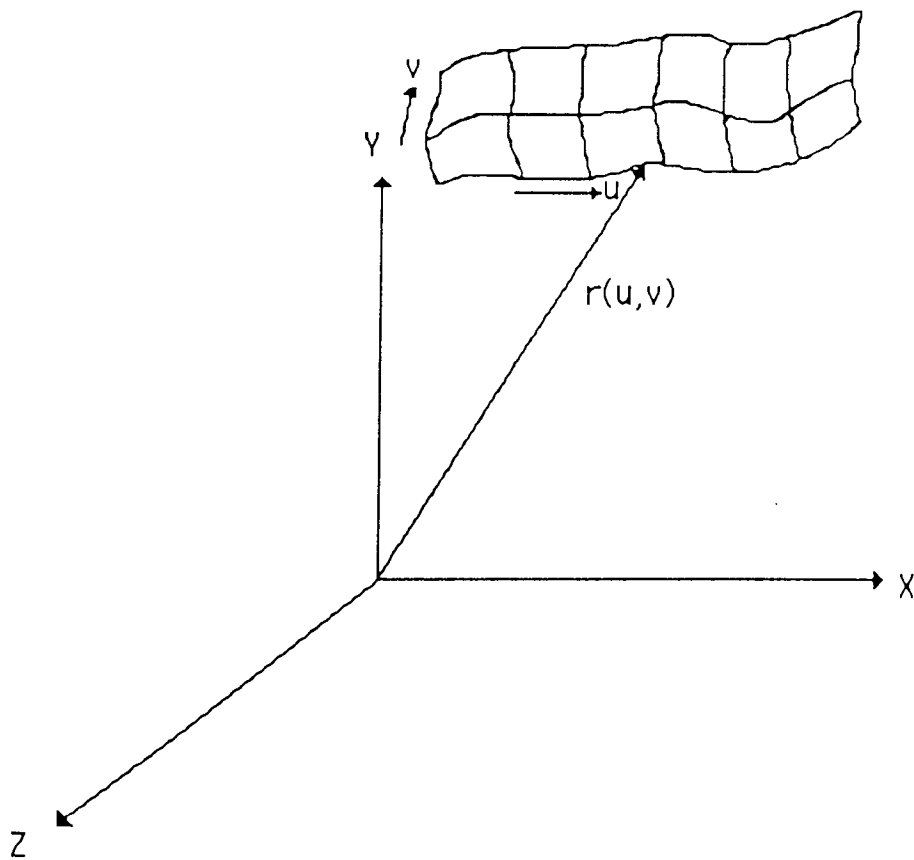


Figure 2.10 Parametric representation of a surface

2.3.2 Types of Modelling

To design complex surfaces, several methods of interpolation or approximation have been used. Most of these methods are based on cubic and rational polynomial segments. Some others use higher degree polynomials. Two basic types of surface modelling are currently used [Goldman, 1987]: *Transfinite interpolation* and *Discrete approximation*.

2.3.2.1 Transfinite Interpolation

In this method a surface must be constructed to pass through a given collection of curves. Two methods are now being used: local and global. In a local method a large well-behaved surface can be made by piecing together many patches. Each patch is defined by, and interpolates to, its four boundary curves. In the global method any arbitrary rectangular mesh of curves can be interpolated by using cubic cardinal splines. Coons [1964] and Gordon [1969] developed, respectively, the local and global methods.

2.3.2.2 Discrete Approximation

A relatively small set of points, called control points, is defined. Then, the system generates a surface to approximate the shape described by these points. This method also allows the use of local and global methods. One of the global methods is Bezier [Bezier, 1972,1974], in which Bernstein polynomials are used to create a single polynomial surface (Bezier patch) to approximate all the given data [Forrest, 1972]. There are two basic problems with this method: one is that moving any control point affects the entire surface, and the second is for a large number of data points, high degree polynomials are required.

One of the local methods uses a B-spline blending function to construct piecewise, polynomial surfaces [Riesnfeld, 1973]. In this method the effect of any control point is local and additional data do not need to increase the polynomial degree because the surfaces are piecewise polynomials. The use of B-spline guarantees smoothness between adjacent patches.

Sculptured surfaces were developed to replace the classic lofting techniques of designing surfaces. The well known parametric cubic patches of Coons [1964] and Ferguson [1964] and the Bezier techniques have been most successful. A synthesis of sculptured surfaces with the wireframe method may be considered as a future solid modeller.

2.4 Solid Modelling

To overcome the problems with wireframe and surface modelling, this original technique was developed. It is an important aspect of geometric modelling which is used to create and communicate shape information. The goal of solid modelling is to create unambiguous and complete geometric representations for 3D solid objects. By definition, a solid is a three-dimensional object with a well defined inside and outside separated by a boundary. There have been several approaches for generating and storing computer models of solids: boundary representation, constructive solid geometry, sweep representation, octree encoding, and logical operations on half spaces can be mentioned as examples. Some of them are explained in the following sections.

2.4.1 Constructive Solid Geometry (CSG)

In this method a solid object is defined in terms of simple objects called primitives, such as *cube*, *cylinder*, *sphere*, *cone*, *torus* and so on. It is expressed by a binary

tree called the CSG tree, in which the root represents the object and the leaves are simple primitives. Also each node of the tree describes a regularized boolean operation (*Union, Difference, Intersection*) that operates on its branches and makes a more complex intermediate object.

There are differences between regularized boolean operations and those used in classical set theory. The straightforward application of set-theoretic boolean operations to the set of points defined by a three-dimensional solid may lead to anomalous results (e.g. dangling edge, as illustrated in figure 2.13b). To avoid these problems the boolean operations are refined in such a way that they operate on and produce regular sets; the refined operators are indicated by the "*" superscript. Most concepts of CSG, including regularized boolean operators, primitives and boundary evaluation, were first introduced by Voelcker and Requicha [1977]. The CSG method is very concise and each binary tree is guaranteed to represent a unique, physical solid, provided that the primitives themselves have complete, unambiguous description. Moreover, boolean operations can be used to reflect the manufacturing process and lead directly to a process plan; it is also useful for applications such as mass properties calculation. Figure 2.11 shows a solid object, and its CSG element representation is shown in figure 2.12. The CSG construction tree is given in figure 2.14.

The primitives that have been mentioned so far can be defined in the form of a boundary representation, which will be described later or which can be expressed in terms of other lower entities called "half-spaces". A half-space is generated by an infinite surface which divides the 3D space into two parts, and may be specified by an inequality (e.g. $x \geq 0$).

Therefore, simple objects as well as primitives can be defined by the intersection of a number of half-spaces. For example, a unit cube may be represented as the intersection of the following half-spaces:

$X \geq -1$, $X \leq 1$, $Y \geq -1$, $Y \leq 1$, $Z \geq -1$, $Z \leq 1$
 and also the half-space $X^2 + Y^2 + Z^2 \leq 1$ represents a unit sphere.

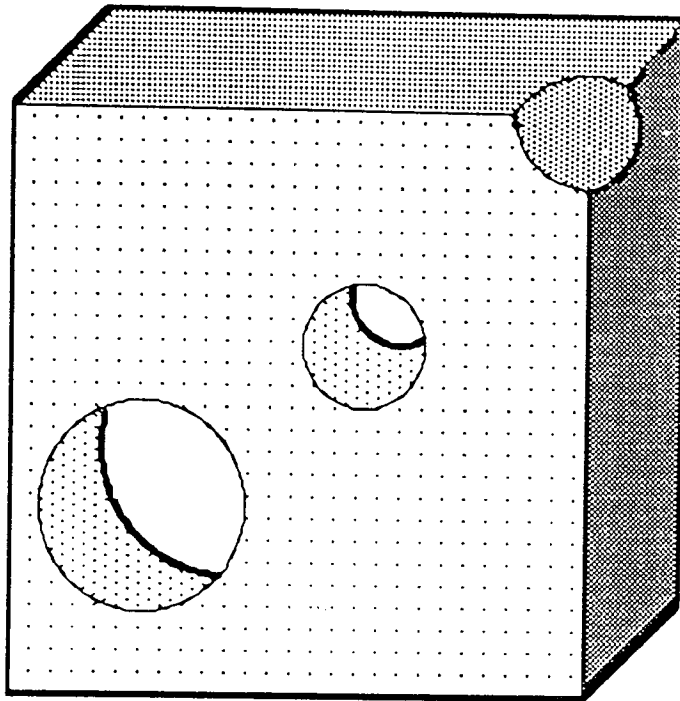


Figure 2.11 A solid object

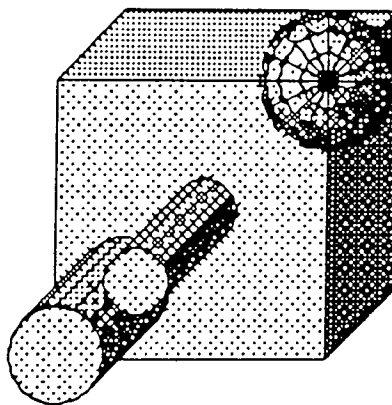


Figure 2.12 CSG element representation for the solid object of figure 2.11

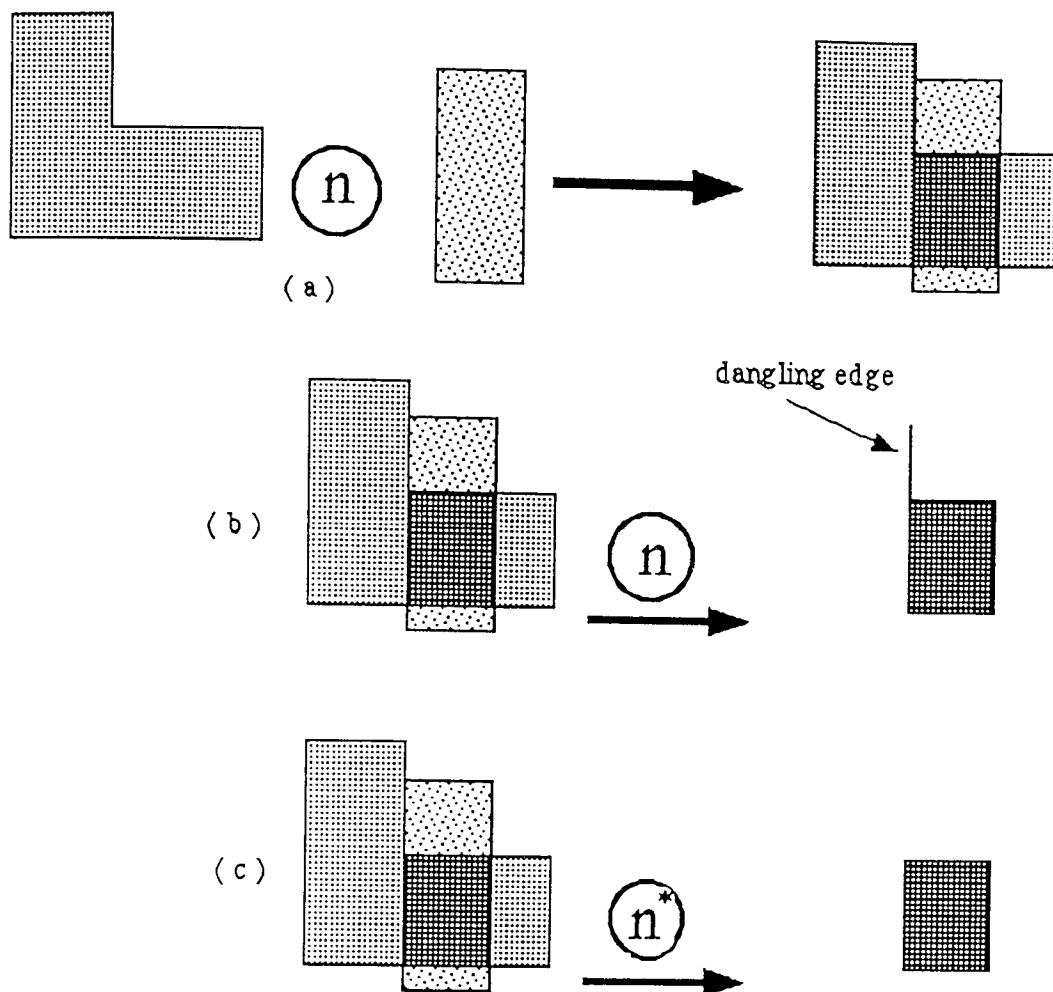


Figure 2.13 (a) Set-theoretic (b) Non-regularized (c) Regularized boolean operations

2.4.2 Boundary Representation

This method is closer to surface modelling, but here only complete solids are represented and topological information is also stored. Therefore a solid is shown by its boundary surfaces (geometry) and the specification of how these surfaces are joined together (topology). The boundary of a solid is the primary interface between the solid and surrounding environment. Boundary models are always deduced from the CSG model. Some solid modellers based on boundary representation use CSG as input to guarantee physical integrity and use boundary information in the data base to provide explicit geometry and topology. An advantage of such systems is that the user can easily work with high level solid primitives rather than curves and surfaces.

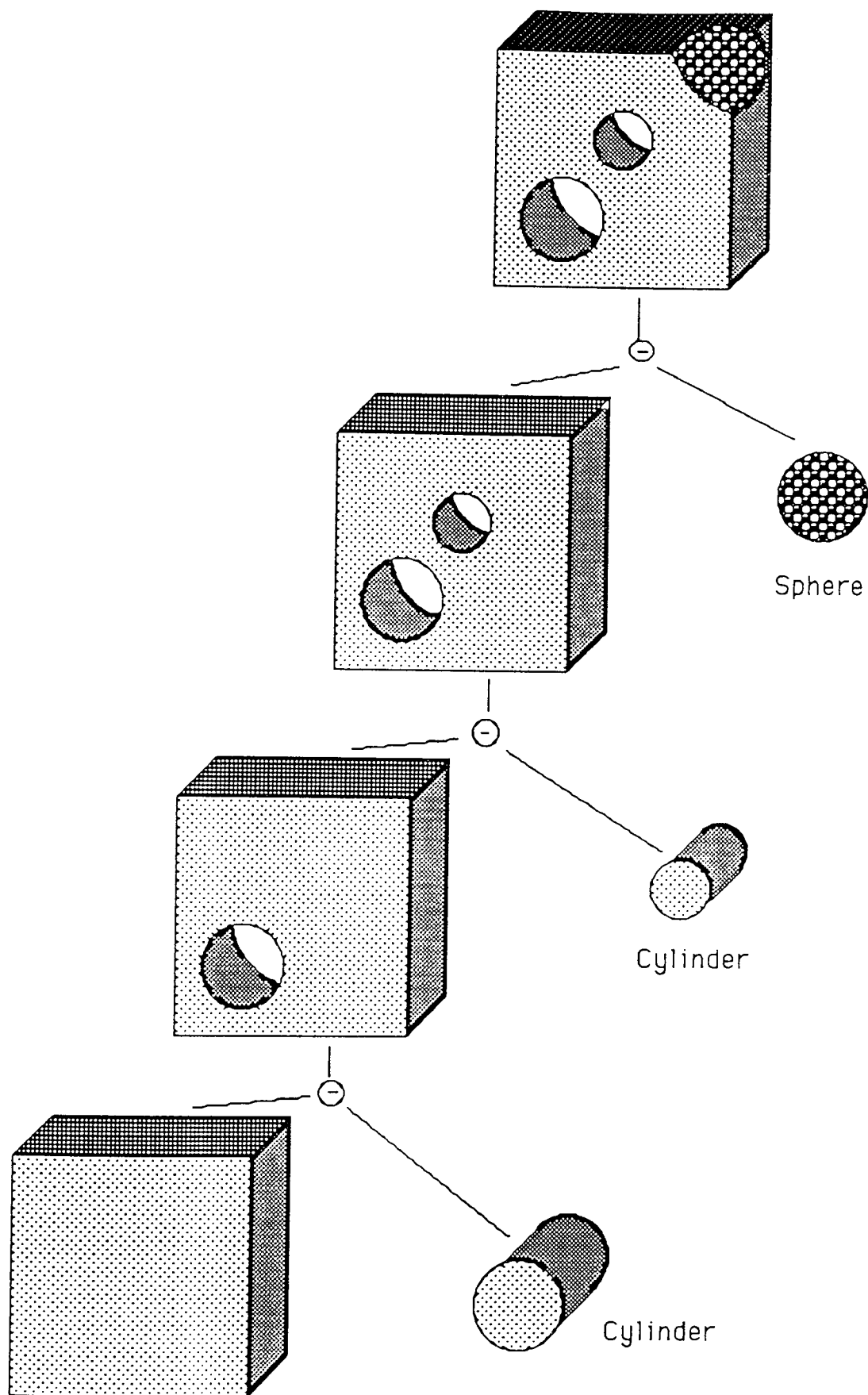


Figure 2.14 The CSG tree

The main disadvantages of boundary representation are its computation time and large storage requirement. Figure 2.15 is the boundary representation of the solid object shown in figure 2.11. The description of the object in term of its boundary faces, edges and vertices is detailed in figure 2.16.

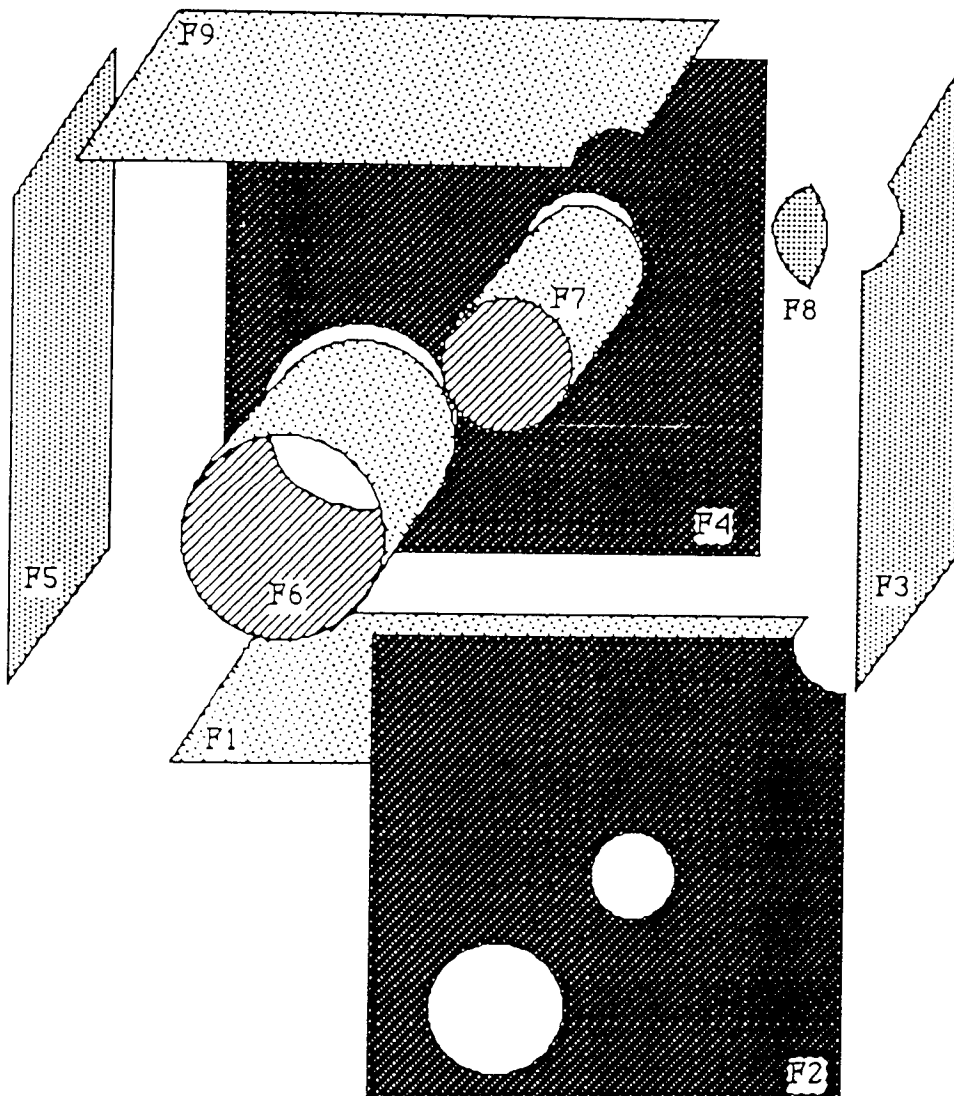


Figure 2.15 Boundary representation of the solid of figure 2.11

2.4.2.1 Euler Operation

There is a well-known relationship between the number of vertices (V), edges (E), and faces (F) of a simple polyhedron, called Euler's formula for polyhedra:

$$V - E + F = 2$$

An Euler object always satisfies Euler's formula. The processes that add or delete faces, edges, and vertices to create a new Euler object are called Euler operators. These operations provide a rational method for constructing solid, polyhedra-like objects and ensure that they are topologically valid.

Since Euler's formula is not restricted to plane-faced polyhedra but also applies to any surface on which a proper net can be constructed, the formula becomes a tool for checking validity of any solid whose surfaces can be expressed as a net of patches, curve segments, and vertices. To use Euler's formula, the following conditions must also be satisfied:

- 1) All faces are simply connected, i.e. with no holes in them, and bounded by a single ring of edges.
- 2) The solid object is simply connected and has no holes through it.
- 3) Each edge adjoins exactly two faces and is terminated by a vertex at each end.
- 4) At least three edges meet at each vertex.

In general, the Euler's formula is modified as follows:

$$V - E + F - H + 2 P = 2 B$$

where :

H = number of holes in faces

P = number of passages (holes through the entire object)

B = number of separate, disjoint objects

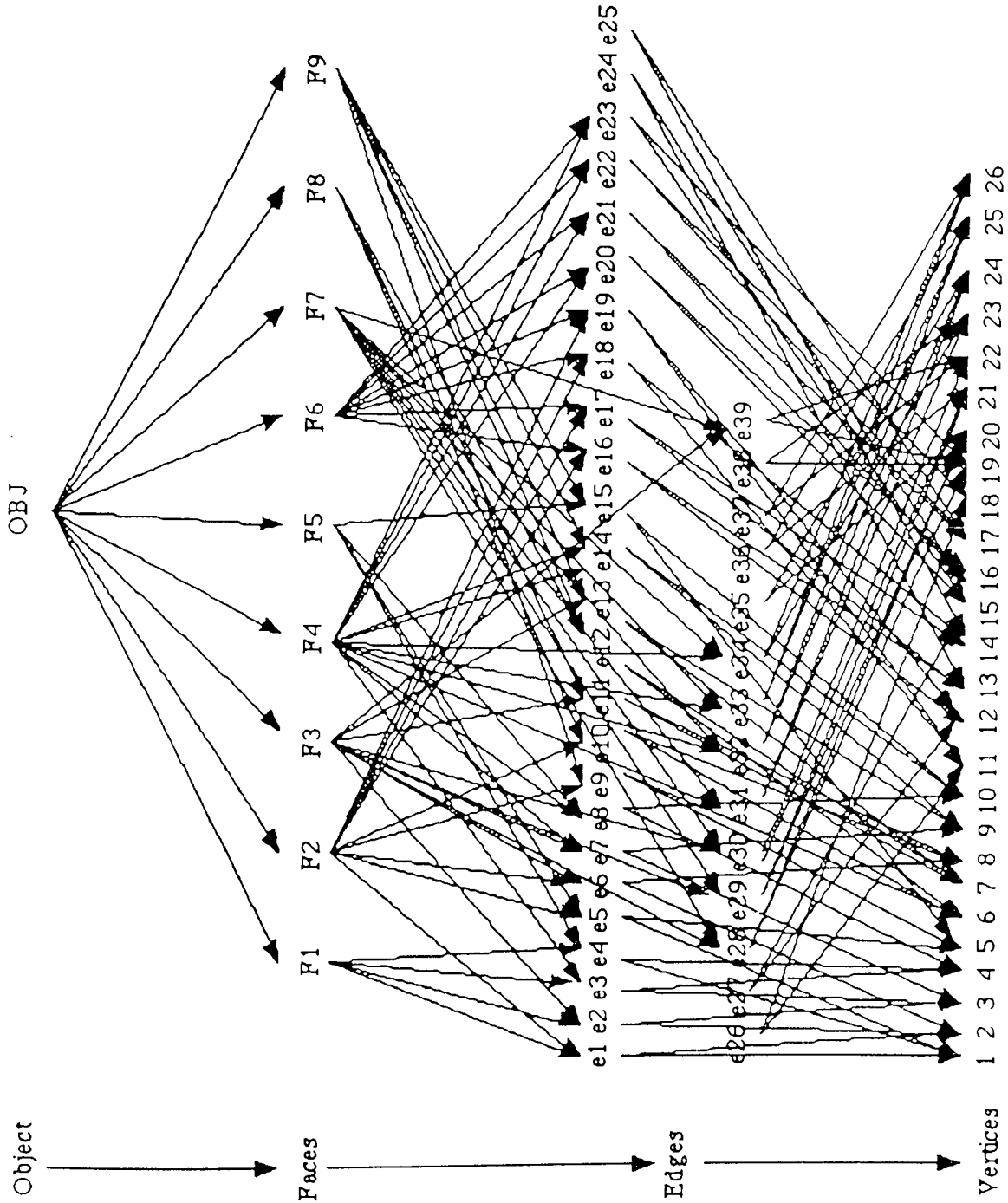


Figure 2.16 Boundary description (topology)

2.4.2.2 Winged-edge data structure

This method was introduced by Baumgart [1975] and it is an efficient way of storing the boundary representation information for polyhedron objects. In this data structure, as illustrated in figure 2.17, every edge is assigned a direction, and from each edge there are pointers to :

- the two faces intersecting at that edge; these are called F_{right} and F_{left} as observed from the outside of the object.
- the next edge in the sequence of edges bounding F_{right} in clockwise order.
- the next edge in the sequence of edges bounding F_{right} in counter clockwise order.
- the two vertices bounding the edge.
- the next edge in the sequence of edges bounding F_{left} in clockwise order.
- the next edge in the sequence of edges bounding F_{left} in counter clockwise order.

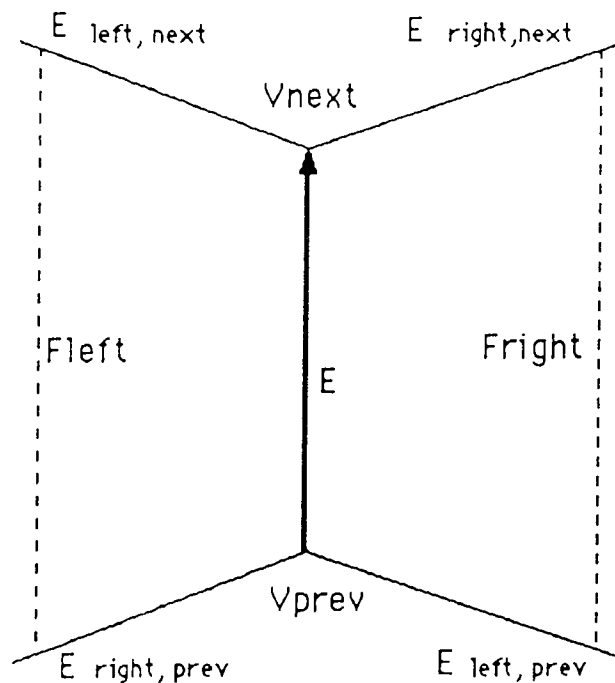


Figure 2.17 Winged-edge representation

2.4.3 Sweep Representation

Sweep representation for modelling is simple to understand and is based on the notion of moving a point, curve or surface along some path. For solid modelling, two elements are required. One is an element to be moved and the other is a trajectory to move it along. The element can be a curve, surface or solid, and the trajectory is an analytically definable path. Two principal types of trajectories are depicted: translation and rotation (fig. 2.18 and fig. 2.19). In all cases, the shape of the object being swept along does not change. Sweep representation is practical and efficient for modelling constant cross-section mechanical parts.

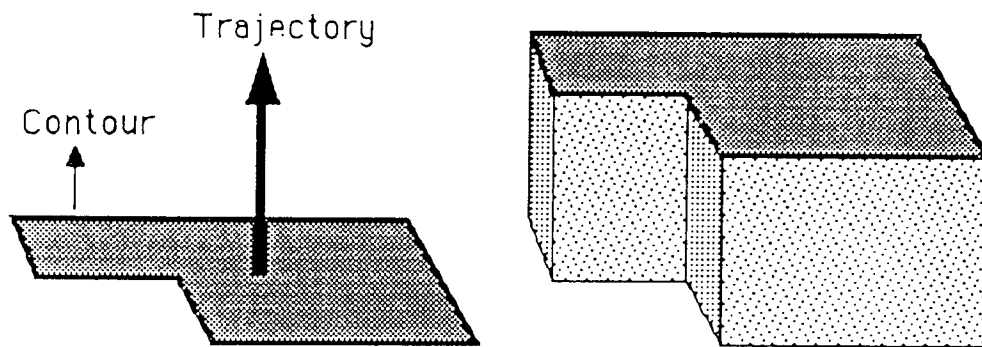


Figure 2.18 Translational sweep

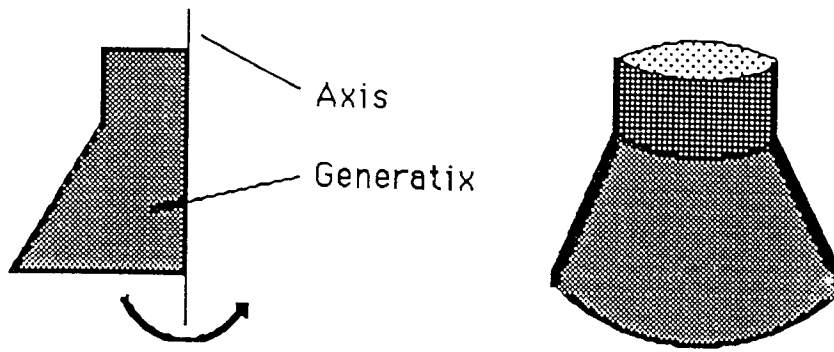


Figure 2.19 Rotational sweep

Beyond these two simple methods, the scale of the object to be swept can be changed across the sweep direction. This is called "variable cross section". For example, if a circle in plane X-Y (figure 2.20a) is swept across the Z-axis with its scale varying according to the line shown in figure 2.20b, then it generates a cone (figure 2.20c). This is a linear form modulation of a circle. In general, the representation of a boundary in X-Y plane can be defined by:

$$H(u) = \begin{bmatrix} h_1(u) \\ h_2(u) \end{bmatrix} \quad \text{where } u_1 \leq u \leq u_2$$

and the modularity function as:

$$M(t) = \begin{bmatrix} m_1(t) \\ m_2(t) \end{bmatrix} \quad \text{where } t_1 \leq t \leq t_2$$

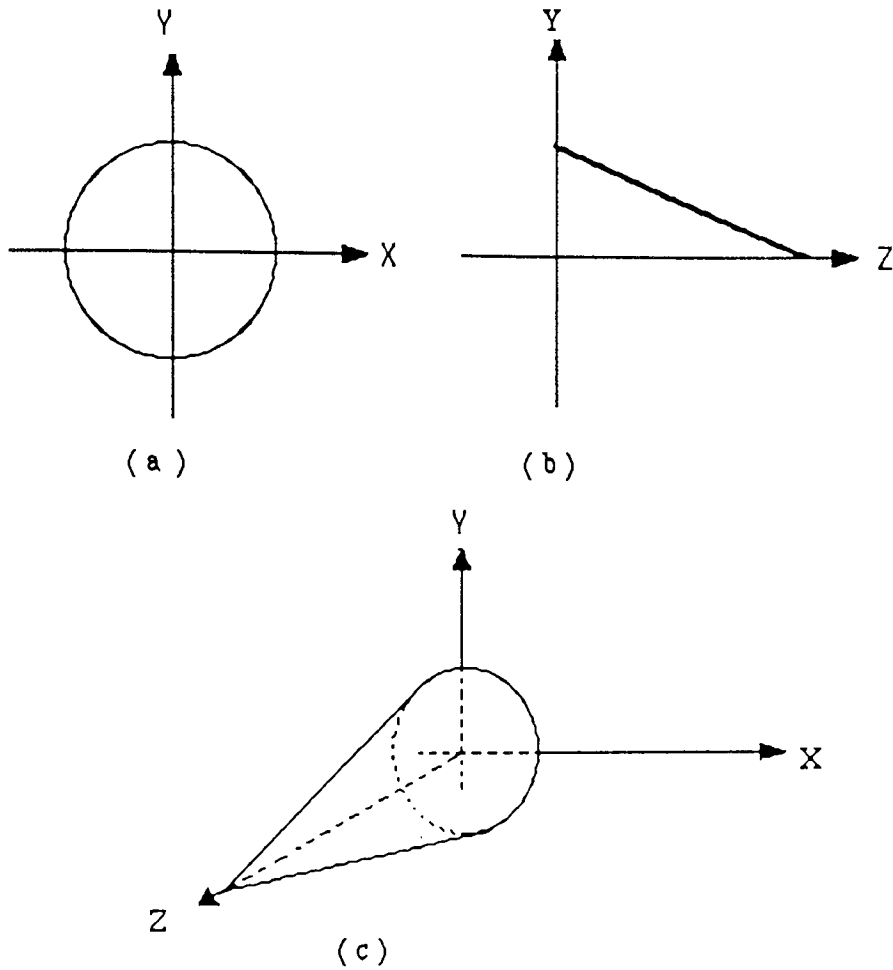


Figure 2.20 Variable cross section sweep

Then, the spherical product of these two gives the solid object:

$$M(t) \times H(u) = \begin{bmatrix} m_1(t)h_1(u) \\ m_1(t)h_2(u) \\ m_2(t) \end{bmatrix} \quad \text{where } u_1 \leq u \leq u_2, t_1 \leq t \leq t_2$$

It is obvious that $m_1(t)$ effectively scales the x and y values for the position on the surface and $m_2(t)$ gives the z -value. A boundary function called the super-ellipse may be chosen :

$$H(\theta) = \begin{bmatrix} a \cos^u \theta \\ b \sin^u \theta \end{bmatrix}$$

Depending on the value of u , this function generates a range of useful cross-sections for engineering purposes, as shown in figure 2.21.

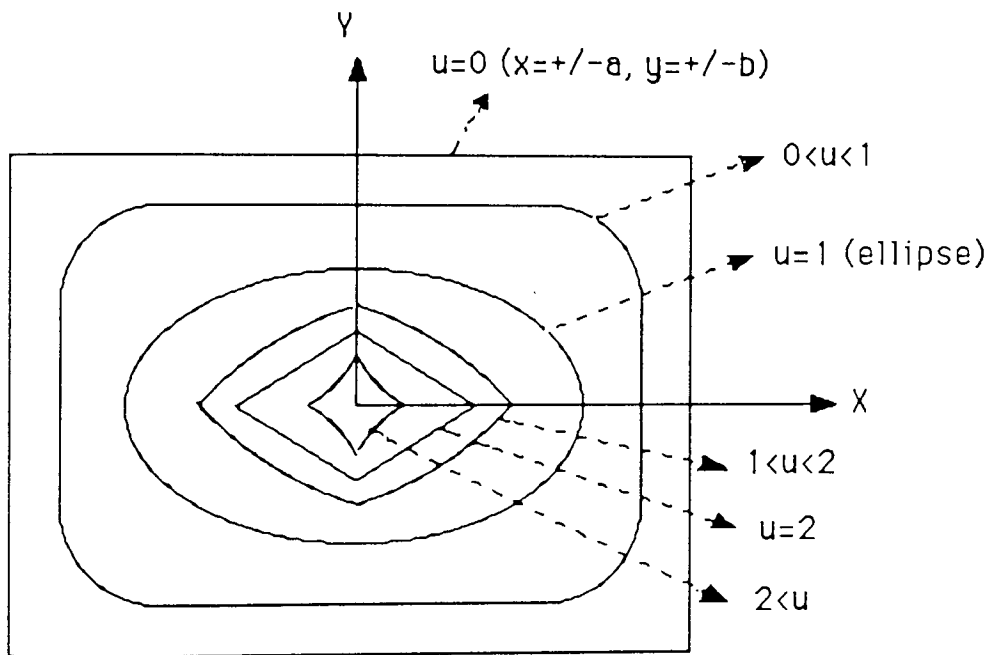


Figure 2.21 The effect of varying value of u

2.4.4 Cell Decomposition

This method decomposes an object to smaller parts. Cell decomposition is regularly used in structural analysis, and is the basis for finite element modelling.

2.4.4.1 Spatial occupancy enumeration

Spatial enumeration is a special case of cell decomposition. Here cells are cuboid in shape and located in a fixed spatial grid. As the size of the cube decreases, this method approaches the representation of a solid body as a set of contiguous points in space. The maximum resolution of the model is determined by the cell size. Easy access to a given point and assurance of spatial uniqueness are advantages of this method. Of course, there are also many disadvantages: there is no explicit relationship between the parts of an object, and the technique demands significant computer memory. *Octree* encoding suggests a way of enumerating spatial occupancy more efficiently (fig. 2.22). The root node of the Octree represents the entire object. Leaf nodes represent regions that require no further subdivision.

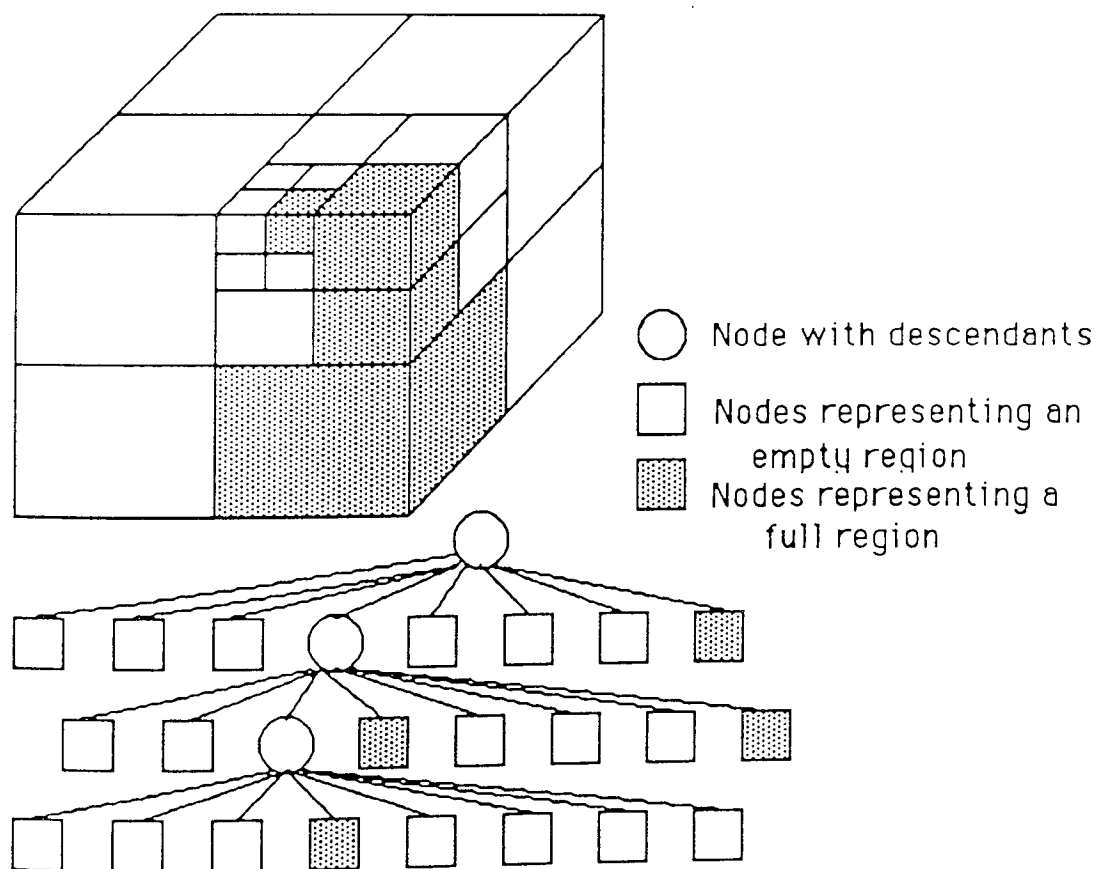


Figure 2.22 Octree representation

Chapter Three

THEORY OF ENGINEERING DRAWING

Chapter Three

THEORY OF ENGINEERING DRAWING

3.1 Introduction

Engineers must engage in a range of activities such as analysis, design, drawing and manufacture, which are concerned with communicating ideas about artefacts. Engineering drawing plays a crucial role in this communication process. The graphical form is ideal for quickly conveying complex ideas in an *unambiguous* manner. Engineering drawing practice has been standardised in an attempt to set down rules and procedures for the graphical form. There are standard conventions for representing objects by many 2D pictures (views). The ability to interpret the various views from engineering drawings and to form them into mental visualizations of the 3D nature of artefacts is as essential now as it ever was before the introduction of computing into engineering drawing. An understanding of the conventions and terminology employed in drawings is vital.

3.2 Orthographic Projections

The majority of drawings found in engineering practice are based upon the principles of orthographic projection. Two imaginary flat planes are placed to cross each other at right angles (orthogonally), as shown in figure 3.1. The two planes, known as the horizontal plane (HP) and vertical plane (VP), are thus placed so as to form four quadrants in space [Davies *et al.*, 1986]. The object which is to be drawn is placed

within either one of two of the four quadrants and is then viewed from the front and from above. The outlines of the shape seen by this viewing are then projected on to the two planes. Among these four quadrants only two of them are suitable for producing sensible projections for engineering drawings and they are known as First Angle and Third Angle.

Both First and Third Angle PLANS
as seen in this direction

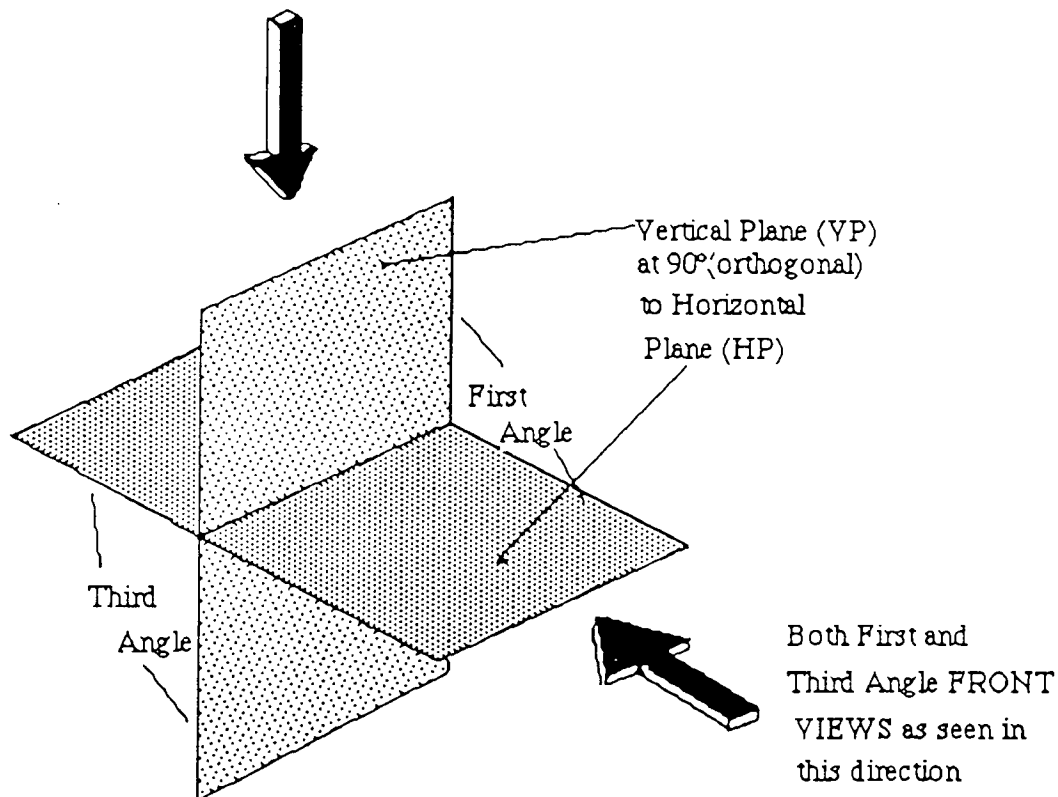


Figure 3.1 Orthographic projection

3.2.1 First Angle Orthographic Projection

The component is placed in the space within the First Angle quadrant. It is then viewed from the front, and what is seen is drawn on to the vertical plane (VP). This gives a *front view* (figure 3.2). Second viewing position from above allows projection to be drawn on to the horizontal plane (HP) to give a *plan*. A second vertical plane is placed on the right hand of the component and a viewing position to the left of component provides the projections on to the second VP to obtain an *end view*. At the end three planes are swung into a two-dimensional position to give a 2D drawing (figure 3.3). Figure 3.5 shows the first angle orthographic views for object of figure 3.4.

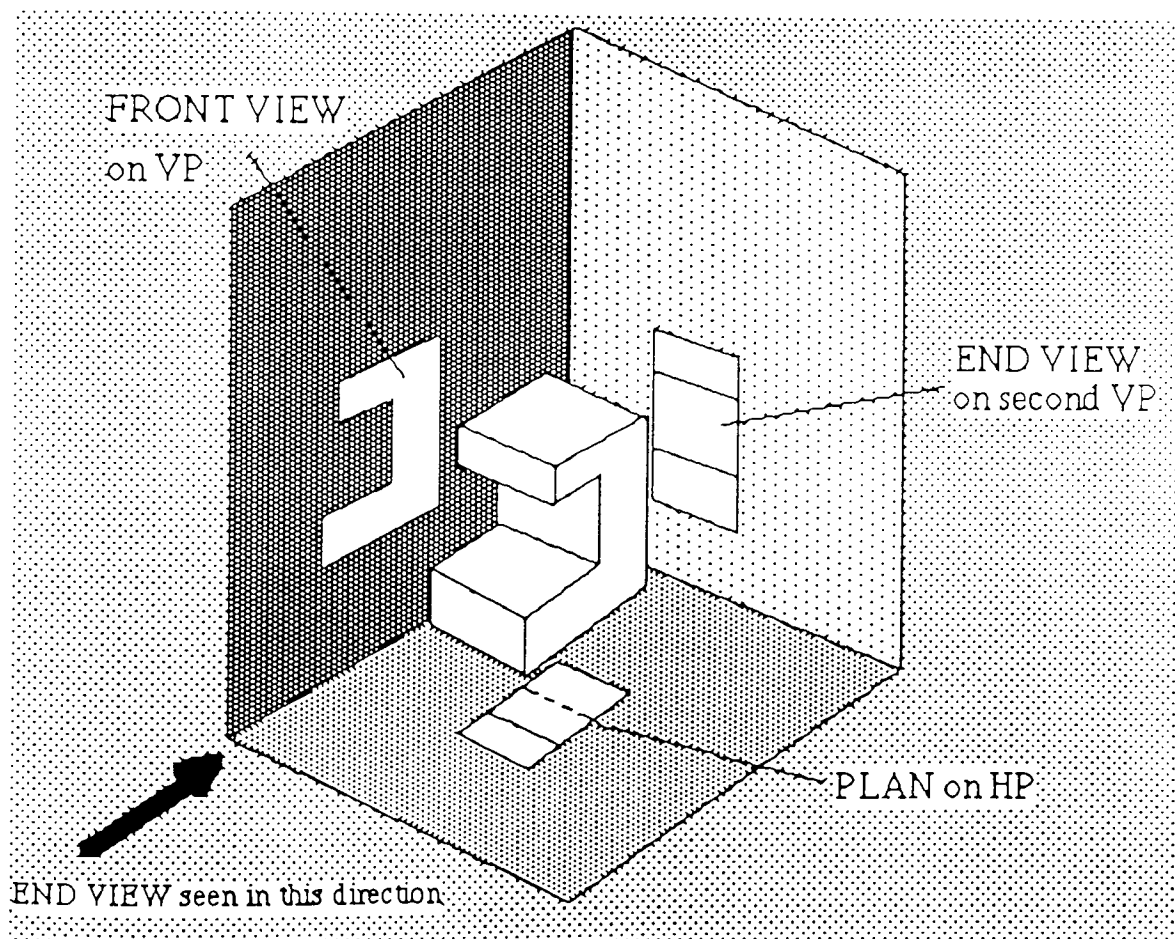


Figure 3.2 First Angle orthographic view

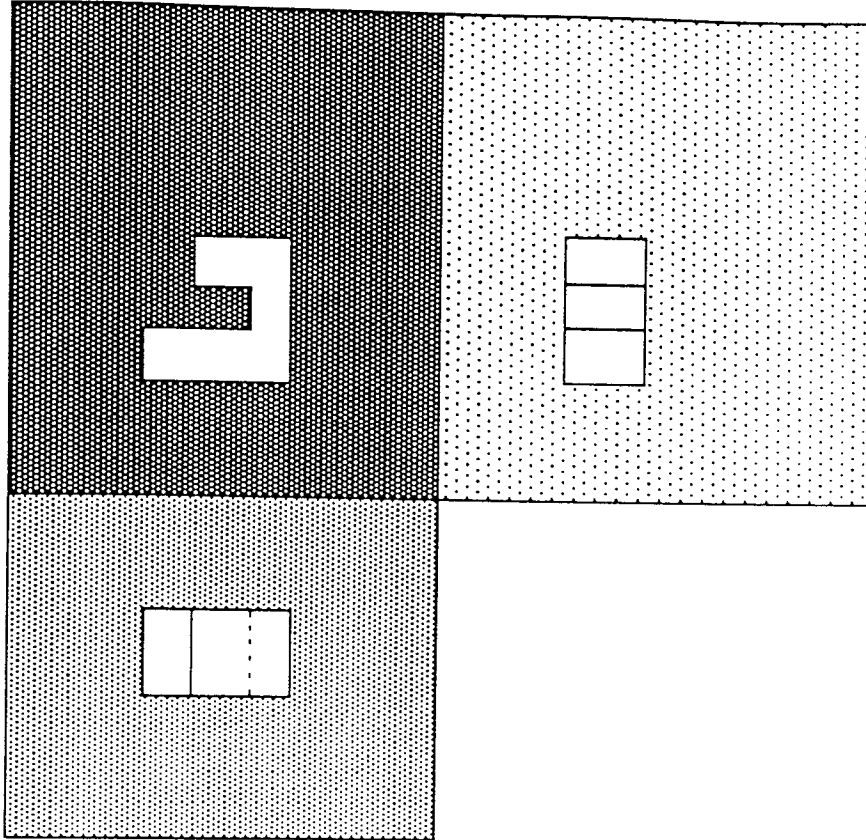


Figure 3.3 The three planes swung into a two-dimensional position

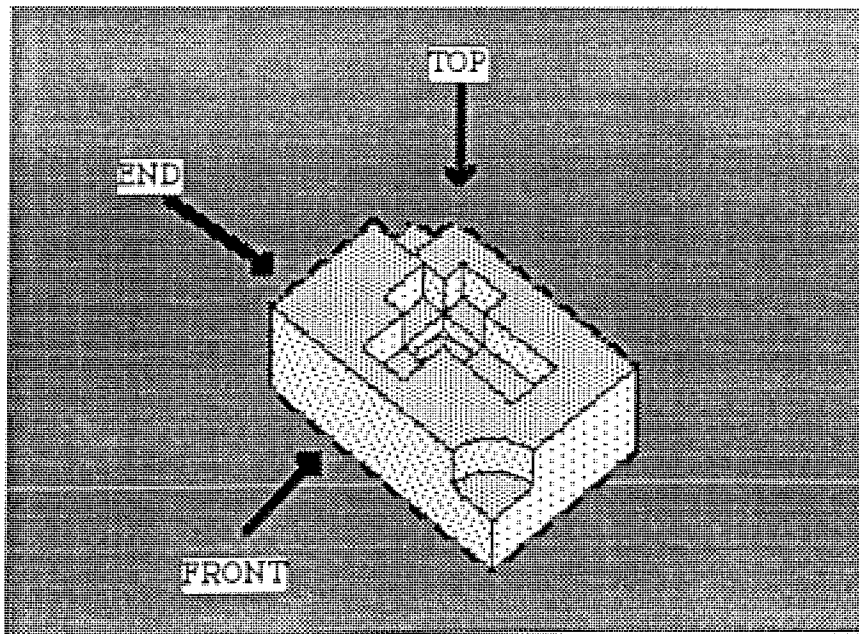


Figure 3.4 A solid object with direction of different views

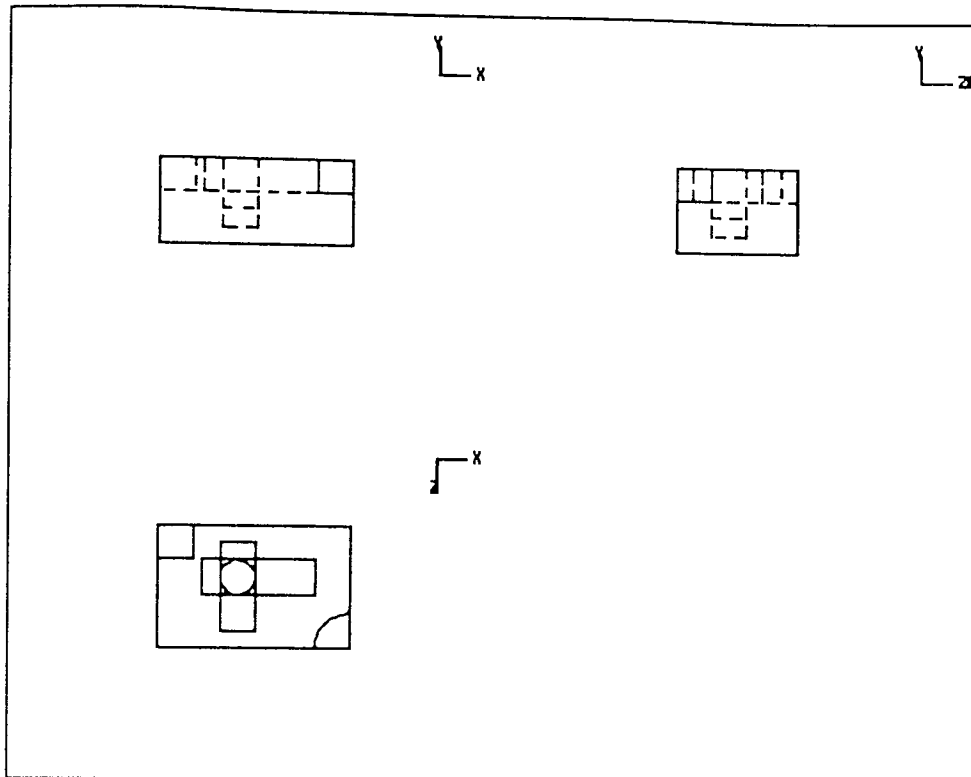


Figure 3.5 First Angle orthographic views of the object in figure 3.4

3.2.2 Third Angle Orthographic projection

This time the component is placed in the space within the Third Angle quadrant which is formed by the two orthogonally placed vertical and horizontal planes. The direction of viewing to obtain projections are the same as those employed in First Angle projection (figure 3.2). The required views are drawn on to the planes through which they can be seen. An end view drawn on a second vertical plane placed on the left of the component complete a three-view projection in Third Angle. Figure 3.6 represents the third angle orthographic views of the same object of figure 3.4.

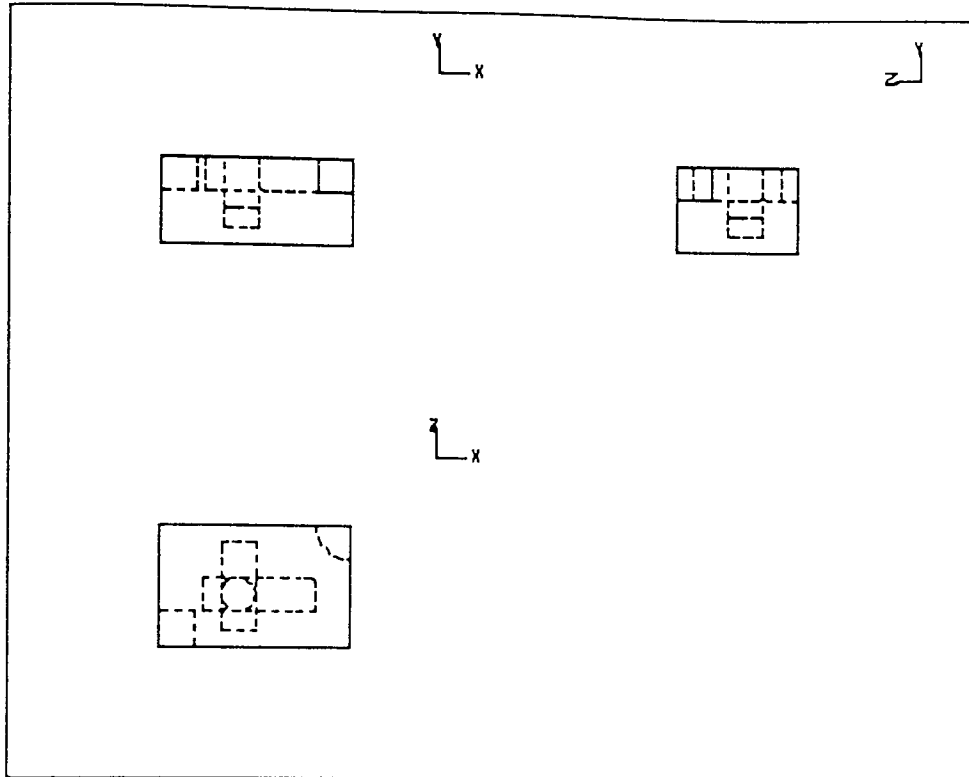


Figure 3.6 Third Angle orthographic views of last example (by: BOXER)

3.2.3 Orthographic projections of primitives and prismatic objects

Primitives are simple basic objects which are used as elements in Constructive Solid Geometry (CSG) for making complex objects. Here the base faces of primitives are assumed to be parallel with one of the main planes. In this case orthographic projections for some primitives, which are called "signatures", are shown in figure 3.7. Later these signatures are used to identify a corresponding primitive.

Prismatic objects have a constant cross-section along one of the main directions. A simple prismatic object is shown in figure 3.8.

This may be transformed into a complex prismatic object, having a multiply-connected cross-section, by drilling holes, as shown in figure 3.9.

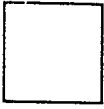

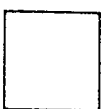
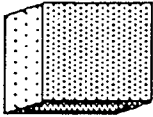



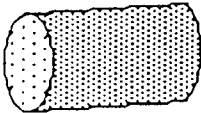
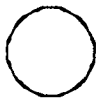


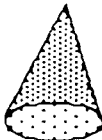



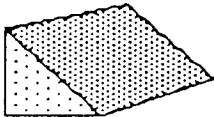



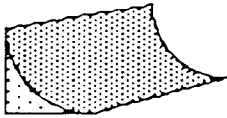



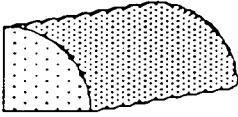
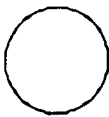
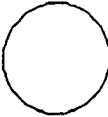
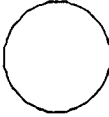
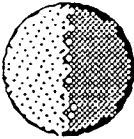
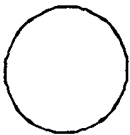



Primitive name	Orthographic views (signatures)			3D Primitive
Cuboid				
Cylinder				
Cone				
Wedge				
Fillet				
Sector				
Sphere				
Hemi-sphere				
.....

Figure 3.7 Orthographic projection of primitives

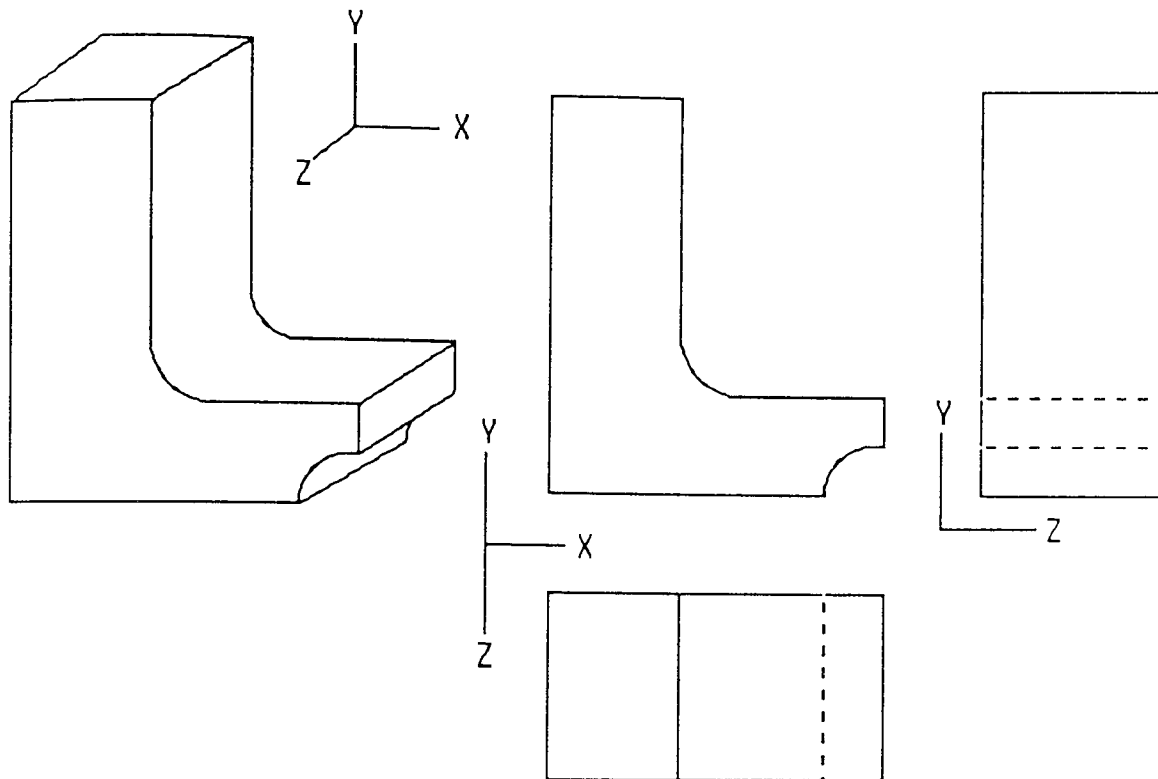


Figure 3.8 Orthographic views of a simple prismatic object

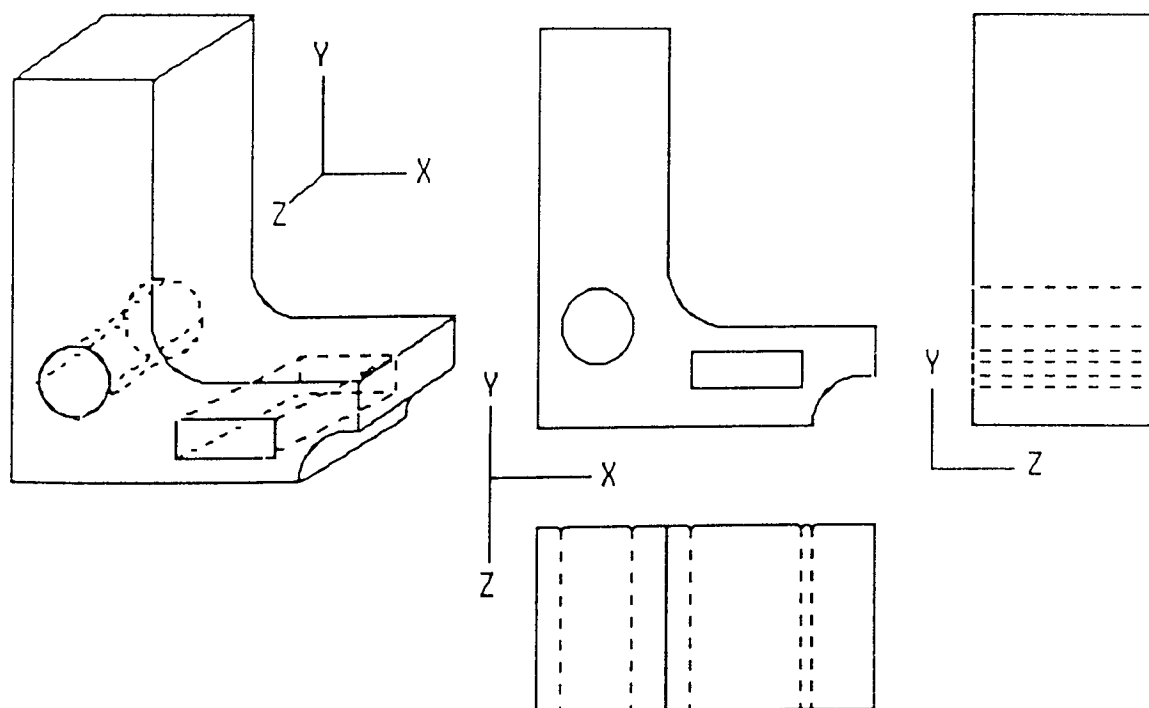


Figure 3.9 Orthographic views of a complex prismatic object

Simple prismatic objects have at least one view consisting of one closed loop only, which is the perimeter loop; complex prismatic objects have at least two or more closed disjoint loops. This special view is called the *base* view. All other loops in other views are rectangular.

3.2.4 Orthographic views of arcs and lines

Arcs in 3D space are assumed to be located on a flat surface. Therefore, the following situations may occur:

- 1) The arc's plane is parallel to one of the main planes. In this case the projection of the arc in the parallel plane is the same as the original one, and the other projections will be straight lines (figure 3.10a).
- 2) The arc's plane is orthogonal to one of the main planes. This time only the projection on the main plane orthogonal to the arc's plane will be a line and two others will be different arcs (figure 3.10b).
- 3) Neither of cases 1 and 2 : the projection of the arc in all views will be different arcs (figure 3.10c).

A similar analysis may be made for straight lines:

- 1) The line is parallel to one of the main planes, in this case the true length of the line appears on the mentioned parallel plane (figure 3.11a).
- 2) The line is orthogonal to one of the main planes, therefore the projection of the line on this plane is only a point and the true length of the line (edge) appears in two other planes (figure 3.11b).
- 3) Neither of the above cases, so projections of the line appear in all three views with lengths of less than the true one, proportional to the cosines of the angles between line

and planes (figure 3.11c).

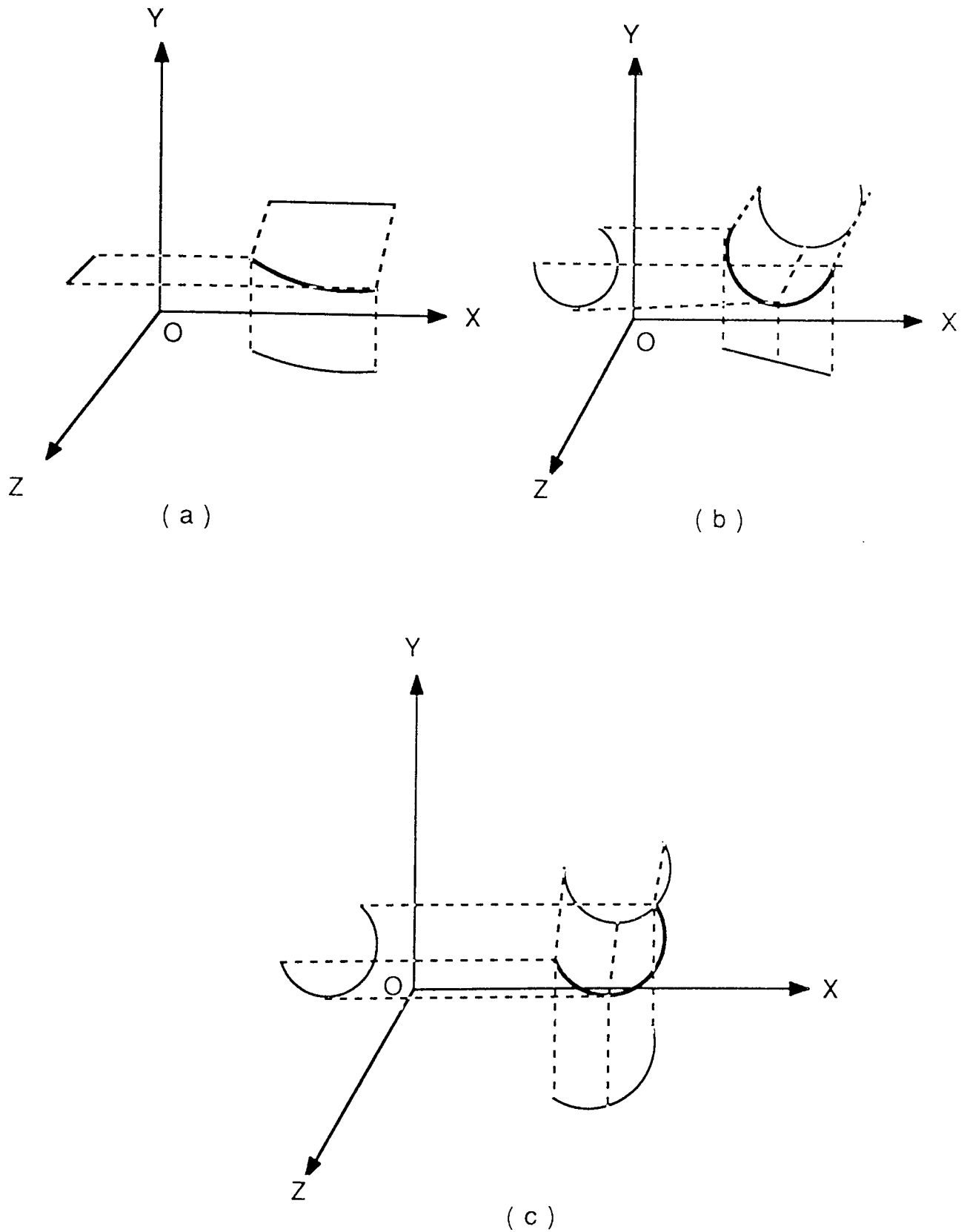


Figure 3.10 Different types of arc's orthographic views

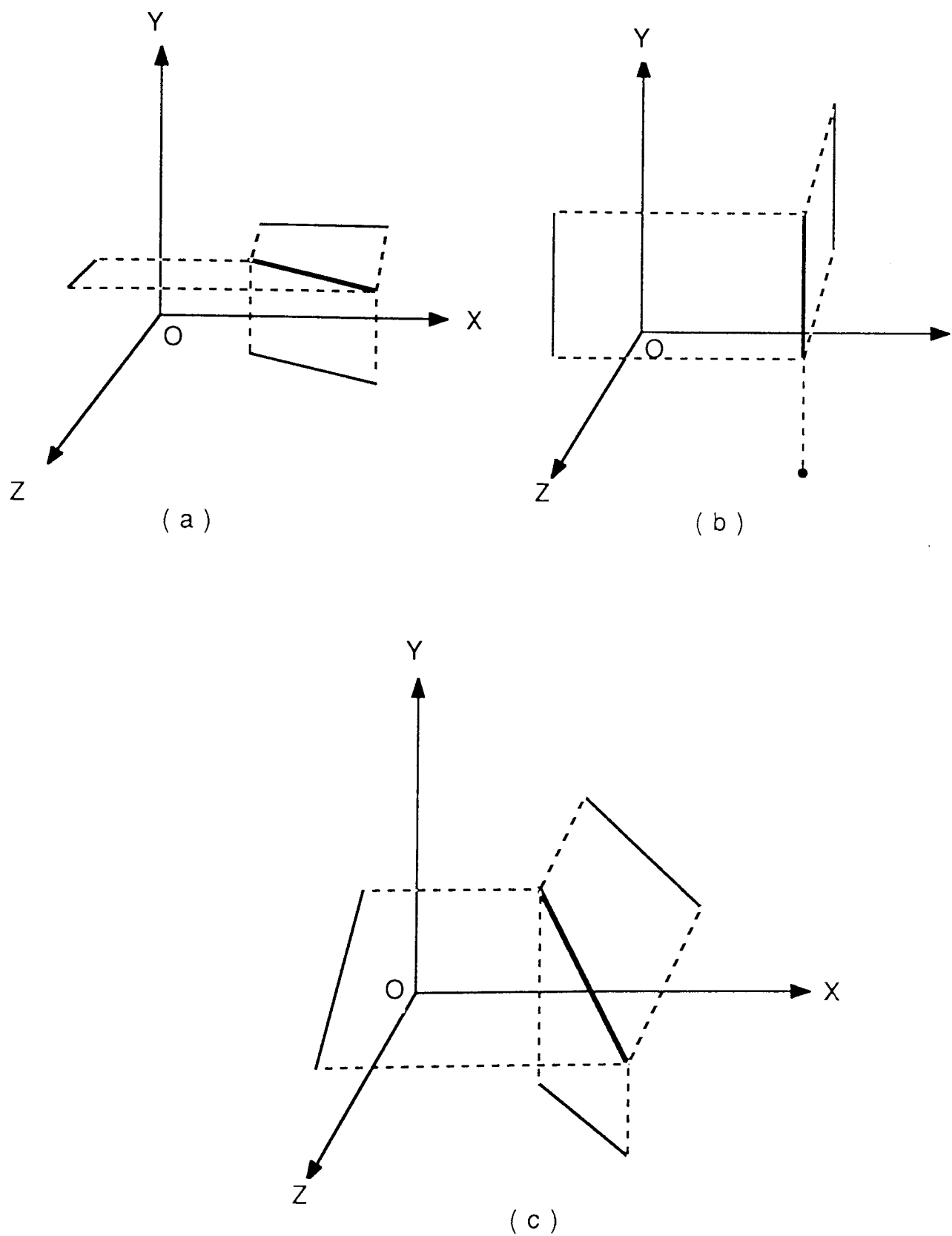


Figure 3.11 Different cases of line's orthographic views

3.2.5 Visibility

An essential step in drawing an orthographic view is the correct determination of the visibility of the lines that make up the view. The outline of a view will always be visible but the lines within the outline may be visible or hidden, depending on the relative position of those lines with respect to the line of sight.

3.3 Engineering Drawing Conventions

3.3.1 Line styles

Drawings are made of a variety of lines and in any orthographic view, different kinds of line drawing may be seen. Lines vary in thickness and shape. Line symbols help people in reading the drawings and interpreting what it is meant. Each type expresses a kind of property according to some standard conventions. Accepted standard line symbols are published by the American National Standards Association in publication ANSI Y14.2M_1979. A few of them are explained below:

Continuous thick line to show outlines.

Thick chain line is drawn adjacent to an outline and indicates a surface requiring special treatment.

Thick dashed line is usually used in UK designed engineering drawing, to show hidden details.

Continuous thin line represents projection line and, if ending with arrows, is used to show dimensions.

Broken thin line is used to indicate hidden outline or hidden edge.

In the project, after all refinements, only two types of lines, are accepted. One is a continuous line, representing visible edges, and the other is a dashed line, representing

hidden (invisible) edges.

3.3.2 Fictitious and unspecified lines

In engineering drawings, to simplify the representation of objects and to reduce the number of lines drawn in the document, some lines are conventionally eliminated from the drawing. The interpretation of this missing information is obvious to a trained human observer and is the result of their experience. Such 'experience' must somehow be communicated to the software that is to process a drawing of this nature. For example, the absence of a centre line implies lack of symmetry and such implication must be communicated to intelligent software. Some simple computer programs, however, are not designed intelligently, and this information must be specified. In this project it is possible to determine these lines from other information but, since it is not the main purpose of the project, and in order to save time, they are assumed to be specified at the input. Examples of such fictitious lines are the projections of tangency and silhouette edges as shown in figure 3.12.

3.3.3 Symbolic representation

In any engineering drawing, apart from the lines and arcs which show the picture of 3D edges (intersection of surfaces), there are many other characters and symbols. All of these symbols have a special meaning according to some predefined standard conventions. In many cases it is not possible actually to draw a specific item as part of a drawing. Therefore, symbols are used to represent complex features that are difficult to show by drawing literally. They speed up the drafting process and simplify the drawing. For example, an electrical drawing for a power plant cannot detail each switch or motor as it really appears. Instead a *symbolic* drawing that represents a more

complex item is used. Symbols for various industries are standardized.

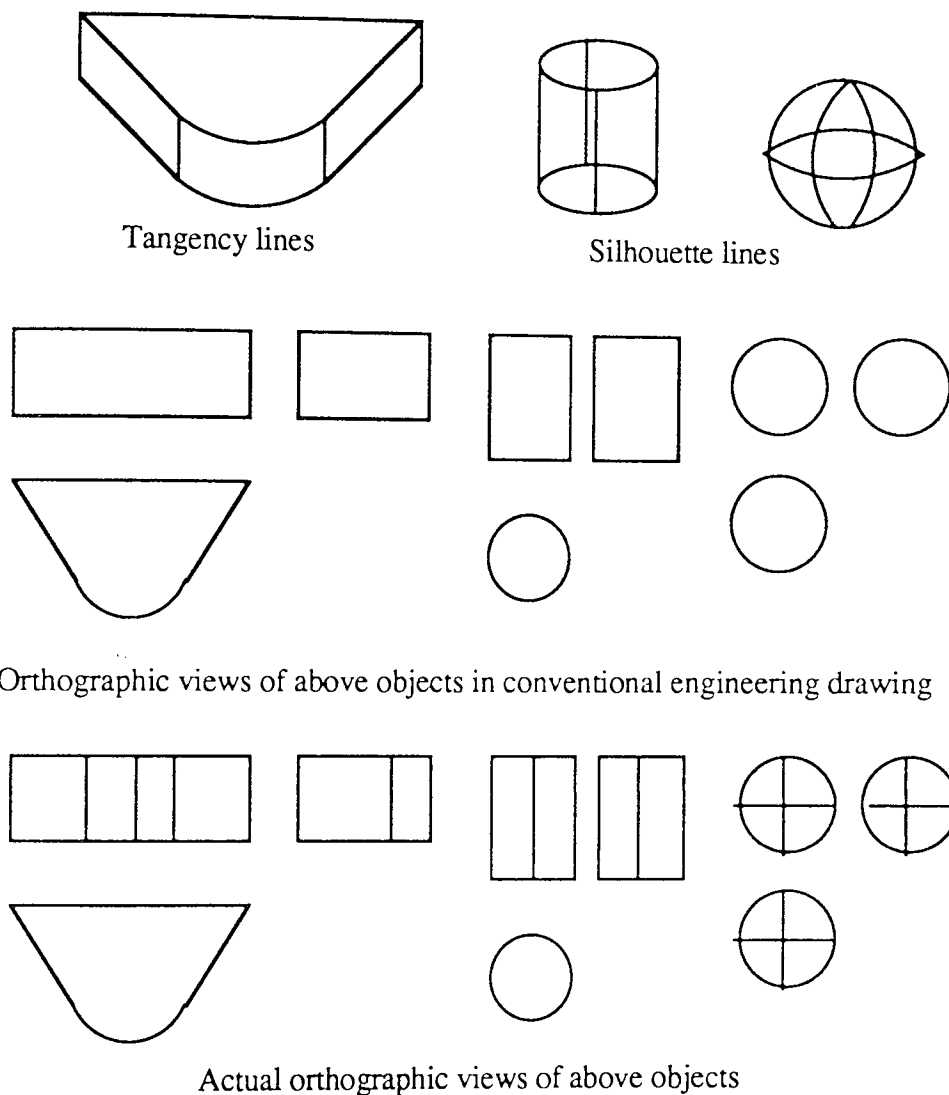


Figure 3.12 Examples of fictitious lines in engineering drawing

3.4 Views Needed

As many as six views on horizontal and vertical planes are possible within the system of orthographic projection. Even further views are possible taken with imaginary planes at angles other than 90° to each other. However, a good rule is that one should draw as few views as possible. Thus a flat component made from thin sheet material may be clearly and fully described in a single view (a front view, assuming the third dimension

is a constant value). Other items may require two views (a front view and an end view, or a front view and a plan). More complicated objects may require three, four or more views to describe completely all the necessary features. Sometimes a separate view, out of projection with the main views, may suitably describe features not clearly shown by the main views. This view is called a *special view*. In some cases *sectional views* (or *sections*) are used to define clearly internal shapes and forms of object.

3.5 Pictorial Drawing

Most of the drawings used in engineering are based on the principle of orthographic projection and will thus largely consist of views of the solid object projected on to a flat surface. There are occasions when it will be found necessary to produce pictorial 3D drawings to explain, or to enable easier comprehension of, orthographic projections. Pictorial drawings are also of value when an explanation of design features, design details and functions are required. There are three methods in common use for producing pictorial drawings: *isometric drawing*, *oblique cabinet drawing* and *simple perspective drawing*.

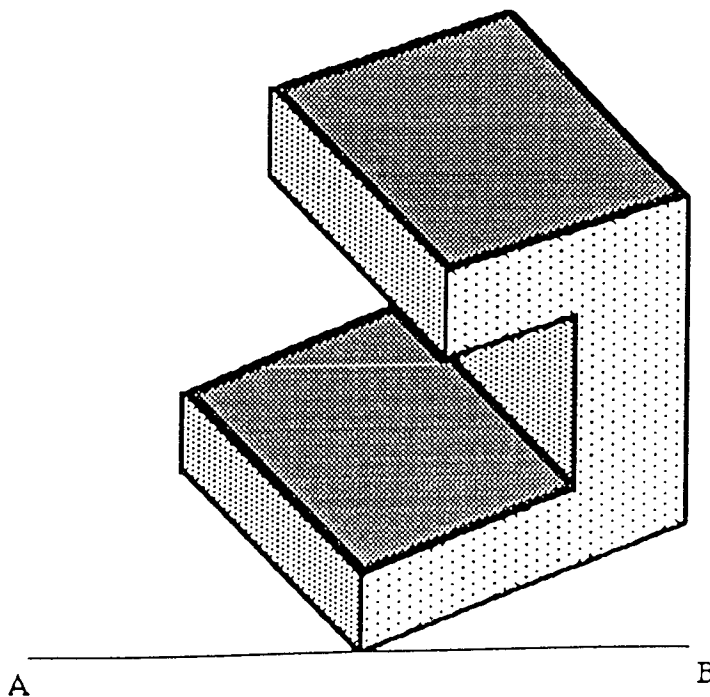


Figure 3.13 A simple Isometric view

3.5.1 Isometric views

In isometric drawing all sloping lines are drawn at 30° or 60° to a base line AB and all verticals are drawn at 90° to AB. The measurements of length (L), width (W) and height (H) are taken directly along the sloping and vertical lines, with no scaling (figure 3.13).

3.5.2 Oblique Drawing

Oblique drawing is a form of pictorial drawing, in which front facing surfaces are drawn as true views of the front face of the object and receding edges are drawn parallel to each other at angles of 30° , 45° or 60° . It is common practice when making oblique drawing to use the same scale along receding edges as for the front facing surfaces.

Cabinet drawing is a particular form of oblique drawing in which receding edges are drawn at angle of 45° to the front surface and measurements along the 45° are taken at half size of the drawing scale as shown in figure 3.14.

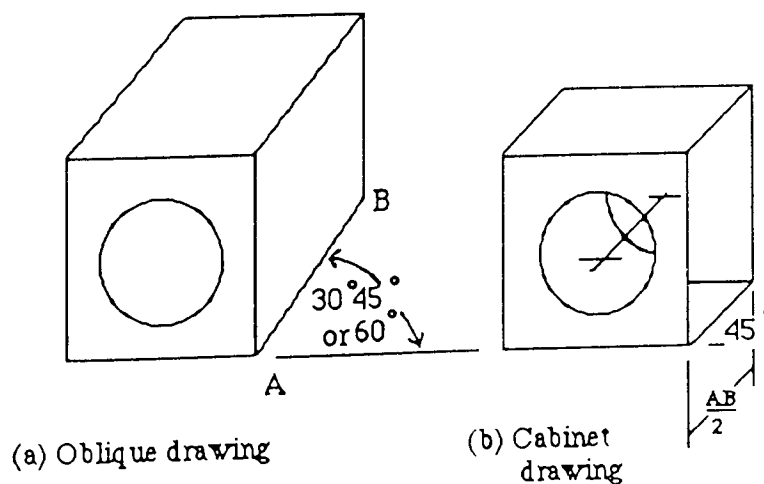


Figure 3.14 Oblique and Oblique cabinet drawings

3.5.3 Perspective Drawing

This method is similar to the way of observing objects by the eyes. It means that lines receding from the observer appear to converge towards a point in the distance. Technical perspective drawing of this type is assumed to be based upon the use of one, two or three vanishing points towards which sets of lines converge. They are referred to as one-point, two-point or three-point perspective (figures 3.15, 3.16 and 3.17).

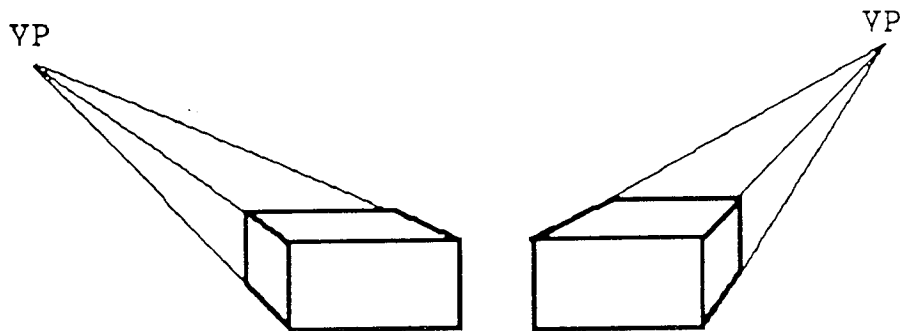


Figure 3.15 Two examples of one-point perspective

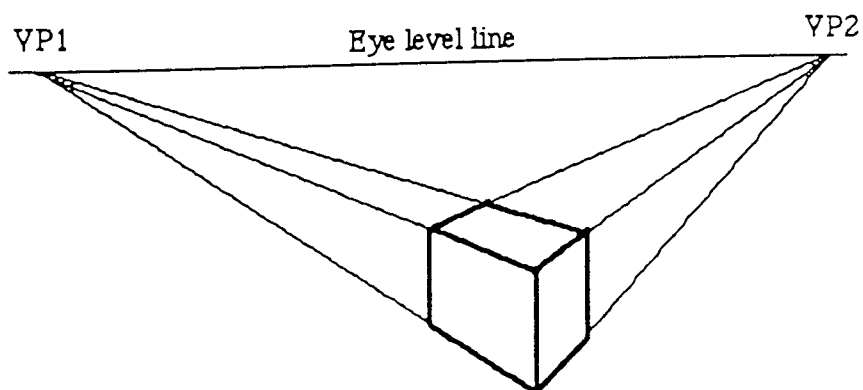


Figure 3.16 Two-point perspective

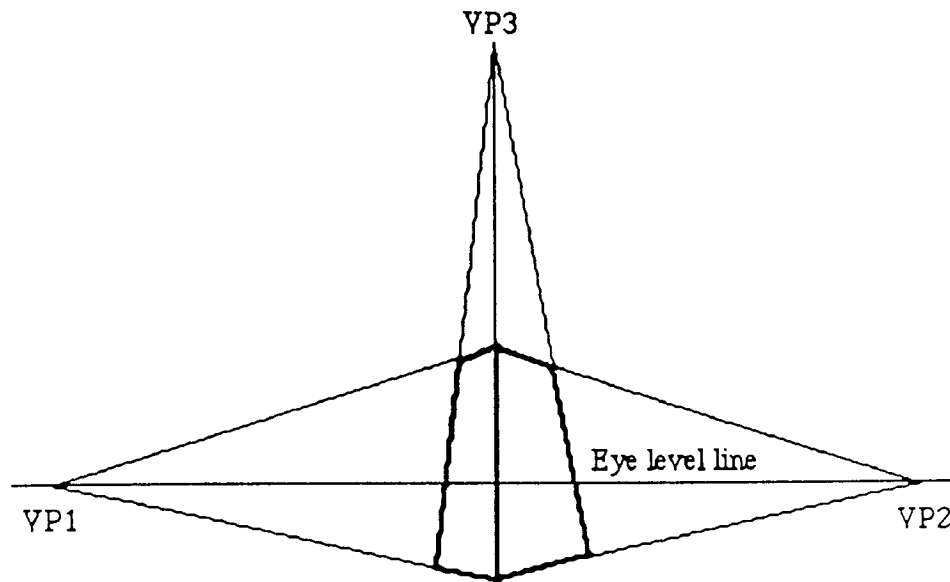


Figure 3.17 Three-point perspective

3.6 Dimensioning and Tolerances

Dimensioning plays an important role in engineering drawing. Good dimensioning clearly defines the size, shape and location of the parts of a component described in an engineering drawing. Features such as lengths, diameters, angles and positions should have been completely and accurately described by the dimensions contained in a drawing.

In manufacturing process, all parts are subject to size and form variability. To account for this variability, tolerances should be specified as part of the design process and included in the database. Current solid modellers contain complete unambiguous representations of the nominal or ideal part, but lack tolerance facilities. Hence some design and production activities that need tolerancing information such as fully automated manufacturing and assembly planning, and quality control, cannot be supported by these kind of modellers. Thus, tolerances may be added to dimensions of

length and to other dimensions such as radii, diameters and angles, and a dimension without tolerance cannot be considered as a practical possibility. Tolerances define upper and lower limits of acceptable size for any such particular dimensions. They do not define the possible variation of features such as the geometrical shape and form of a dimensioned object.

At present, in the project, the initial tolerance for the dimension of length is set to a constant value, and whenever it is necessary, it is changed proportional to the maximum length of the part. Optionally it can be changed in such a way to be user defined parameter. The tolerance value for angle is selected by the program as it is required, depending on the type of process involved.

Chapter Four

2D TO 3D CONVERSION

Chapter Four

2D TO 3D CONVERSION

4.1 Introduction

The process of reconstructing a solid object from its orthographic views is highly complex. Briefly put, only expert persons are able to interpret a 2D engineering drawing and imagine its 3D representation. This understanding is based on having knowledge of engineering drawings, the conventions used in producing orthographic views, previous experience, and so on.

To be able to automate these processes and interpret the solid object from these pictures by the computer, one must give all the information concerning this area to the computer. Of course, to make any 2D to 3D conversion process as practical as possible demands significant computer power (e.g. faster processors, more memories, higher storage capacities), and that means greater costs. But, as mentioned earlier, as the technology improves and the cost of hardware is reduced, more research effort is motivated, and better and more efficient algorithms are developed.

In the next section, some of the more significant work that has been done on 3D reconstruction is reviewed.

4.2 Review of Published Works

In addition to fundamental work on geometric modelling, some people have tried to use computer power in transferring two-dimensional orthographic views into three-dimensional objects. In this regard Sutherland [1974] made a multi-pen tablet system for inputting 3D data from their 2D views directly to the computer. Thornton [1978] also used a digitizer tablet to interactively locate points within 2D projections of a 3D form to input graphically. His program was able to produce a wireframe model with only straight lines. Idesawa [1973,1975] uses labeling techniques in finding edges and faces in three views. He presents a formulation for the process of generating solid figures from 2D drawings. This system works for polyhedral cases. He points out a "Ghost Figure Problem" of a wired figure obtained during the procedure of the system and presents a method to overcome it. He also proposes a method to solve a hidden line problem. Woo and Hammer [1977] present another method which uses simple matching of features followed by a tree search guided by constraints on physically realizable objects. Only straight lines and circular arcs are used; these may both be dashed.

Some other people use heuristic functions to reject ghost edges and faces produced during wireframe generation. These functions are mainly based on the relations between 3D nodes, edges and faces. For example, any node having less than three edges connected to it is interpreted as a ghost node and therefore edges connected to it may also be regarded as ghost edges. Other heuristic functions are check for satisfaction of Moebius's and/or Euler's rules. Lafue [1976] tries to reduce ambiguity problems heuristically. The general strategy is bottom-up identification of objects and aggregation of them into vertices, then faces and finally polyhedra. The program does not contain any high level geometrical concepts. It limits the drawing constraints imposed on the user. In blocked cases it asks for the user's help. Heuristic approaches are used by Preiss [1981,1983] to find the likeliest solution among alternatives and

they are applied to plane-faced bodies. Remi Lequette [1988] extends the use of heuristic methods for cylindrical, conical or toroidal surfaces as well as planar surfaces. The method used here first builds an intermediate three-dimensional wireframe, then a heuristic is used to find edges between tangent surfaces that are usually not drawn. Faces are then build from this wireframe and the algorithm sorts them to construct a solid. Aldefeld [1983,1984] considers mechanical parts consisting of several elementary prismatic objects. These objects are assumed to have a base parallel to one of the coordinate planes. The interpretation process is done under user guidance.

All of the works mentioned here are based on wireframe modelling. Therefore, they have many limitations in practical use. For example, they cannot provide the volumetric information required for manufacturing, assembly and design analysis processes.

Wesley and Markowsky [1980,1981] used algebraic topology concepts and definitions of geometric entities to obtain volumetric descriptions in terms of solid materials and empty spaces. Markowsky and Wesley [1980] described the topology of surfaces and edges for objects in terms of their wireframe; in their later work [Wesley & Markowsky, 1981] these features are described in terms of their projections. These algorithms were restricted to objects with straight line edges and planar surfaces. This concept is developed by Sakurai and Gossard [1983] to extend the interpretation process to objects like cylinders, cones, spheres and toroids. Another algorithm is presented by Kaining, Tang and Sun [1985]. It makes some improvements to the Wesley-Markowsky algorithm, which is a typical hierarchical reconstruction algorithm limited to polyhedral objects, and extracts the idea of pattern recognition expressed in Aldefeld's algorithm. This algorithm successfully rejects pathological cases and finds all the solutions with the same set of orthographic views. Later Wesley and Markowsky [1984] survey and discuss the various attempted solutions in terms of the approaches used in conversion of 2D to 3D solids and their problems. They then explore the degree of automation for one polyhedral algorithm.

In using the principles of Constructive Solid Geometry the early work done by Braid [1973] should be noted. He uses only cube and cylinder primitives and builds up the shape of 2¹₂D or 3D mechanical components by adding or subtracting volumes. Other interesting work is based on interactive interpretation process described by Ho Bin [1986]. In the following sections some of these works will be reviewed in more detail. Also, another separate section is devoted to describing the more recent work presented by Kargas, Cooley and Richards [1988], which forms the base for the development of current project. This approach tries to convert orthographic views directly into CSG form.

4.2.1 A System to Generate a Solid Figure from Three Views

The work reported on such a system by Idesawa [1973], is one of the earliest practical projects done in the area of transforming 2D projections to 3D solids by computer. It considers only polyhedral objects. After definition of some terms and notation such as *Solid*, *Face*, *Edge line*, *Vertex* and so on, the author discusses the relation between the solid and three views. He also points to the ghost figure problem, caused by the lack of one to one correspondence between solid and drawings.

Idesawa regards the problem as inverse transformation and formulises his algorithm into the following steps:

- a) Transformation to three-dimensional points
- b) Transformation to three-dimensional lines
- c) Ghost figure elimination
- d) Construction of faces

Each step is briefly reviewed separately.

4.2.1.1 Transformation to three-dimensional points

In engineering drawing each 2D point is a representation in two coordinates of three-dimensional coordinates. For instance, a point $P_{xyz}=(x,y,z)$ in 3D is shown in X-Y, X-Z and Z-Y planes as three points $P_{xy}=(x,y)$, $P_{xz}=(x,z)$, and $P_{zy}=(z,y)$ in three views respectively. In this step the reverse function is performed and all 3D points that generate 2D vertices are produced from the three input views.

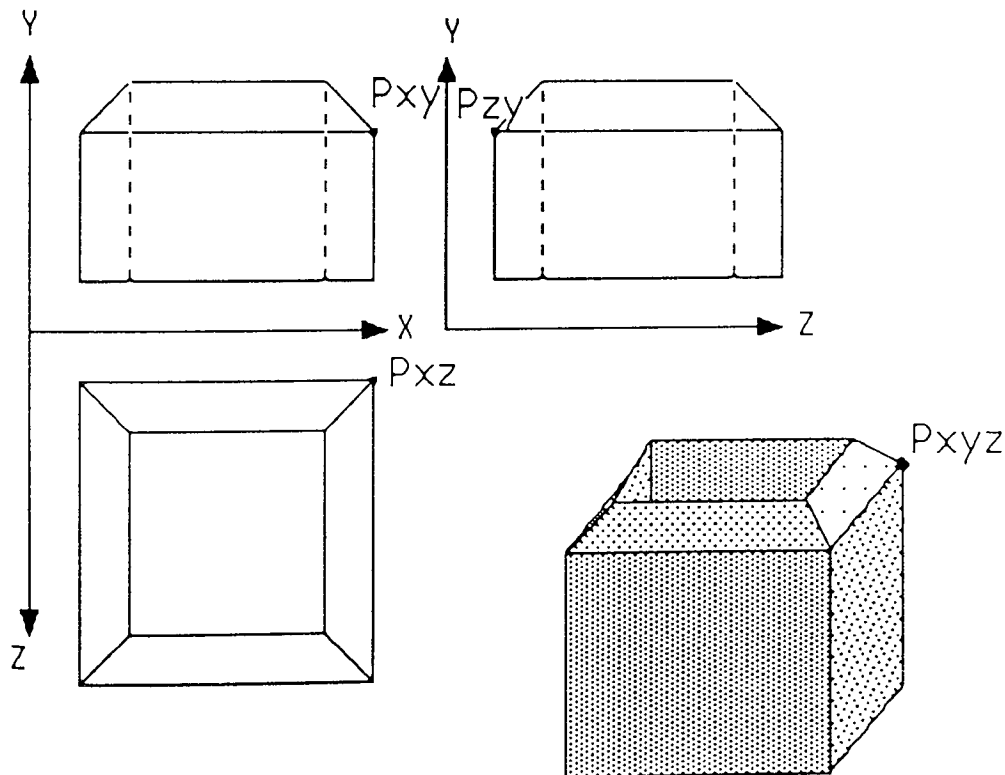


Figure 4.1 Input views

4.2.1.2 Transformation to the three-dimensional lines

In the case of polyhedra, all edges are direct lines. They are therefore represented as a pair of end vertices. Again, like the previous step, the 3D points created so far are used and all possible 3D edges that produce 2D pictures in different views are found. After completion of these two steps, a wireframe of edges is produced which, beside the edges of the original object, also contains some extra edges and vertices. These extra items are called ghost lines and ghost points or, in other words, "ghost figures" (bold part in figure 4.2).

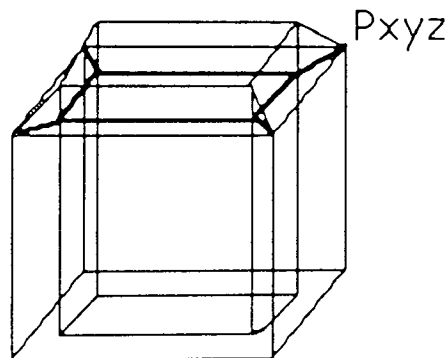


Figure 4.2 Ghost figures

4.2.1.3 Ghost figure elimination

The purpose of this step is to remove ghost lines and ghost points previously mentioned. To do this task, for the case of polyhedra, Idesawa [1973] proposed a set of twelve rules :

- (a) Isolating point is a ghost point.
- (b) Terminal point is a ghost point and associate edge line starting from this point is a ghost line.
- (c) Corner point is a ghost point and two associate edge lines starting from this point

are ghost lines.

(d) Middle point is a ghost point.

(e) Associate vertex is an intersection of three associate edge lines, and any two of them are on the same direct line. Then this point is a ghost point and the associate edge line not on the direct line is a ghost line.

(f) Associate vertex is an intersection of three associate edge lines, and these lines are on the same plane but not (e). Then this point is a ghost point and these three line segments are ghost lines.

(g) Associate vertex is an intersection of four associate edge lines on the same plane and less than two sets of associate edge lines are on the same direct line. Then this point is a ghost point and the lines which are not on the same direct line are ghost lines.

(h) Associate vertex is an intersection of n associate edge lines and more than $2n/3$ lines are lying on the same plane but there is no set of lines on the same direct line. Then more than one associate edge lines starting from this point are ghost lines.

(i) Considering the relations between adjacent points of an associate vertex, contradictions occur in the relations between faces, edge lines and vertices of solid body. Then this point contains some ghost lines.

(j) Investigating more adjacent associate vertices, inconsistencies are found out. Then it is probable that the associate edge lines which connect these points contain some ghost lines.

(k) Constructing the faces of solid, if contradictions occur, then it is probable that some ghost lines and/or ghost points are contained.

(l) When an associate edge line is eliminated, the three view is not changed. Then it is probable that the eliminated edge line is a ghost line.

There are some ghost figures which can be judged at the stage of constructing faces; they are eliminated in that stage.

4.2.1.4 Construction of the faces

To make a solid out of the generated wireframe, it is necessary to construct a set of faces. Therefore, after searching the three-dimensional edges, only those faces are selected which satisfy the following conditions :

- 1) There are n faces which contain a vertex given as an intersection of n edge lines.
- 2) An edge line constitutes the boundary of two faces, and runs in opposite direction to each other in the row of boundaries.
- 3) A boundary of a face is enclosed.

In this search an edge line is distinguished as a ghost line and eliminated if cannot be any boundary of faces.

The main weakness of this method is its application limitation: it is used only for polyhedral objects and therefore cannot handle objects having curved surfaces. Another negative point about Idesawa's method is the lack of a reliable algorithm to recognize false elements from true elements, which may lead to deletion of true elements.

4.2.2 On Automatic Recognition of 3D Structure from 2D Representation

A new approach to the problem has been presented by Aldefeld [1983]. Aldefeld's philosophy is to view a complex part as being composed of elementary objects belonging to a set of predefined classes, and to recognize these elementary objects by making use of the knowledge about the class-dependent pattern of their two-dimensional representations. It is assumed that there are three orthographic views (Front, Side and Top) for the object and the projection of all edges and boundaries,

including tangency lines, exists in input views. The classes of objects which are considered by the method are only uniform thickness objects with the base parallel to one of the coordinate planes. Each elementary object has a 2D pattern in each view which specifies the object. Each 2D pattern consists of a number of 2D primitives such as lines, arcs and circles.

4.2.2.1 Main components

Main concepts of the algorithm are shown in figure 4.3. The segmentation procedure is responsible for finding basic primitives that serve as building blocks in the formation of higher-level structures. The "attributes and relationship" process searches the data for one or more new entities during each cycle (e.g. a loop has been added), and assigns attribute values and inserts the required relationship. Finding substructures that their patterns verify represent partial structure of the objects is the task of a "Substructure selection" process. Heuristic information is used in this process. These substructures are passed to different modules for recognition of objects. Each module knows about exactly one object class.

4.2.2.2 Details of the method

For the case of uniform thickness objects, different types of entities and their relations are given in figure 4.4. The lowest level elements are the primitives. For example, $\text{CONTACT}(p,q)$ shows that primitives **p** and **q** have at least one common point each and $\text{ADJACENT}(a,b)$ means that **a** and **b** have a common primitive.

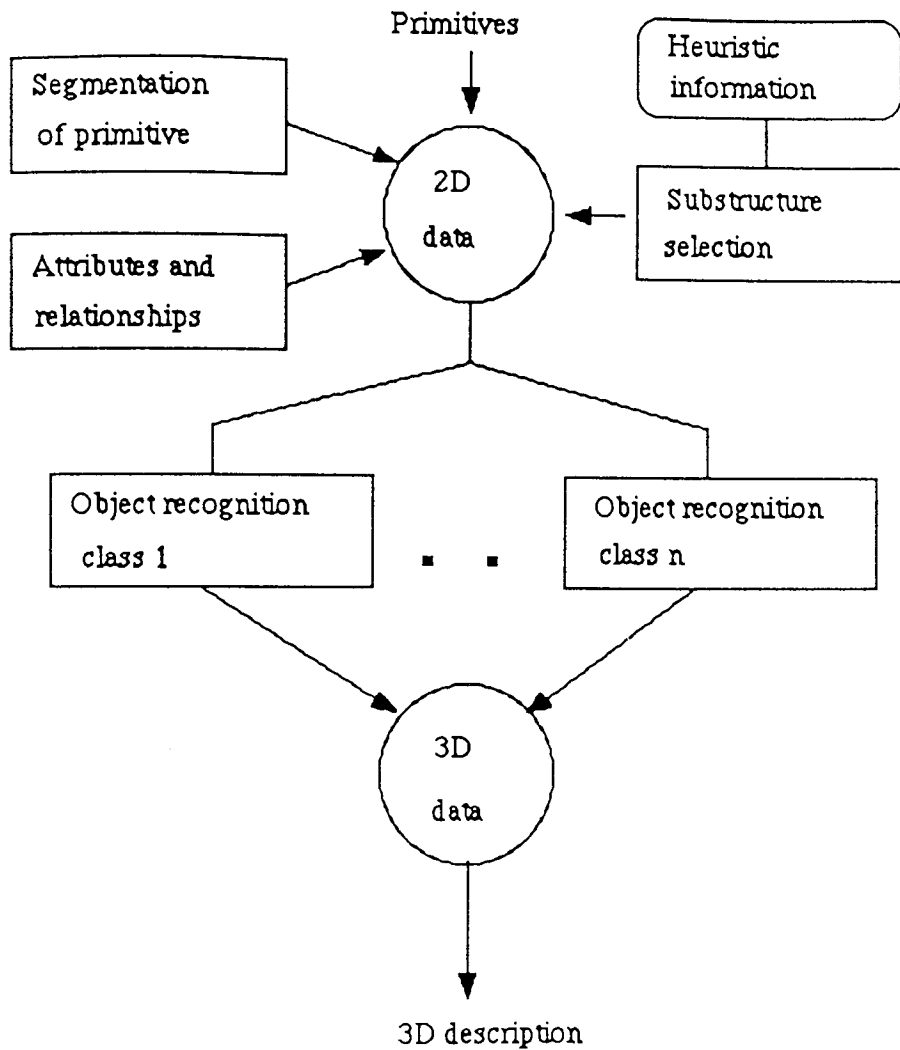


Figure 4.3 Main components of the system

Considering the example shown in figure 4.5, which complies with the condition mentioned earlier, one of the 2D view patterns is an arbitrary loop. It is referred to as the silhouette of the base (front view), and both of the two other patterns are composed of a rectangular loop and a number of line segments. These are known as lateral silhouettes.

The following model-guided recognition algorithm is proposed for recognition of uniform thickness objects for which a loop L in the base view is given:

- 1) Search in view V_2 (non-base view) and find all rectangles matching L .
- 2) Search in the other non-base view V_3 and find rectangles which match both L and those found in step 1.
- 3) Scan the loop L to find all features that need to have a line segment in one or both of the other non-base view. Examples are the corners formed by the primitives constituting loop L .
- 4) For each matched pair, find the complete set of line segments required by the features. If this is successful, then a complete object pattern given by the union of these matching loops and generated line segments has been found.

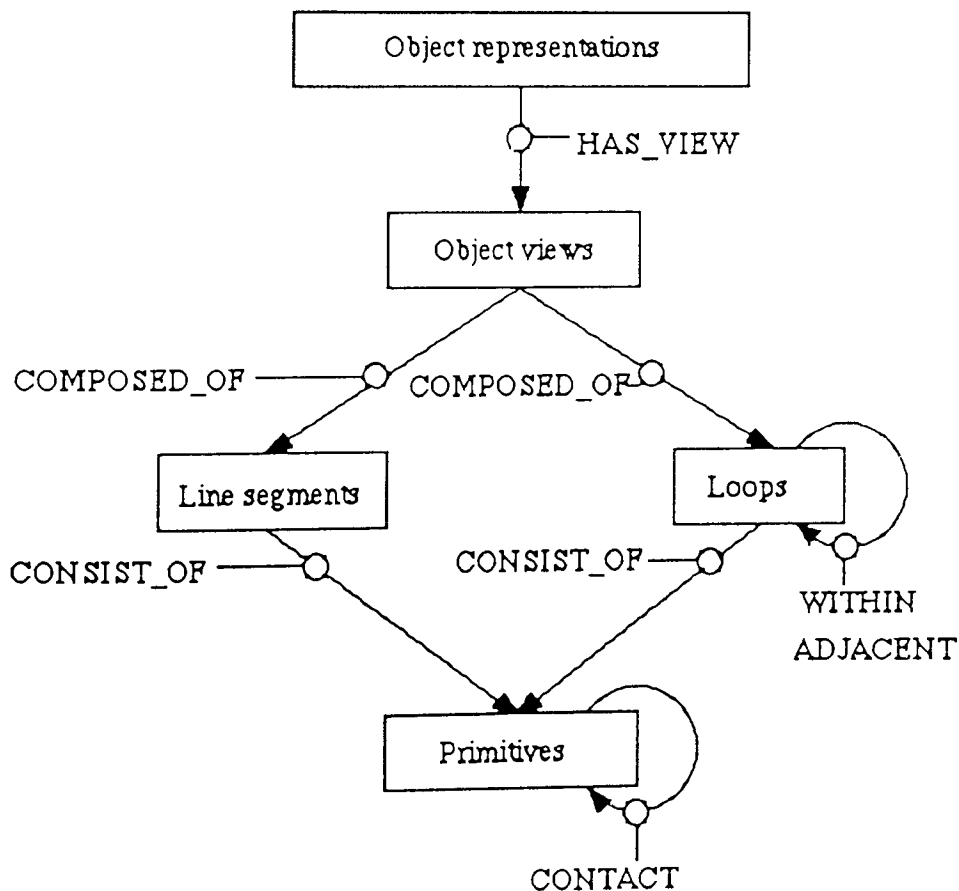


Figure 4.4 Types of entities and relationships defining the data structure

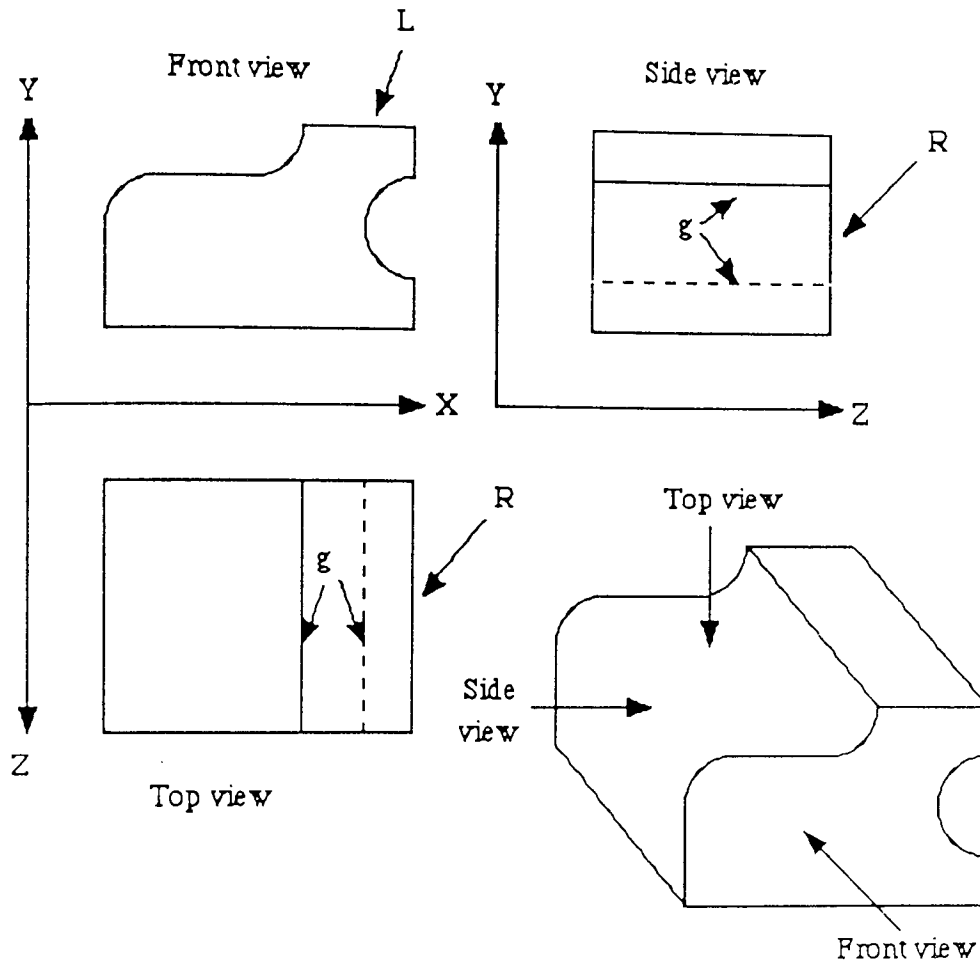


Figure 4.5 A uniform-thickness object

The model-guided algorithm will only work if true patterns of elementary objects are passed to it. Unfortunately, when an object is represented as orthographic views, because of the overlapping of faces and edges, it is not always straightforward to extract true patterns. Aldefeld uses a heuristic method to solve this problem. His technique is based on a best first search method which uses an evaluation function that assigns scores to patterns on the weight of their characteristics. Patterns are then selected from the score list (the highest score first) and they are passed to the recognition process. Two different ideas are used in scoring. The first is to let the score of each loop depend on the number of its primitives that have not yet been recognized

as part of an object representation. Second, to let the scores depend on the occurrences of relationships and on the values of attributes. For example, a pattern which is not adjacent to any other pattern (i.e. isolated), is given a higher score than a pattern which has the attribute "adjacent" assigned to it, because an isolated loop must necessarily represent a silhouette of at least one object.

After the above discussion Aldefeld's complete algorithm can be expressed in the following major steps:

- 1) Create the relational structure of primitives.
- 2) Find and store all elementary loops, and assign their attribute values and insert their relationships. Tag all loops as "open".
- 3) Assign a score to each open loop according to the evaluation function [opcit]. Select the loop, L, with the highest score.
- 4) Assuming the loop L as the base silhouette, then by using the model-guided recognition algorithm, try to reject or verify this hypothesis. If complete patterns are found, then create the corresponding 3D structure.
- 5) Generate all loops that include both L and one of the loops adjacent to L. Check for existence of each loop; if it does not exist, add it to the set of data together with its attributes and all new relationships. Tag new loops as "open" and L as "closed".
- 6) According to the engineering drawing rules, check whether objects found so far conform with all input data. If yes, end the procedure; otherwise continue with step 3.

Since the Aldefeld's algorithm works with one partial object at a time, and ignores the global context, it is able to work only on a local basis. This becomes one of its great weakness in differentiating between solid bodies and cavities. Hence, it may produces wrong partial solids because of the silhouette interferences. Also, its limited domain of uniform-thickness objects is another disadvantage. To generalize the method to non-uniform objects therefore not only needs more heuristics, but also needs the extension

of the domain of partial objects to include those objects which, for example, have rotational symmetry, and relaxation of the restriction on their spatial orientation.

4.2.3 Recognition of 3D Objects from Orthographic Projections

This method, due to Gu, Tang and Sun [1985] offers another way of tackling the problem. It makes some improvement and complements to the Wesley-Markowsky algorithm, which is a typical hierarchical construction algorithm limited to polyhedra objects, and extracts the idea of pattern recognition expressed in Aldefeld algorithm. The main problems addressed here are:

- 1) Finding all solutions for the same three views.
- 2) How to handle pathological cases.
- 3) How to cover as many types of objects as possible.

4.2.3.1 Reconstruction algorithm

A step-wise method has been proposed here, as shown in figure 4.6. In the following each step is explained in more detail:

(1) Input the views

Input views comprising of straight lines, ellipses (including arcs), hyperbolas and regular higher order curves are input either by keyboard or by digitizer.

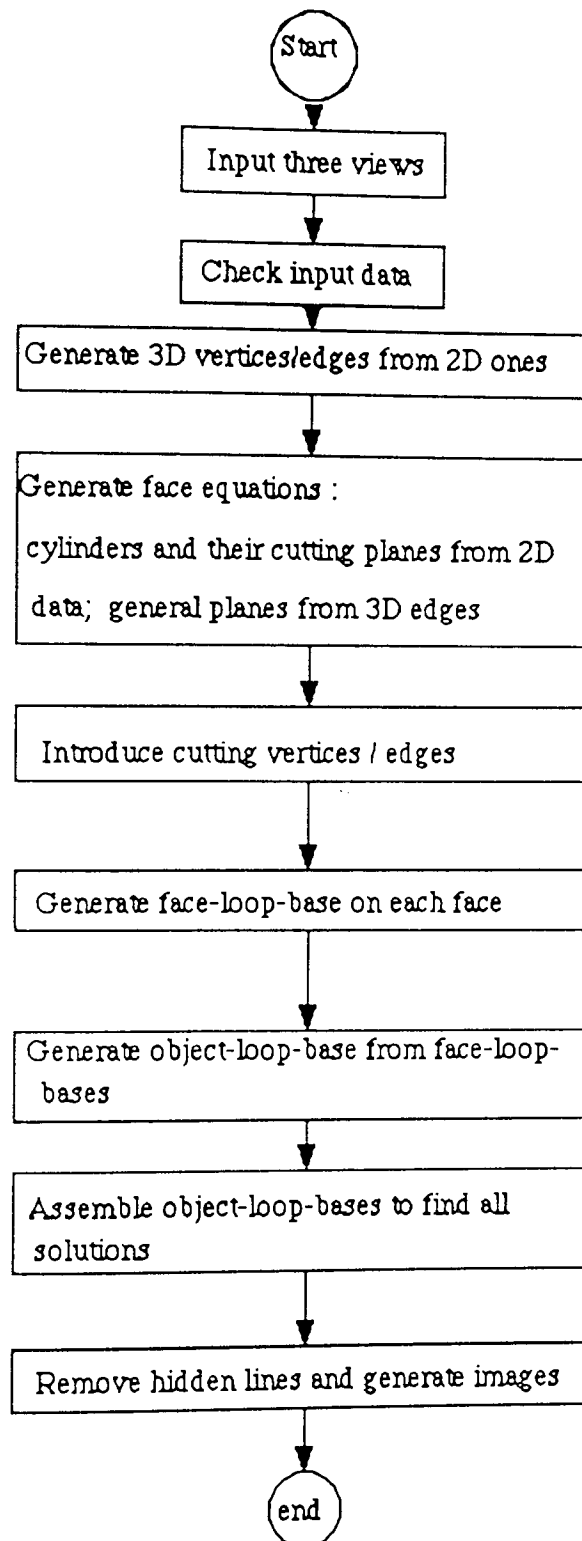


Figure 4.6 Flowchart of the algorithm

(2) Check input data

Check the input data for validity.

(3) Generate 3D edges

Three types of 3D edges, straight lines, ellipses and regular higher order curves are allowed in this method. For generation of each type a set of principles is considered as described below:

Matching principle : Let E_f , E_t and E_s be projections of edges or vertices in front, top and side views respectively. They are represented as a group of matching edges if their surrounding rectangles, which are shown by vectors $[X_{fmin}, Z_{fmin}, X_{fmax}, Z_{fmax}]$, $[X_{tmin}, Y_{tmin}, X_{tmax}, Y_{tmax}]$ and $[Y_{smin}, Z_{smin}, Y_{smax}, Z_{smax}]$ respectively, satisfy the following conditions :

$$\begin{array}{ll} X_{fmin} = X_{tmin} , & X_{fmax} = X_{tmax} \\ Y_{tmin} = Y_{smin} , & Y_{tmax} = Y_{smax} \\ Z_{fmin} = Z_{smin} , & Z_{fmax} = Z_{smax} \end{array}$$

This constitutes a group of matching edges that may be a set of projections of one 3D edge, regardless of its shape.

Line mode : A 3D straight line can be extracted from a group of matching edges E_f , E_t and E_s in three view if and only if

- a) E_f , E_t and E_s are 2D straight lines (at most one of which can be a 2D vertex);
- b) There is a group of their endpoints coinciding with the matching principle.

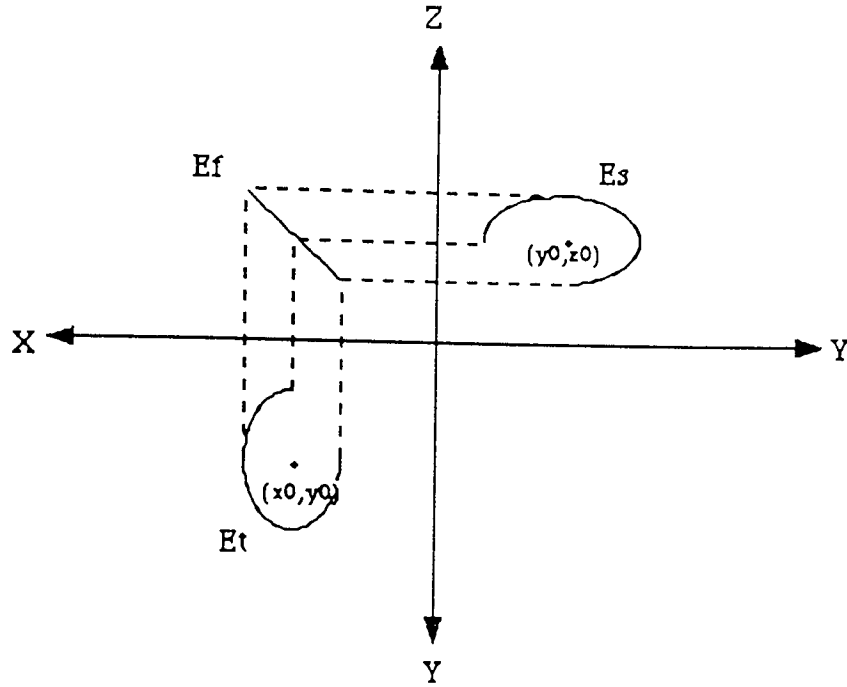


Figure 4.7 Ellipse mode

Ellipse mode : Assuming that the axis of the generating cylinder of an ellipse is parallel to OXY plane, as depicted in figure 4.7, then a 3D ellipse edge can be derived from a group of matching edges E_f , E_t and E_s if and only if

- a) E_f , E_t and E_s are either 2D ellipses with their axes parallel to coordinate axes or 2D straight lines, and at least one ellipse as well as one straight line exists among them;
- b) If two ellipses exist among E_f , E_t and E_s , then their centres have the same coordinate value in their shared coordinate;
- c) Each group of end points of ellipses, as well as another group of points on ellipse arcs, satisfy the matching principle.

Higher order curve mode : Intersections of two cylinders with different radii or non-crossed axes whose axes are parallel to coordinate axes, are defined as higher order curves. It may be derived from such a group of matching edges E_f , E_t and E_s as two of which are circular arcs while one of which is either a hyperbola or regular higher order

curves as shown in figure 4.8.

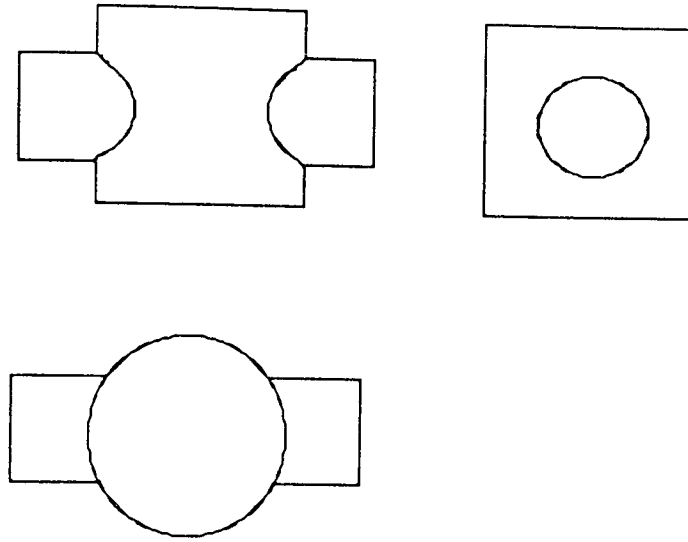


Figure 4.8 Higher order curve mode

(4) Generate face equations

Any couple of non-collinear 3D straight lines which have a common endpoint may be used to produce the basic plane equation. A cutting plane will be generated by sweeping its projective line on one view. Normally for the definition of a cylinder three kinds of geometric parameters are required. For instance, these parameters can be one point on the cylinder axis, known as the location point, the radius and a group of orientation numbers of the axis. These parameters can be derived from higher curve mode. For ellipse mode, this calculation is a little more complicated, as is described here.

Derive location point : matching principle is used to find the centre of the 3D ellipse.

Derive radius : By reference to figure 4.9 if the axis L of the generating cylinder M of 3D ellipse E is parallel to OXY plane, and the generating plane N of E is perpendicular

to OXY plane but not to OXZ plane, then:

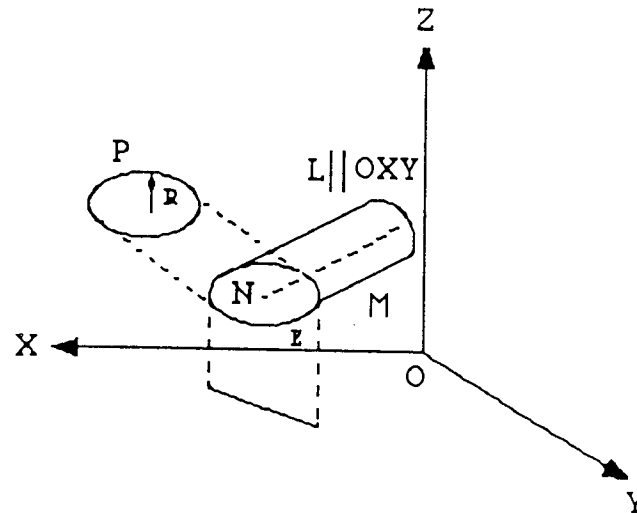


Figure 4.9 Derive radius

- a) the orthographic projection of E on OXZ plane is an ellipse P with its axis parallel either to X or Z axis;
- b) the length of the axis of P parallel to Z axis is equal to the diameter of M.

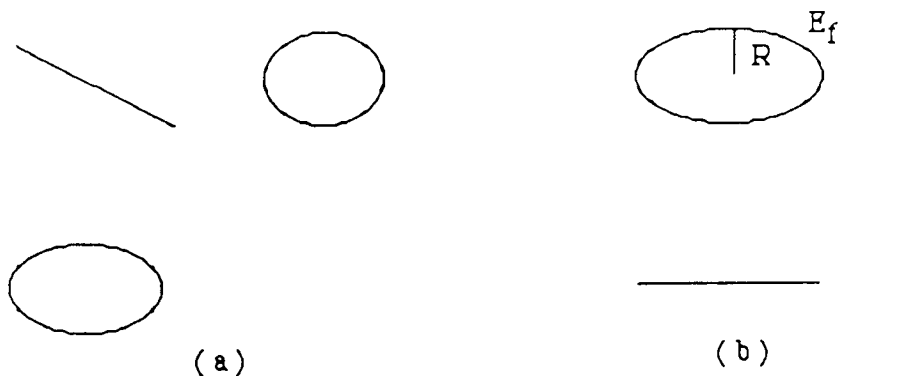


Figure 4.10 Two cases to derive radius

At the beginning it should be decided which coordinate plane the axis L of the cylinder M is parallel to, then derive the radius. In this regard two examples are shown in figure 4.10. In 4.10(a), only one straight line exists in the group of matching edges, and axis L must be parallel to the plane OXZ on which the line is located. In 4.10(b), only one

ellipse E_f exists in the group of matching edges, and the length of short axis of E_f is just equal to that of the diameter of the cylinder.

Derive a group of orientation numbers of axis : According to the figure 4.11, points $O(x_0, z_0)$ and $A(x_1, z_1)$ have been selected on line E_f corresponding to the centre point (x_0, y_0) and a point (x_1, y_0) of ellipse E_t respectively. Now, the point O (front view) is chosen as the centre, an auxiliary circle is drawn with the radius of the generating cylinder as its radius. Then a group of orientation numbers (dx, dz) of tangent line AT_1 is that of the projection of the cylinder axis. Using the same method for side view, a group of orientation numbers (dy, dz') is found. Therefore, letting $dz=dz'$, then (dx, dy, dz) is what is being sought.

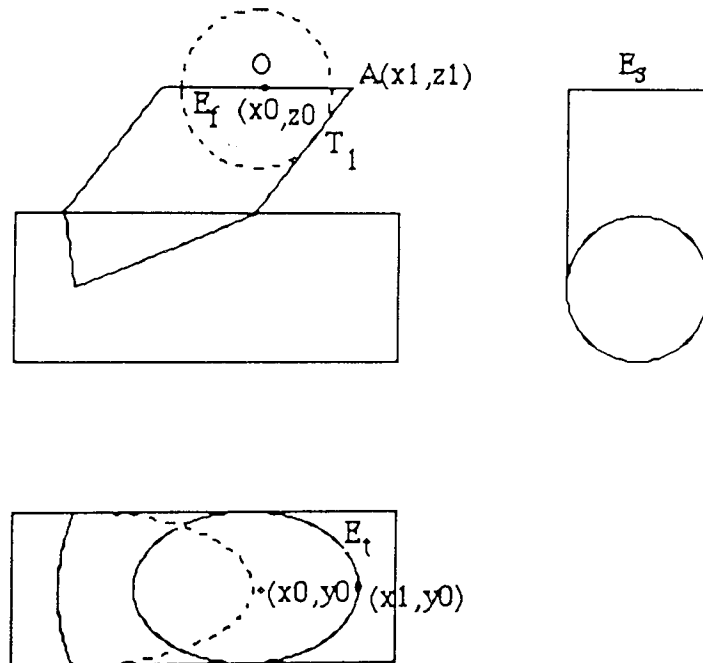


Figure 4.11 Using auxiliary circle to find cylinder axis

(5) Cutting vertices and cutting edges

Among 3D edges and faces, there may be two types of pathological cases. One happens when two edges intersect each other at one of their interior points rather than endpoints, like points P and Q in figure 4.12 which do not appear in orthographic views. Another type occurs when two faces intersect each other and generate an interior

line (e.g. AEGC and DHFB in figure 4.12, which produce edge PQ). Of course these cases do not exist in any well-defined geometric object.

This pathological intersection point/line is referred to as a cutting vertex/edge. To avoid pathological cases, this cutting vertex/edge is used to split its two generating edge/faces into four.

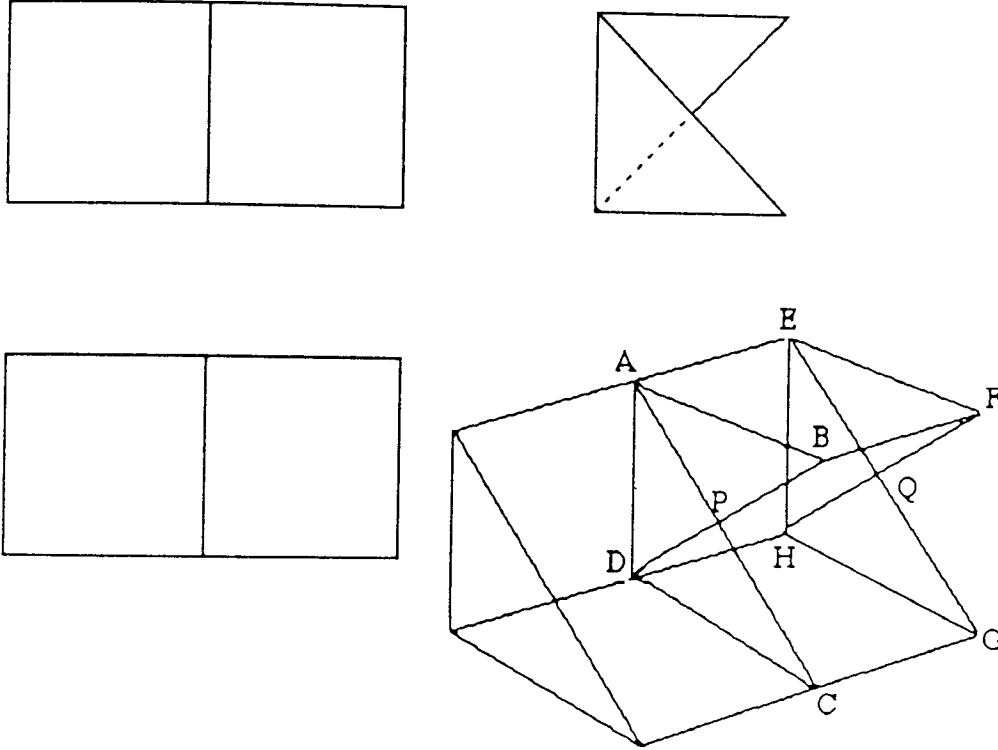


Figure 4.12 Cutting vertex/edge

(6) Generate face-loop-bases

A face-loop of face f is a simply interconnected area bounded by a subset of edges on f (figure 4.13). Sign "+" is used to denote the union of face-loops and " $E(Sf_i)$ " to show the boundary edge set of the face-loop Sf_i .

The boundary edge set $E(Sf_1+Sf_2+...+Sf_m)$ of the union of the face-loops $Sf_1, Sf_2, ..., Sf_m$ is defined as :

$$E(sf_1+sf_2+...+sf_m)=\bigcup_{i=1}^m E(sf_i)-\left(\bigcup_{i=1}^{m-1} \bigcup_{j=i+1}^m E(sf_i) \cap E(sf_j)\right)$$

Applying the definition for the example shown in figure 4.13, the following results will

be achieved :

$$Sf_4 = Sf_1 + Sf_2 + Sf_3$$

$$Sf_5 = Sf_1 + Sf_2$$

$$Sf_6 = Sf_2 + Sf_3$$

$$Sf_7 = Sf_1 + Sf_3$$

Assume that $Bf = Sf_1 + Sf_2 + \dots + Sf_k$ is a set of face-loops on face f . If any face-loop on f can be produced from one or more faces in Bf , but each face-loop in Bf cannot be generated from other face-loops in Bf , then set Bf is called a face-loop-base on f . For instance, for the example shown in figure 4.13, the face-loop-base on face f is :

$$Bf = Sf_1, Sf_2, Sf_3$$

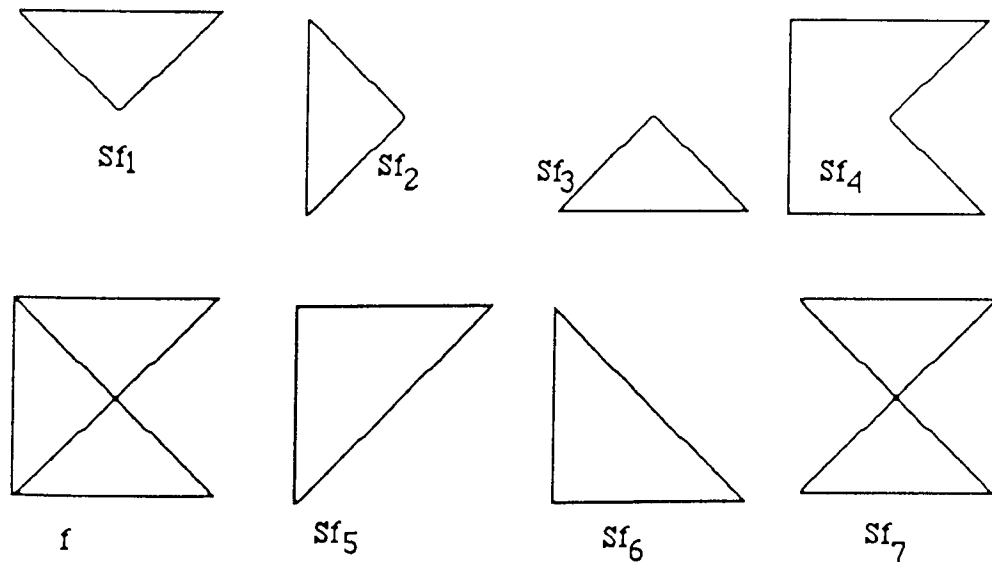


Figure 4.13 Face-loops

Suppose that $E(f)$ and $V(f)$ are the set of edges and the set of vertices on face f . If the dangle edges and the vertices with degree less than two are deleted as well as the cutting vertices, then all of the edges in $E(f)$ make a polygon mesh on f . Each polygon

in the mesh is either disjoint or connected with some vertices or/and edges in common. It is clear that each polygon in the mesh cannot be generated from any union of other polygons, while any bounded area can be generated by the union of these polygons. Therefore, all the polygons in the mesh form the face-loop-base on face f .

The face-loop-base for a face f may be found as follows :

- * For each vertex V_i on f , sort its incident edges on f in counter clockwise order such as $e_1 e_2 \dots e_k$. Then e_1 is the left-adjacent-edge of e_2 at V_i , and e_k is the left-adjacent-edge of e_1 .

- * Select an ordered edge $e_i(V_i, V_j)$ followed by picking its left-adjacent-edge $e_j(V_j, V_k)$ at V_j , then pick the left-adjacent of e_j at V_k , and so on. The face-loop will be formed when edge $e_m(V_m, V_i)$ joining the first edge e_i is picked. The left side of each ordered edge is defined as the interior of L .

- * Since there are two ways of traversing each edge, either from V_i to V_j or *vice versa*, all of the face-loops will be obtained when each edge is picked twice in different directions.

- * The face-loop-base is made from all of the bounded face-loops except unbounded ones.

(7) Generate object-loop-base

The concept of face-loop-base and union of face loops are extended here, and the philosophy of edge-loop-base and the union of object-loops are derived. For a description of the method, consider figure 4.14.

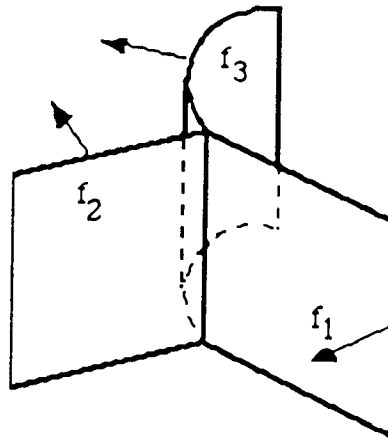


Figure 4.14 Incident faces

Suppose an ordered face $-f_1$, in which the "-" sign indicates that the interior of the object-loop to be produced is on the opposite side from that to which the normal of f_1 points, is selected. Then pick the ordered face $-f_3$ to ensure that there is not any face-loop in the interior of the object-loop. If $+f_1$ is selected first, then the second should be $-f_2$. An object-loop B will be formed if the ordered face-loops are picked repeatedly until each edge in B has been included in two face-loops in B.

By traversing each face-loop in two direction, all of the object-loops will be obtained. The object-loop-base is composed of all the bounded object-loops and no unbounded ones.

(8) Assemble object-loops

Object-loops are assembled according to the following steps to produce solutions :

- * Deleting a face-loop which is shared by two object-loops only, because any interior face in an object is not allowed.
- * Deleting any edge that is shared only by two face-loops which are on the same

face.

The generated object is an acceptable solution if its orthographic views comply with those given in the input. All the solutions matching with the given 2D projections are found by checking all of the assembly of the object-loops.

(9) Remove hidden lines

Finally hidden lines are removed to produce the object image.

4.2.4 Inputting Constructive Solid Geometry Representation Directly from 2D Orthographic Engineering Drawings

Another method, due to Ho Bin [1986], for interpretation of 2D engineering drawings is presented here. This method regards parts and objects as a combination of primitives. These representations are input directly from 2D orthographic views by a digitizer. Each primitive is input according to the following steps :

Step 1 - Input the type of primitive. Only five types of primitives are accepted: cuboid, tetra pyramid, cylinder, cone and sphere. The axes of each primitive are assumed to be perpendicular to one of the projection planes.

Step 2 - Each primitive is assigned a "+" sign if it occupies a part of space or "-" sign if it is a hole or a cavity. This sign is also input.

Step 3 - Input three points from the base of the primitive.

Step 4 - Input two points to specify the height of the primitive.

Then the following data are derived from the coordinates input for the five points :

* The numbers of the view on which its base and height are projected.

- * The three coordinates of the base centre.
- * The radius of the base circle for the case of cylinder, cone and sphere.
- * The length, the width and the angle of the base rectangle if the primitive is cuboid or pyramid.
- * The value of the angle between the axis (height) and the X-Y projection plane, or between the axis (height) and the X-Z projection plane if the axis is parallel to X-Y plane.
- * The value of the primitive height.

After completion of the input cycle for a primitive, a 3D model of the primitive is built and 2D projections are displayed on the screen. This intermediate output is compared by the user, with the input; the user then decides if it is correct to continue to input the next primitive.

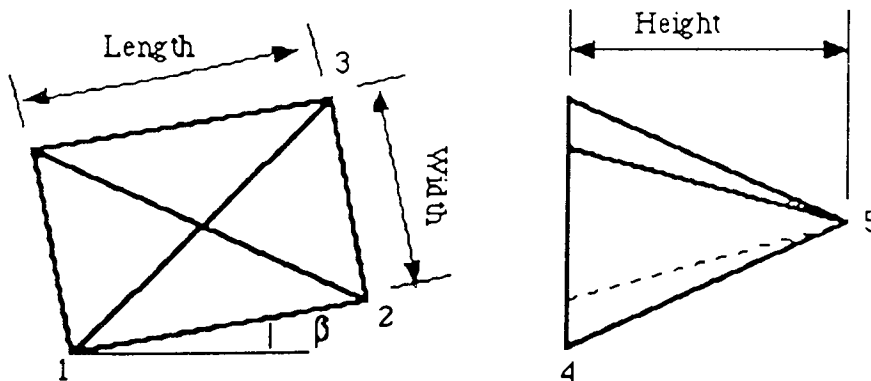


Figure 4.15 Views of an input primitive

At the end of the input process, information on all primitives including their sign is used for the construction of a binary CSG tree.

Some aspects of the algorithm used for conversion of 2D orthographic views into 3D model will be explained with examples. Considering the figure 4.15, when the axis of

the input view is perpendicular to one of the projection planes, the real shape of its base contour and the true length of its height will appear in the principal views (e.g. points 1, 2 and 3). Therefore the base parameters (radius and the coordinates of the centre for circular contour; length, width and angle between the length direction and the horizontal line, and the coordinates of the centre for the rectangular contour) can easily be calculated.

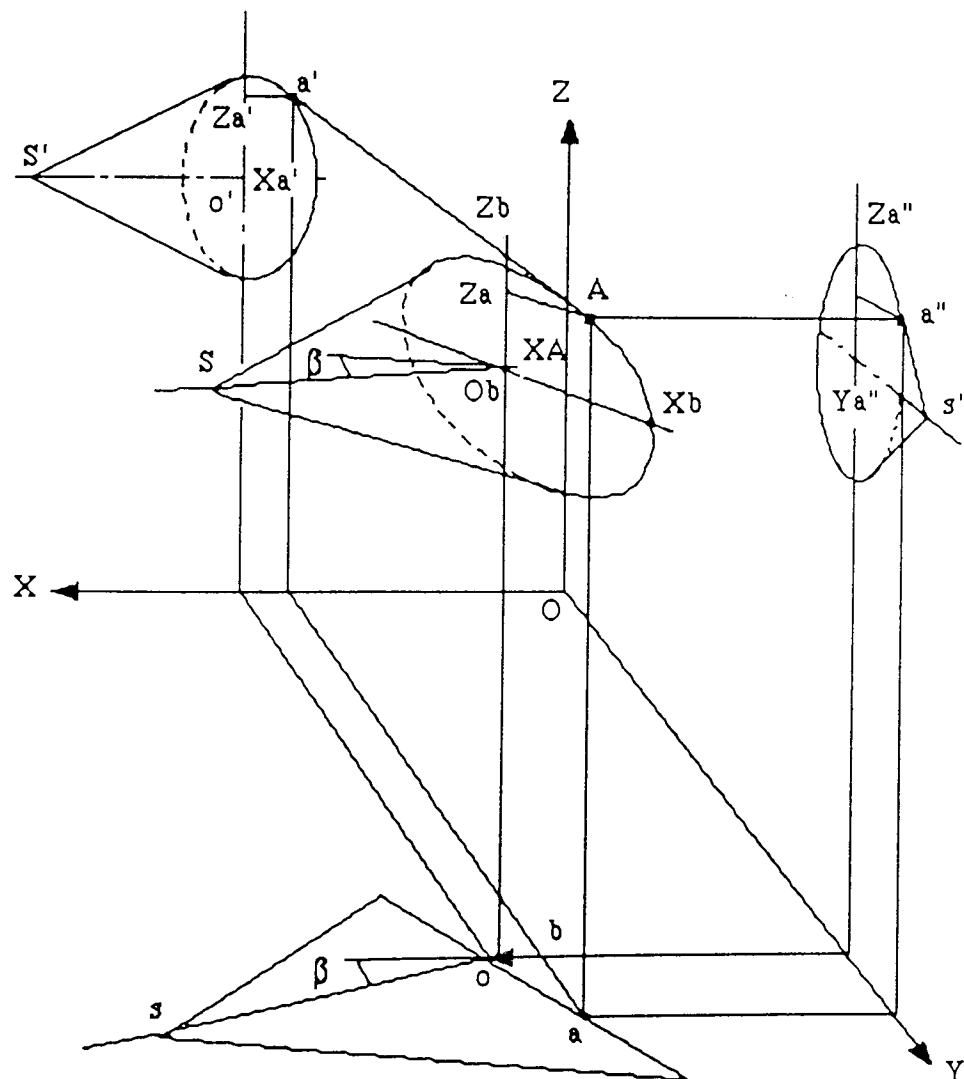


Figure 4.16 Cone and its three principal views

If the axis of an input primitive is oblique to the projection plane, the real shape of its base does not appear in the principal views. In this case the real shape must be constructed in order to find its dimension as explained in the next example.

Consider the cone example shown in figure 4.16 together with its three view projections. Axis SO_b is parallel to the X-Y projection plane and has an angle β to the X-Z projection plane. An arbitrary point such as A is selected on the contour with its projections called a, a', and a". O_bX_b and O_bZ_b are referential coordinate axes of the circular base contour of the cone. The X_b coordinate of point A, AZ_a is equal to ao in the top view. The X and Z coordinates of projection a' relative to the projection of the cone centre o' in the front view, are a'Za' and a'Xa'. The Y and Z coordinates of projection a" relative to projection o" in the side view are a"Za" and a"Ya". The X and Y coordinates of projection a relative to projection o in top view are bo and ab . Since:

$$a'Za' = bo,$$

$$a"Za" = ab,$$

then

$$AZ_A = a'Za' / \sin\beta,$$

or

$$AZ_A = a"Za" / \cos\beta,$$

and

$$AX_A = a'Xa' = a"Ya"$$

This shows that the X_b and Z_b coordinates of the arbitrary point A can be calculated

from coordinates of its 2D projections only if the coordinates of the centre point of the ellipse projection are known.

4.2.5 Automatic Construction of curvilinear Solids from Wireframe Views

This work, due to Remi Lequette [1988], is again based on the notion of using wireframe modelling as an intermediate stage. The same class of object is considered as by Sakurai and Gossard [1983]. This method, however, sometimes uses two views rather than three views and edge generation for tangency edges is different. Here it tries to find the tangency edges on the surfaces, whilst Sakurai and Gossard find them from some features on the projections. Both methods use heuristics in object generation, but in different ways. The algorithm used by the author can be summarised into the following steps:

- a) Checking and preparing of input data
- b) Constructing an intermediate 3D wireframe model
- c) Finding surfaces and candidate faces in the wireframe
- d) Generating all solids that fit with the input data

4.2.5.1 Checking and preparing input data

Two or three sets of 2D vertices and edges are accepted as input data for corresponding views. A segment is defined as a linear or circular curve and is delimited by two nodes. Then data is prepared as follows:

- * Find intersections between segments to create new nodes and split segments.
- * Find special nodes which are silhouette nodes where the tangent to a curve is parallel

with the projection direction, and tangency nodes where two segments have a tangency continuity.

* Record segments supported by the same curve in a linked structure.

4.2.5.2 Construction of wireframe

This section is accomplished in three steps:

1) Construction of vertices of different types including silhouette vertex, tangency vertex and special vertices.

2) Construction of edges as well as axial edges, nonaxial edges and axial edges perpendicular to the view. In the case of view-dependent edges, that view must be considered too.

3) Choose a complete wireframe model associated with the projections (example in figure 4.17). The silhouette edges are also found.

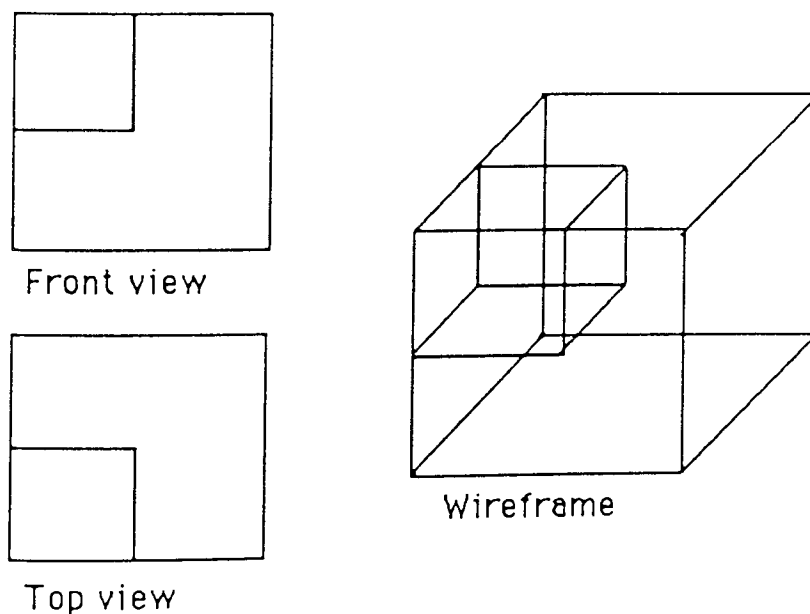


Figure 4.17 Input views and its wireframe representation

4.2.5.3 Surfaces and faces

After creating a pseudo-wireframe from the set of projections, all surfaces in the wireframe will be specified. In this process for identifying a surface, two edges with a common vertex are selected and the following rules are applied:

- a) Two straight lines define a plane.
- b) A straight line and a circle define a cylinder, a cone or a plane, depending on their respective orientations.
- c) Two circles define a sphere or a torus.

By using a depth first search method, all the edges in the selected surface are found, for example the highlighted part of figure 4.18. In this process, it is also possible to create missing tangency edges.

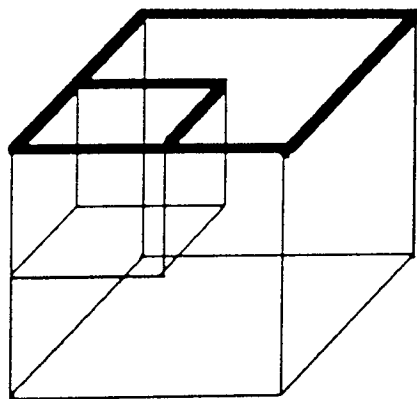


Figure 4.18 Surface construction

4.2.5.4 Solid generation

This is the last stage of the algorithm, in which all solids that can be made with the set of faces are found and they are examined with the original input views. The number of 3D edges that map to a specific 2D segment are counted and assigned to its

corresponding segment in each view (performing segment counts). When the solid is built some edges will disappear from the wireframe (probable ghost ones), therefore the number assigned to each segment will decrease (figure 4.20). The aim is to keep this number positive, because if it becomes zero it means the solid does not correspond with the view (a segment is missing).

During the process each face will receive one of the following labels :

- *Undetermined* : to show that the face has not been examined by the algorithm.
- *Oriented* : it means the face is in the boundary of the solid.
- *Pseudo-face* : it identifies that the face is not part of the solid (figure 4.19).

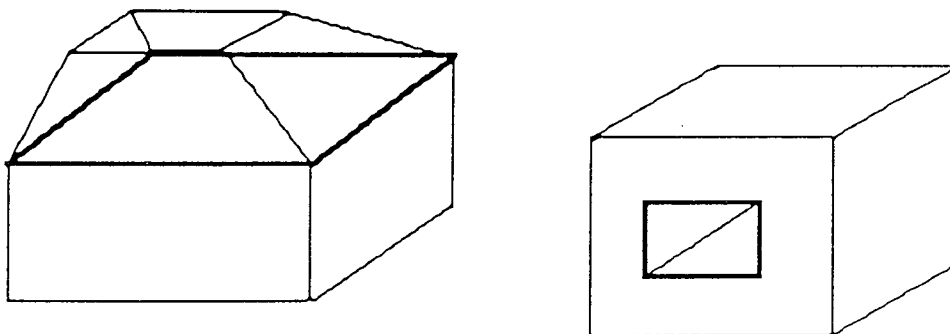


Figure 4.19 Examples of pseudo faces (in bold)

The following cases can happen when the faces around an edge are examined :

- 1) There are two or more undetermined faces, with nothing known about the edge.
- 2) There is one undetermined face, which can be fixed using the Moebius rule [Mortenson, M.E., 1985]. Pseudo-faces are eliminated. If there is one oriented face the undetermined face should be oriented, if there are zero or two oriented faces the undetermined face is a pseudo-face. This then leads to case 4, 5, or 6.
- 3) There are no undetermined faces, in other words all of the faces have been examined by the algorithm but the Moebius rule is not respected.
- 4) All faces are pseudo-faces. The edge is then a pseudo-edge and is not in the wireframe of the solid.

- 5) There are two oriented faces on the same surface, and the edge is a ghost edge.
- 6) Cases 1 to 5 are not applicable, and the edge is a correct edge of the solid.

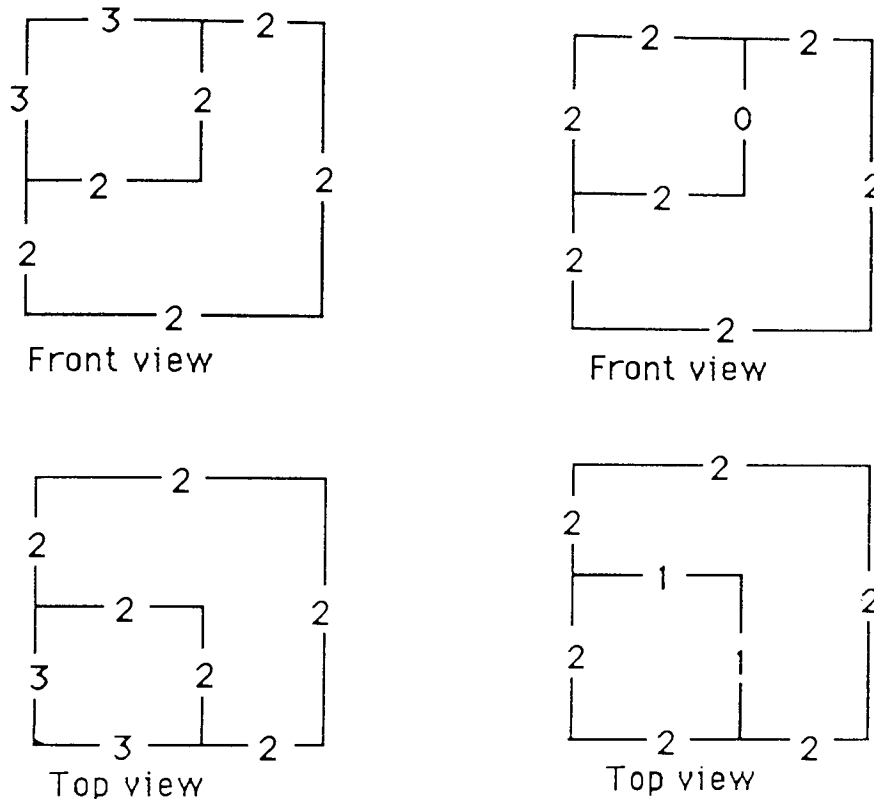


Figure 4.20 Segment counts (left: at the beginning, right: before backtracking)

In cases 4 and 5 the edge is not in the wireframe and is not seen on the projected views. Thus, for each segment onto which the edge projects, the number of edges that project onto the segment is decreased (segment count is decreased by one).

Finally the algorithm for construction of the solid can be outlined. At the beginning all faces are labelled as undetermined, and it is ended when they are all oriented or pseudo-face:

- * First choose an arbitrary undetermined face, called a critical face, and label it oriented. Then examine all the edges of this face and try to fix new faces using case 2, the same being done with the edges of these new faces. When there are no new faces

repeat the process with a new critical face.

* If case 3 (Möbius rule violated) or case 4 or 5 (invisible edge) applies to an edge and a segment is missing on a view, then the solution being constructed is not correct. The process must stop and backtrack, until the last critical face that has been oriented is obtained. This face is labelled pseudo-face and the label undetermined is given to all the faces labelled after this critical face. The process is repeated.

4.2.6 Discussion

Briefly comparing the different reviewed methods of transforming 2D engineering drawings into 3D solids shows that most of these techniques rely on the wireframe building as a base, and then try to make a boundary representation from it. In this process many unwanted items, such as ghost figures, pseudo planes and so on, are created. As a result, much effort is required to reject false items and select the correct ones, which is time consuming. Moreover, dealing with many edges and vertices and checking them for validity brings additional costs through the need to apply many rules, possibly with backtracking.

Those like Ho Bin, who use primitives as a base, are not acting automatically, and their methods cause the whole process to become lengthy due to the intervention of the user.

These problems led Kargas, Cooley and Richards [1988] to think about another method to work with minimum data, which requires minimum intervention by the user. Hence, they proposed a method based on CSG (Constructive Solid Geometry) models using primitives, which will be described in next section. The efficiency of the algorithm will be discussed later.

4.3 Interpretation of Engineering Drawings as Solid Models

In the recent project [Kargas, Cooley & Richards, 1988] the concepts of Constructive Solid Geometry (CSG) have been used as a base to approach the problem. Each primitive is assumed to be represented as a set of unique signatures in orthographic views (refer to figure 3.9) provided that the axes of primitives are parallel to a coordinate axis. Then, within a tree structure used to describe the object, each primitive may be identified by its corresponding signatures. The object is assumed to be cut-out

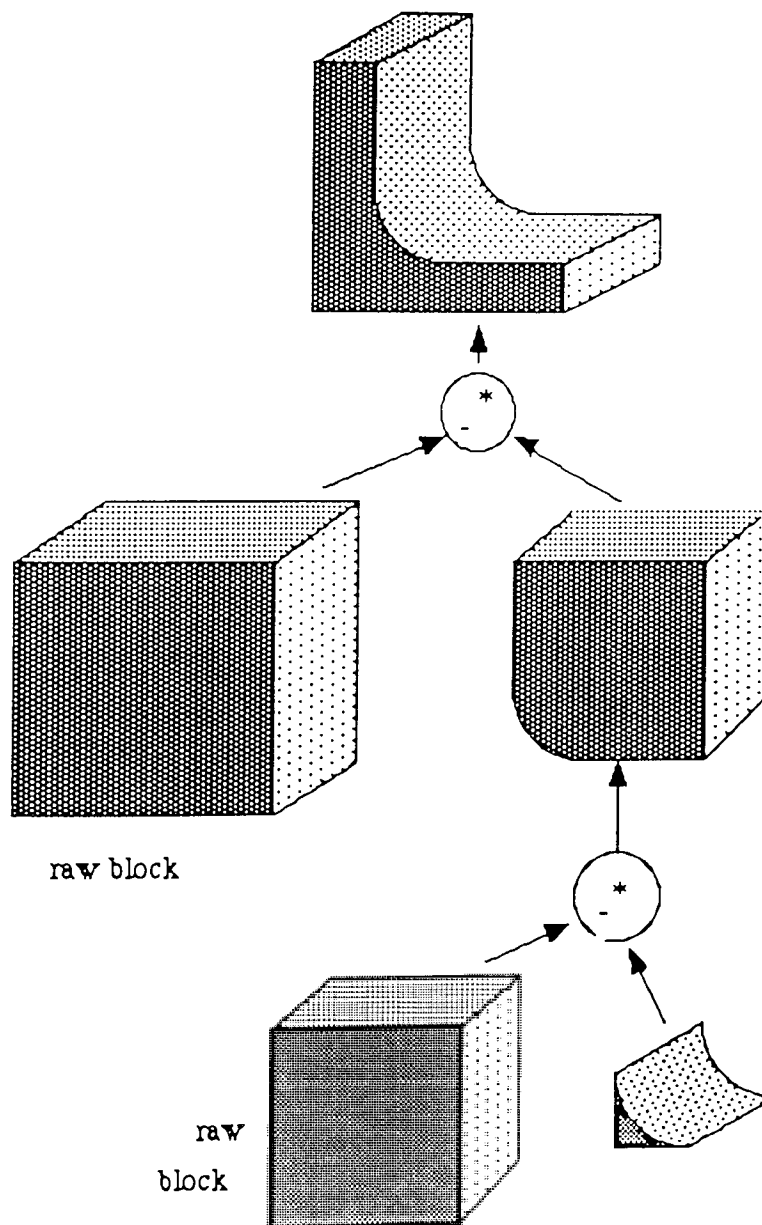


Figure 4.21 Object "cut-out" from a "raw block"

from a single "raw block" as shown in figure 4.21. Here, the objective is to find the volume of materials that must be cut out from the raw block in terms of primitives and a set of boolean operations applied on them. The raw block itself is a 3D primitive or a unit block which is transformed (scaled). A positive sign ("+") is assigned to the raw block and "-" sign to the objects which are cut from the raw block.

4.3.1 The First Approximation Model

According to the assumption the orthographic views of raw block are rectangles which surround the object's 2D views. Therefore the problem is simplified to a two-dimensional one. One may then try to find 2D signatures of primitives that must be cut from the surrounding rectangle to produce the boundary of each view of the object (figures 4.22 and 4.24). When this operation is performed, each 2D signature in a view is used as a base of a uniform-thickness primitive with the third dimension equal to the maximum length of other dimension from any other views. On the other hand an object's model is produced by the extrusion of the boundary of each view and then finding their intersection.

The model which has been generated by this method is called "The First Approximation Model", because it does not represent the real object model in all cases. This method of approaching the problem by creation of the first approximation model has many advantages over previous methods such as :

- It directly converts 2D orthographic views into CSG form.
- It does not need any intermediate step such as wireframe representation.
- It does not have any backtracking and therefore the algorithm is faster.
- It does not produce false objects or ghost figures: in other words only a single solution is found.

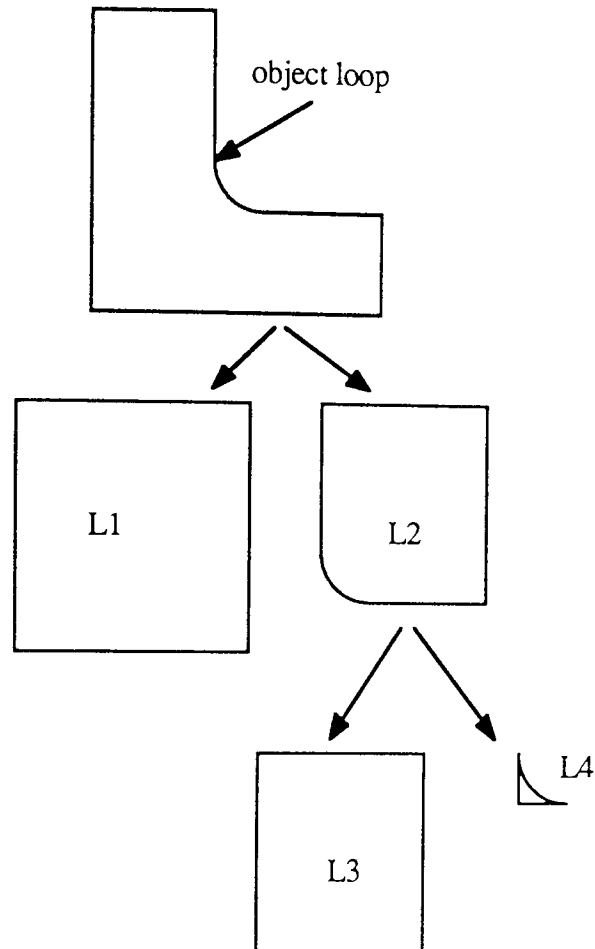


Figure 4.22 Boolean tree of two-dimensional base views

4.3.2 Method of creation

For the whole process of transforming the engineering drawings into a 3D solid model an algorithm as shown in figure 4.23 has been proposed. In this algorithm user intervention has been suggested together with the feedback process for the cases where the output does not represent the object. The algorithm comprises of four steps :

- a) raw data interpretation
- b) data analysis
- c) three-dimensional modelling
- d) output verification

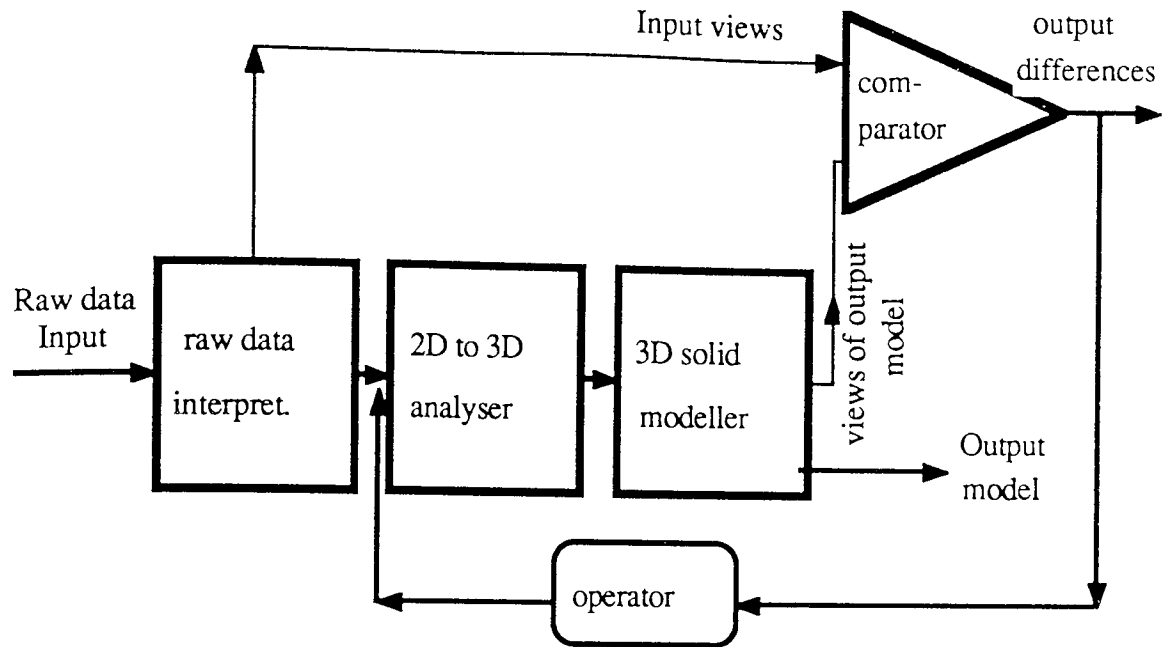


Figure 4.23 Flowchart of the process

Each step is described in the following sections

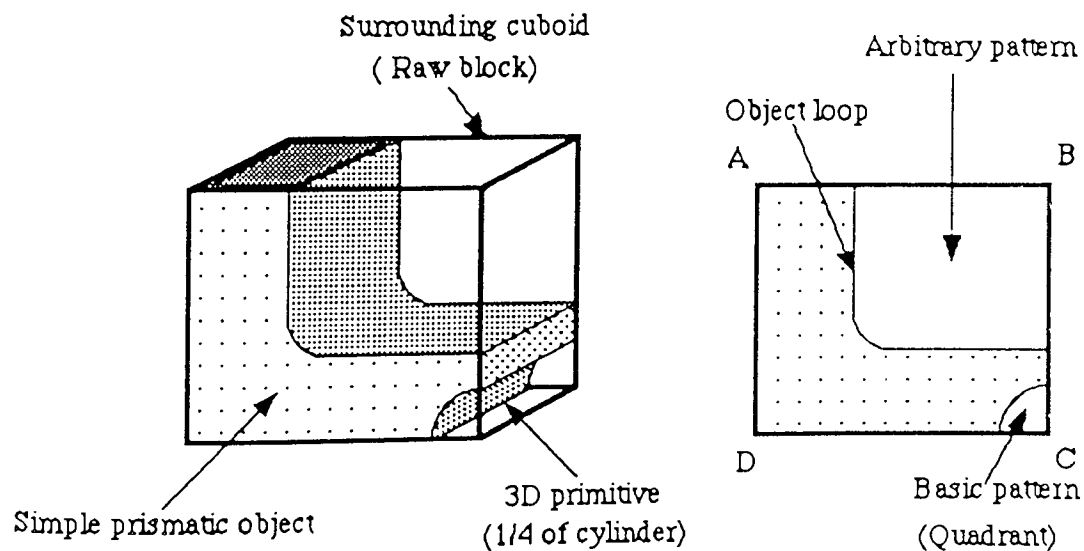
4.3.2.1 Raw data interpretation

In the raw data interpretation step input data is checked for incomplete or false information such as dangling edge. Also some operations are performed on input data to transform them into the structured format for use in the next stage.

4.3.2.2 Data analysis

In this process, each 2D view is examined to locate and identify all the elementary pattern or signatures in order to make the 3D primitives. The boolean operations are also defined for these primitives. Some terms has been defined as follows :

- * The *surrounding cuboid* is the 3D raw block with the maximum dimension of x,y and z of the object.
- * The *surrounding rectangle* is the closed loop comprising an orthographic projection of the surrounding cuboid.
- * The *object loop* is the closed loop that is an orthographic projection of the solid object.
- * A *primitive loop* is a closed loop formed between the surrounding rectangle and the object loop.
- * A *basic pattern* is a primitive loop which can be identified as an elementary two-dimensional shape like rectangle, triangle and so on.
- * An *arbitrary pattern* is a primitive loop which cannot be identified as an elementary 2D shape and requires more processing.
- * A *parent primitive loop* is an arbitrary loop which has been identified as an arbitrary pattern. It is then directly decomposed into a number of primitive loops or children loops.



ABCD = Surrounding rectangle

Figure 4.24 Data analysis terminology

The main part, which is common to prismatic, ortho-prismatic and general three-dimensional objects, consists of extracting object loops from a view and identifying 2D patterns and related boolean operations required for constructing the loop. The class of object is determined by considering the shape and type of the loops in each view. For prismatic objects there will exist one view consisting of either one closed loop (the perimeter) or several closed but disjoint loops. The inside disjoint loops represent through holes and two other views must comprise a number of connected rectangular loops.

A series of tests are carried out to check the position of all object loop nodes in relation to the nodes and sides of the surrounding rectangle and locating the primitive loops formed by the intersection of the surrounding rectangle and the object loop (figure 4.24). Then each of the primitives is checked against a basic pattern or arbitrary pattern. In the case of an arbitrary pattern it is decomposed into basic patterns or further arbitrary patterns (figures 4.22 and 4.25). During the decomposition of an arbitrary pattern the boolean tree is produced. The algorithm uses the "depth-first search" method to decompose an arbitrary pattern.

4.3.2.3 Three-dimensional modelling

Output produced by the last stage, which have been saved as a file, are used as input to a three-dimensional solid modeller. The data stored on file contains two types of information: one is about primitives themselves, such as type, geometry, location of centre, etc.. , and the other is concerned about the boolean operations that must be applied on primitives to build the final object (figure 4.26). Therefore the solid modeller's task is to perform this process and produce the result object, which here is called the first approximation model. The solid modeller used here is BOXER, produced by Pafec Ltd.

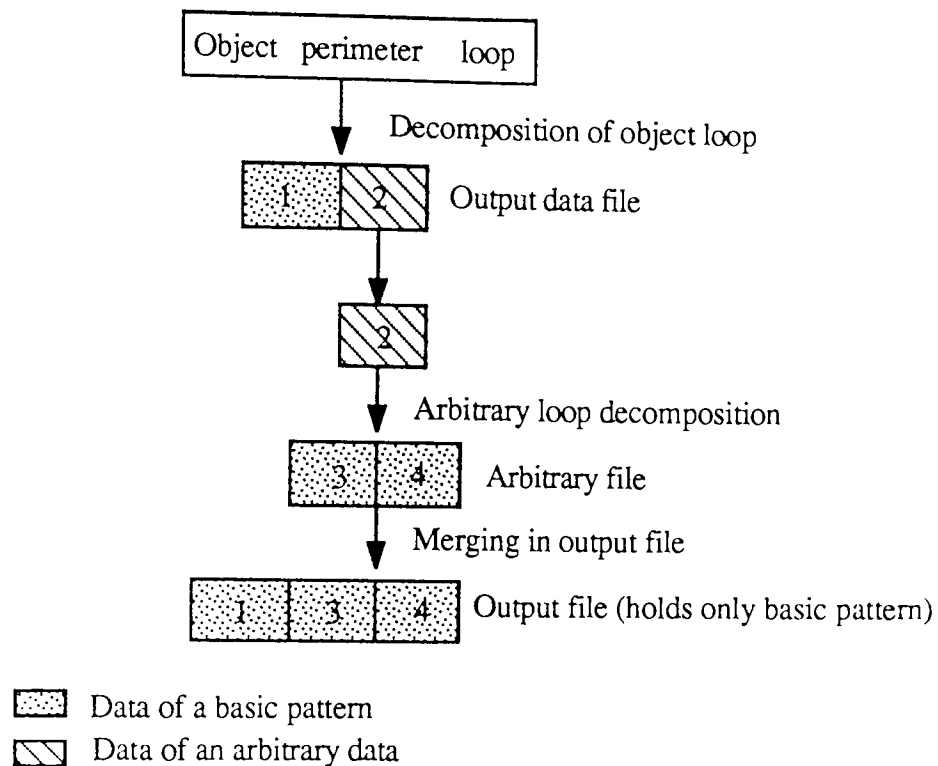


Figure 4.25 Decomposition and merging stages in the analysis process

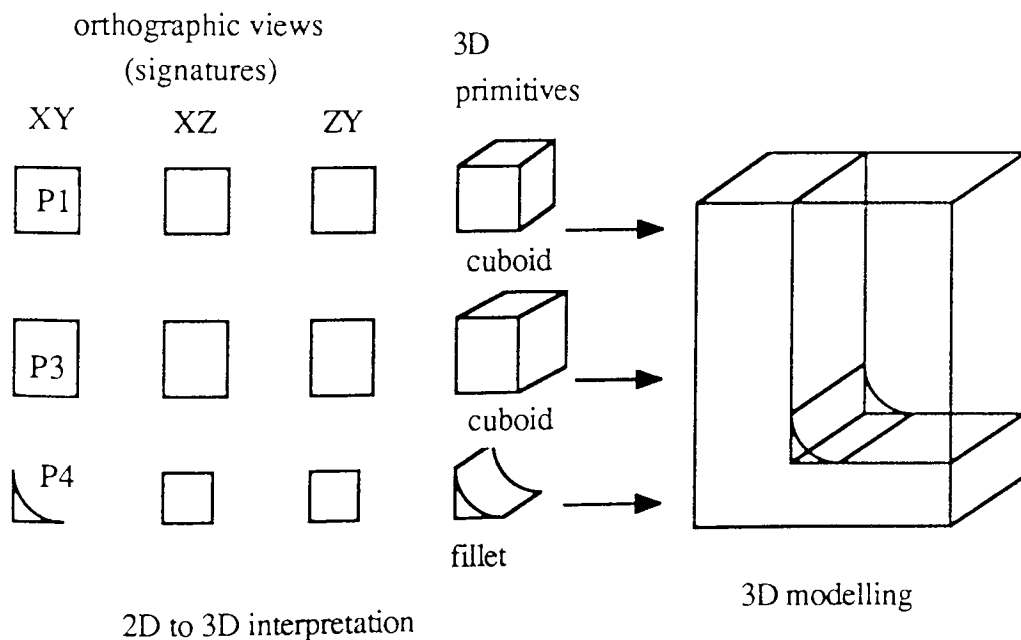


Figure 4.26 Primitive identification and 3D modelling steps

4.3.2.4 Output verification

There are two inputs to this part. One is a set of orthographic views obtained from the raw data interpreter section and the other is the set of orthographic projections output from the three-dimensional solid modeller (the first approximation model). These sets are compared: if they are not the same, then it means that the first approximation model does not represent the real object model and must therefore, be modified in some way.

4.3.3 Deficiencies

Although this recent work presents a new method based on CSG concepts and has some advantages over other works, still there are some deficiencies which must be corrected. Some of these obstacle points are :

- The model produced by this method, does not always show the real object, which is why it is called the first approximation model.
- The model works for some class of object, for instance prismatic and orthoprismatic objects.
- Only boundary of engineering drawings are considered, therefore the internal details are not processed. This causes much information such as dashed lines to be missed.
- Only limited types of primitives (uniform-thickness) are used to build the object, therefore objects that may have been composed of primitives like cone, sphere, and so on cannot be generated properly.
- Since the method uses cutting out of a raw block, the tree structure generated for boolean operations is not necessarily optimum for some objects having a combination of primitives like sphere, cone and torus.
- The proposed feedback algorithm for modification of the output CSG model is very time consuming , for it is using a solid modeller.

Chapter Five

ENHANCEMENT OF FIRST APPROXIMATION MODEL

Chapter Five

ENHANCEMENT OF FIRST APPROXIMATION MODEL

5.1 Introduction

The previous chapter reviewed many works done in relation to transforming two-dimensional orthographic views into three-dimensional solid objects. The method used by Kargas, Cooley and Richards [1988] has been found to be more interesting because of the advantages mentioned there. By the algorithm used in that project, a 3D solid model called the "first approximation model" is generated for a given set of objects. Some deficiencies of the model were, however, also found. To overcome the problems mentioned, and to improve the model toward the real object model, a new algorithm is proposed. This will now be discussed.

5.2 Mathematical Description

In order to enhance the first approximation model and make it closer to the real solid object, a new approach is chosen. The basic idea is to break down an object into a set of first approximation models. Before going any further, however, some terms and notations are defined.

Definitions :

O = The real object

f = The first approximation model function

$O' = f(O)$ = The first approximation model of object O

O_{ij} = Sub-object j at level i

O_{xy} = A prismatic object produced by the extrusion of view xy in Z direction, with the height of " $Z_{\max} - Z_{\min}$ "

O_{xz} = A prismatic object produced by the extrusion of view xz in Y direction, with the height of " $Y_{\max} - Y_{\min}$ "

O_{zy} = A prismatic object produced by the extrusion of view zy in X direction, with the height of " $X_{\max} - X_{\min}$ "

\cup = Union operation

\cap = Intersection operation

$-$ = Subtract operation

Then the first approximation model can be expressed as :

$$O' = f(O) = O_{xy} \cap O_{xz} \cap O_{zy} \quad (5-1)$$

It is obvious that any object can be defined by its first approximation model and a set of some sub-objects. Therefore at the first level of decomposition :

$$O = f(O) - O_{11} - O_{12} - \dots - O_{1n} \quad (5-2)$$

Sub-objects are defined as a set of separate objects or the connected volume trapped between the object and its first approximation model. Note that two separate objects may have only common edges or vertices (not faces).

The relation (5-2) has been represented graphically in figure 5.1, which is called the decomposition tree. At the second stage each sub-object O_{1i} can again be expressed in terms of its first approximation model and a set of sub-objects at level 2. Alternatively

relation (5-2) for sub-object O_{1i} can be written as :

$$O_{1i} = f(O_{1i}) - O_{2i_1} - O_{2i_2} - \dots - O_{2i_m} \quad (5-3)$$

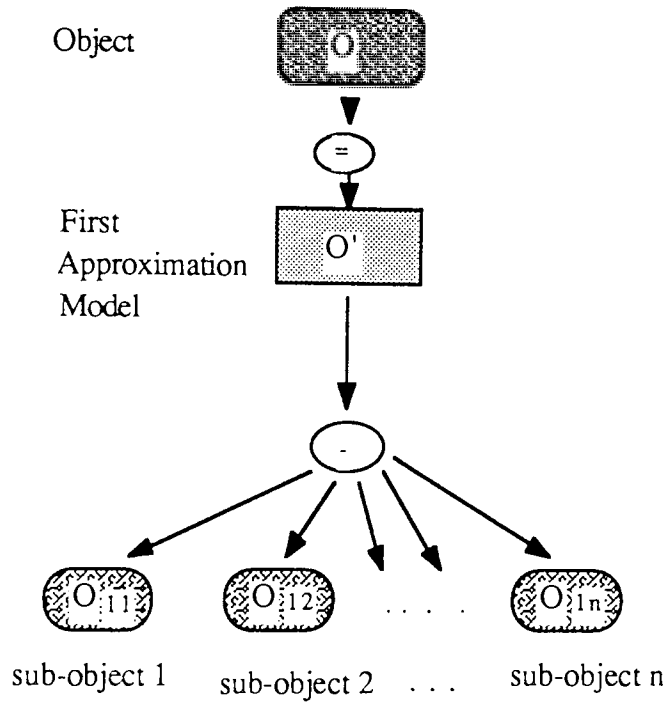


Figure 5.1 The decomposition tree

It is deduced that the same relation (5-2) can be applied for each sub-object recursively as well as for the main object. Therefore the process may be continued until the generated sub-objects at the leaves of the decomposition tree become either primitive, prismatic or orthoprismatic objects, or their volume/dimension is less than some predefined values. In the first case the output model will be an exact representation of the object; the latter case will demonstrate the approximation model of the object. An example of decomposing an object is shown in figure 5.2. This method is similar to the expansion of a function as a polynomial.

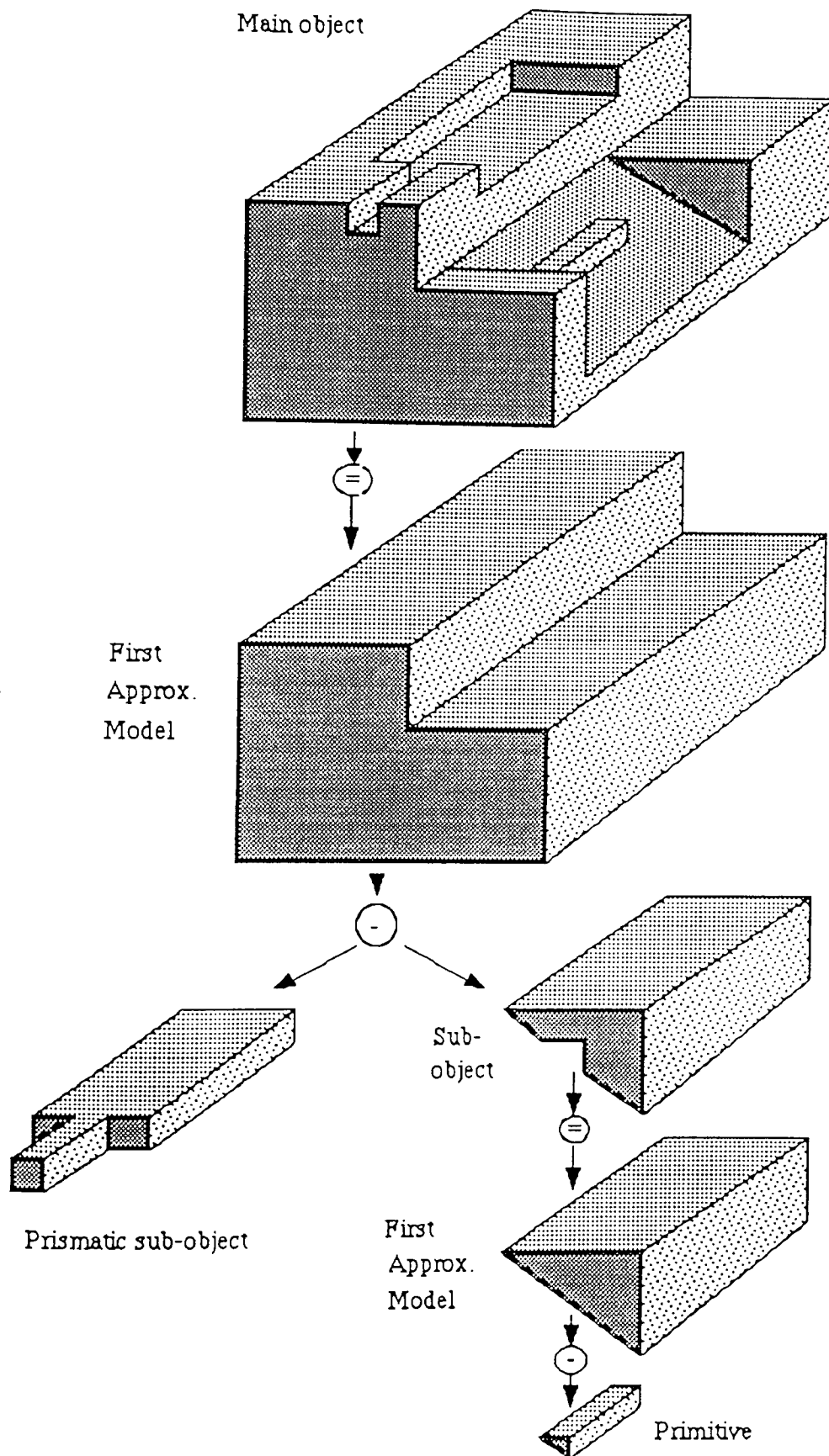


Figure 5.2 An example of decomposition tree for an object

5.3 Full Details of Method used

5.3.1 Scope

The aim of this project is to accept a set of orthographic views as input and generate an output solid model which represents an object in form of CSG model. The output model must be as close as possible to the real object and be in the form of an input command file required by a solid modeller to build the 3D model. This model must also be unique, complete and unambiguous. The input file consists of a set of 2D straight edges, arcs and a set of 2D vertices. Edges can be in solid or dashed form. The types of primitives are extended to nonprismatic primitives such as cone and sphere. Reducing the number of feedback loops, which results in the minimum use of the solid modeller, is also another target. Also, in some ambiguous cases, using other information such as dashed lines is considered. Although theoretically there is no restriction on the type of input drawings, there are some practical limitations which will be discussed later. In each step different outputs are displayed on the graphic screen.

5.3.2 Description

Since the first approximation model is accepted as a solid, using it as a core element towards producing the output model guarantees that the model will also be a solid. Therefore, utilising the basic idea explained in 5.2 as the main body of the algorithm, the problem then reduces to specifying sets of 2D edges and vertices from the orthographic views belonging to different sub-objects. The first approximation model implemented so far is modified and used as a tool throughout the project. To find the sub-objects trapped between the real object and the first approximation model, two kinds of wireframes are found: one for the object (figure 5.3) and the other for the first approximation model, which is also shown in bold in figure 5.3.

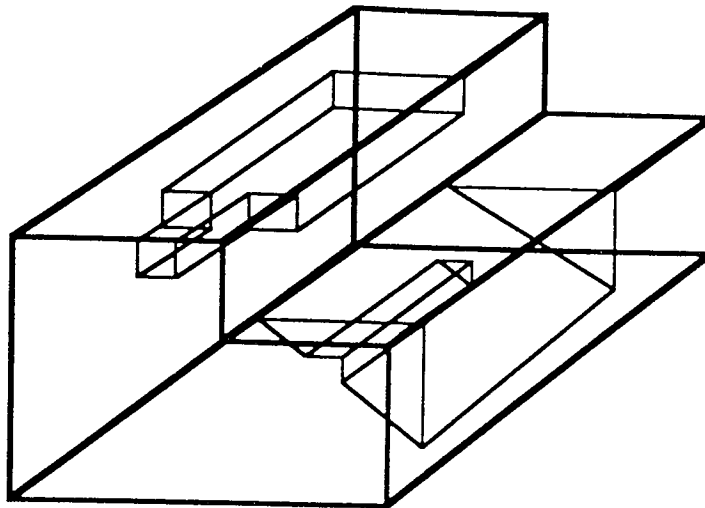


Figure 5.3 Wireframe and first approximation wireframe in bold

By comparing the two wireframe models, it is obvious that the 3D edges which are not lying on the boundary surface of the first approximation model belong to the internal sub-objects (figure 5.5a). Of course, some of these edges may be pseudo ones, as will be discussed in a later section. Certainly, the 2D projection of these 3D elements, combined with some part of the 2D boundary edges of figure 5.4, will represent projections of the sub-objects. This is depicted in a separate picture of figure 5.5b. The method of selection of boundary parts and other related materials will also be explained later. So, after separation and grouping of 2D edges as a set of orthographic views for another object, they are fed back to the program as a new input to generate the first approximation model of the sub-object. This process is repeated recursively until a complete or acceptable approximation model is achieved.

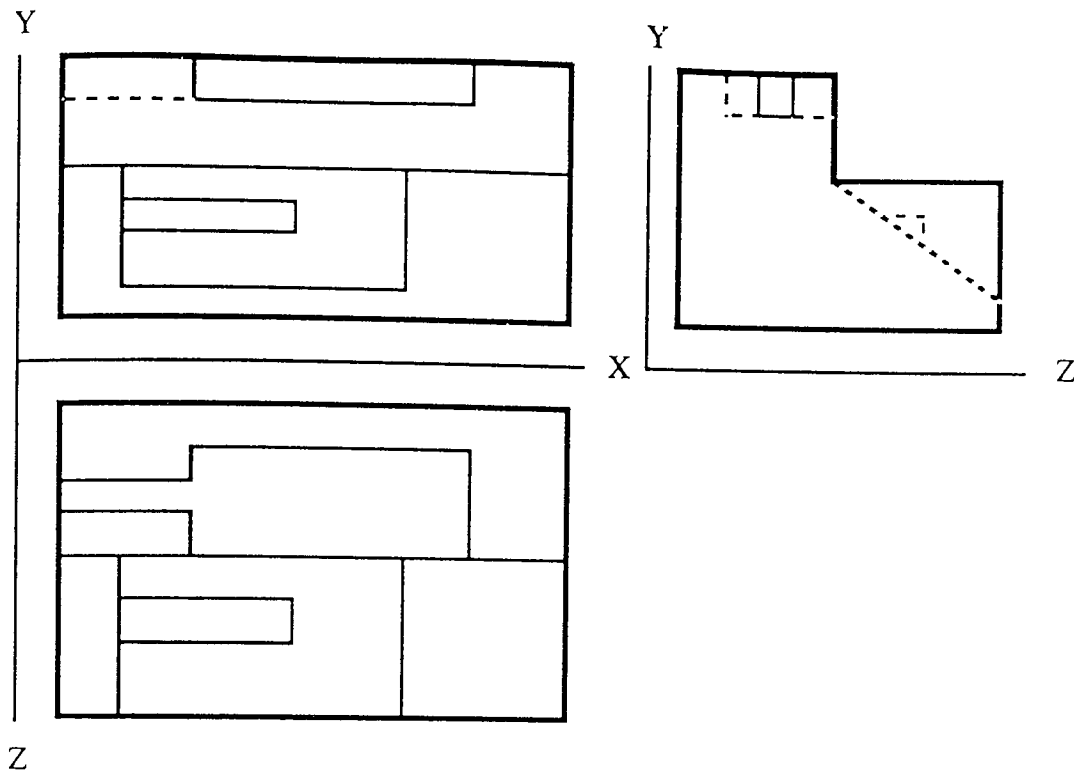


Figure 5.4 Orthographic views

In order to be able to break down the object in the proper sequence and disassemble sub-objects by appropriate boolean operation, in each step the algorithm keeps track of the production of first approximation models for the sub-objects and gradually builds the tree structure representing the decomposition of the object. An overview of the whole process, given in figure 1.5, is repeated in figure 5.6. The method involves the following steps which will be explained in individual sections:

- Find wireframe model of input data
- Find the first approximation wireframe model
- 3D comparison of wireframe models
- 3D to 2D transformation
- Separation and finding of 2D sub-object views
- Performing decomposition tree

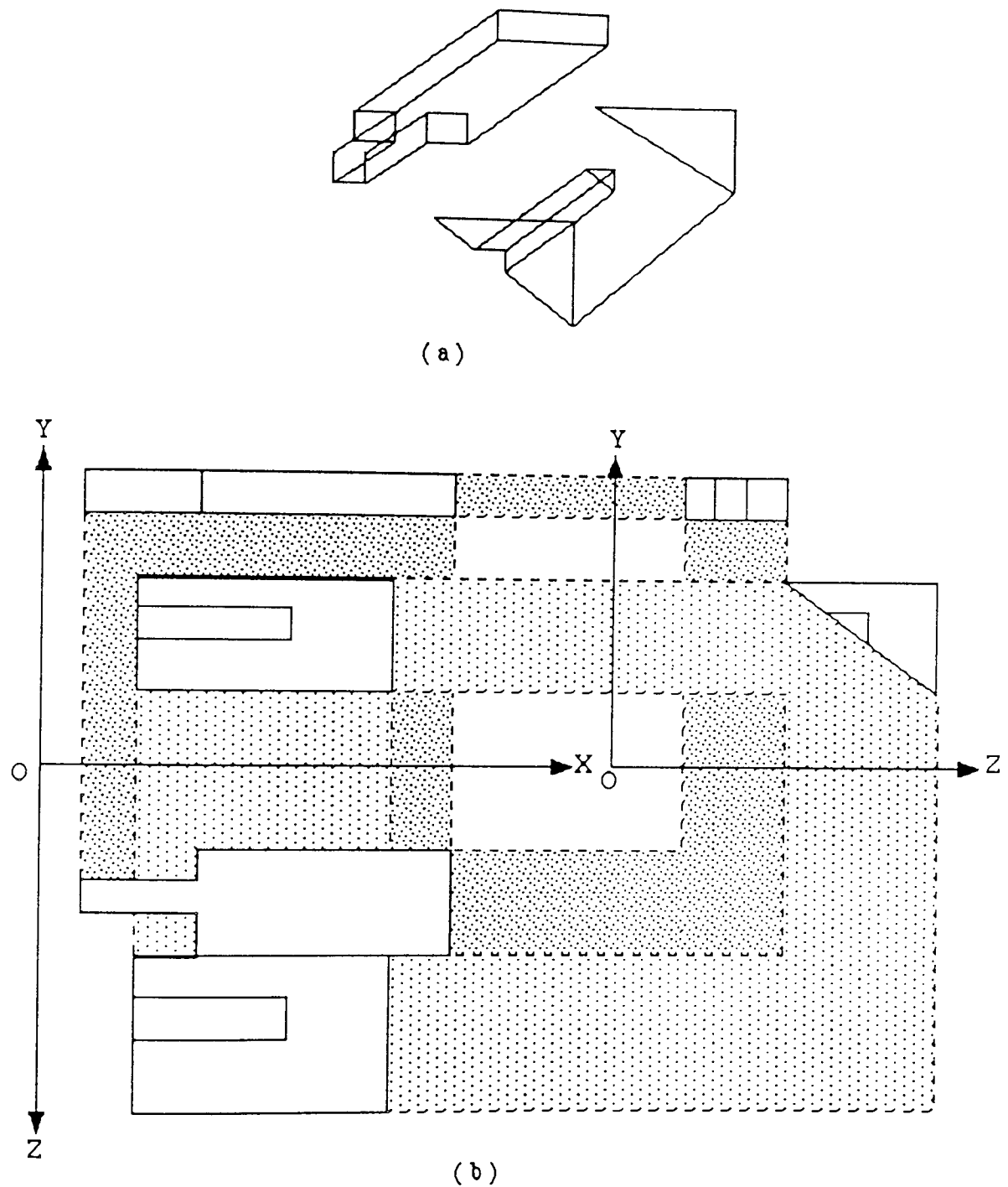


Figure 5.5 (a): 3D edges not on FAM boundary (b) : 2D picture of (a) completed with 2D boundary (in bold)

To avoid using the solid modeller to generate a wireframe model from the first approximation model, a simple method is developed to achieve this by using the boundaries of the 2D input views. It should be mentioned that, even for simple

applications, calling a solid modeller is time consuming, since it initialises graphics and demands another set of procedures.

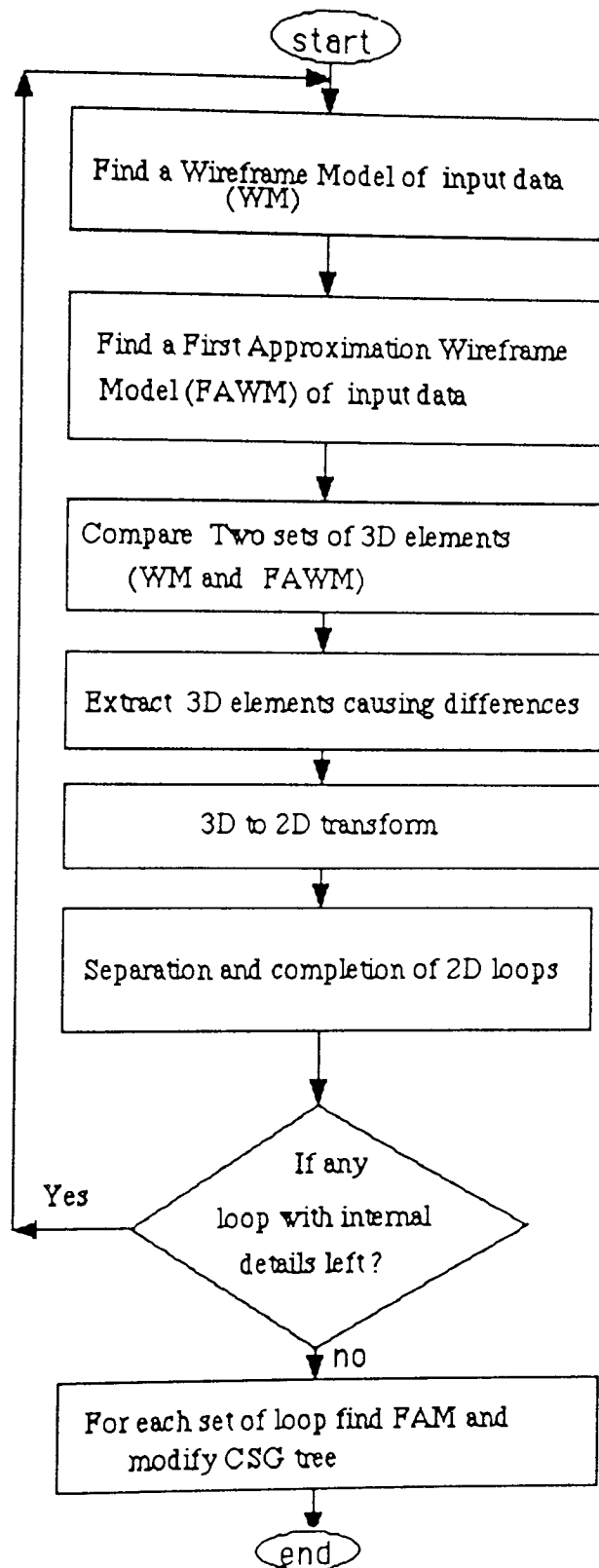


Figure 5.6 Overview of the system

Another issue that should be explained here to clarify the problem is that, although the technique used here is using a wireframe for comparison and separation of 3D elements, it does not mean that the method is totally based on it. The first approximation wireframe model is actually a simple representation of a solid prismatic object and, because of the nature of the first approximation model, it is not ambiguous. Since all of the elements, produced in the leaves of the decomposition tree are solid objects, therefore using the regularised boolean algebra to build the object from bottom to top, will result in a solid object.

5.4 Implementation of the Method

To implement the method proposed in figure 5.1, a program was developed in FORTRAN code on an Apollo workstation. The block diagram of the process is summarised in figure 5.6 and the modules which have been developed and used, together with their interconnections, are shown in figure 5.7, in which the boxes shown in bold, are routines that do the main part of the algorithm. The sequence of call to main modules is controlled by the MAIN module, for which the overall flowchart is presented in figure 5.8.

Brief descriptions of each module of figure 5.7 are given below:

MAIN : It does all parameter initialisation, option definition and monitoring the right sequence execution of the program by calling to appropriate subroutines. It also sets up the procedure for recursive running of the program for sub-objects. At the end it assembles a final file for the solid modeller.

TITLE : Prints title on the output.

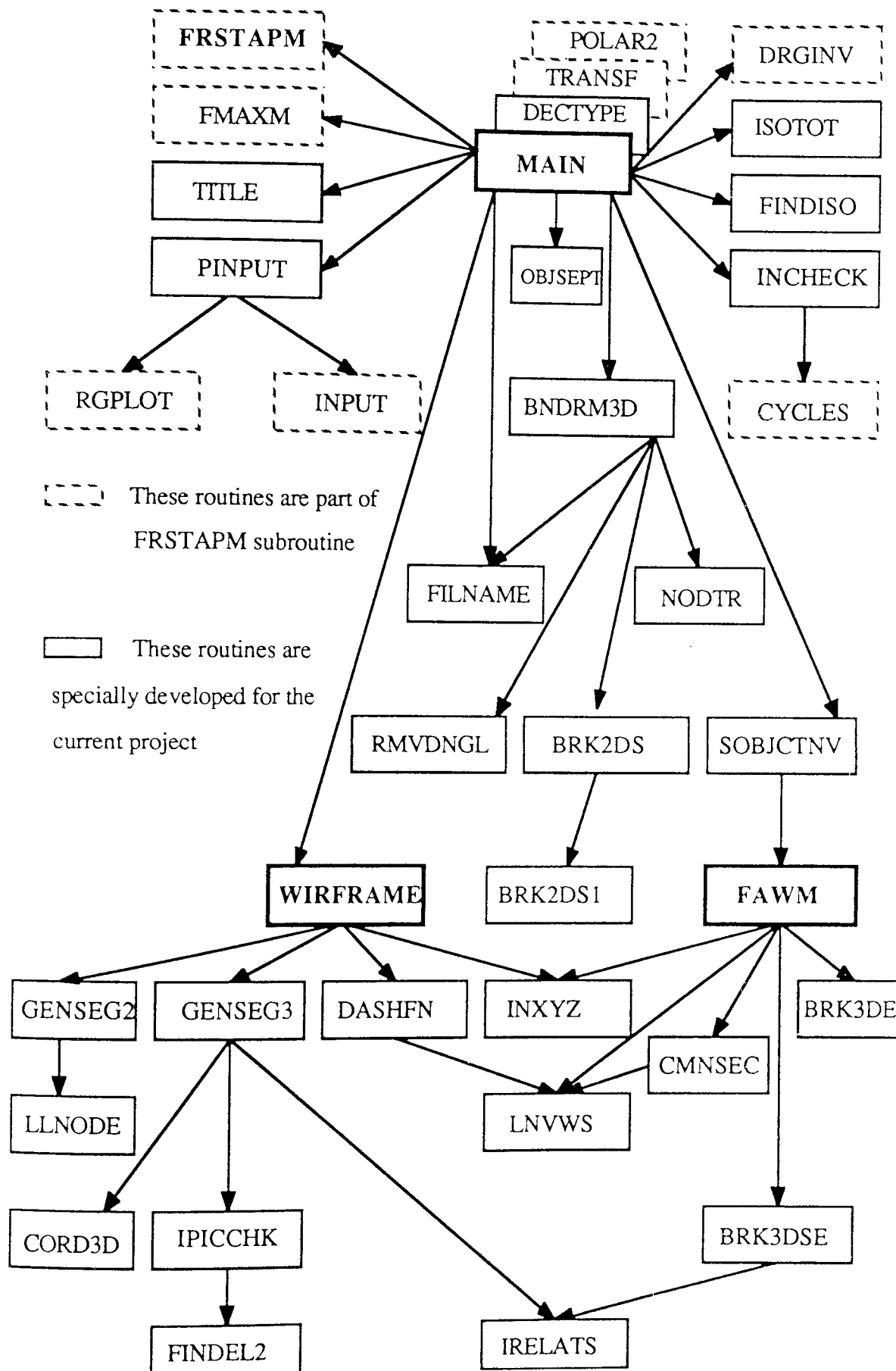


Figure 5.7 Structure of modules of the software implementation

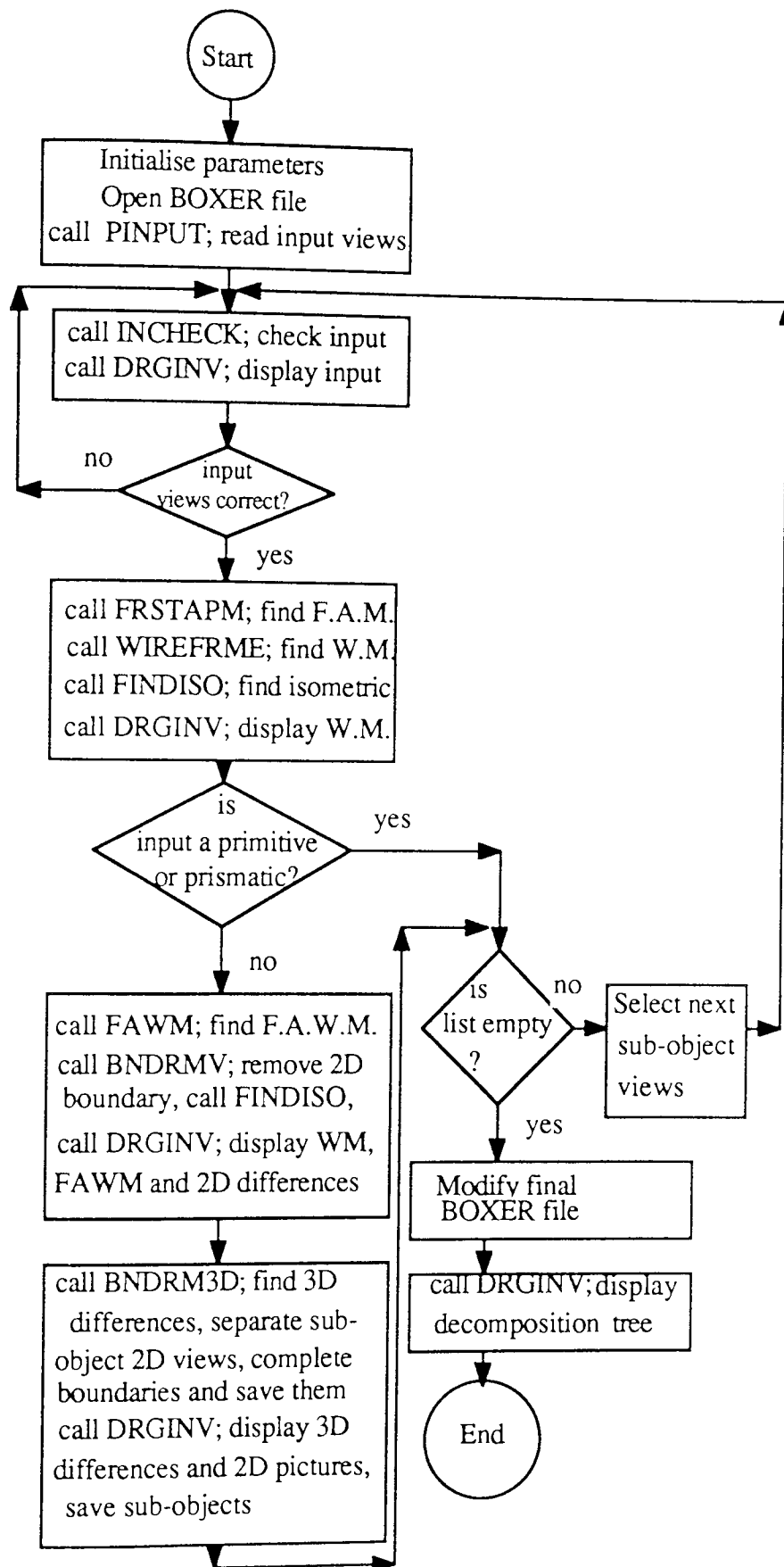


Figure 5.8 Summary flowchart of the MAIN program

PINPUT : Checks for input files: if the input files have not been created before then it creates them and calls INPUT to read data. It also verifies if options are to be input or not.

INCHECK : This routine checks for correctness of the input views data. It breaks all arcs at their extreme points and also finds the boundary edges of each view and marks them for future use.

WIRFRAME : This performs all initialisations relating to the 3D wireframe models. If it is requested, it prints all necessary data during each process. It also finds the 3D node list and generates the 3D wireframe model.

DASHFN : This subroutine checks that if a 3D edge is produced by a 2D dashed type edge and it is the closest element in the direction of sight to the observer, then it is cancelled.

FAWM : This is the main subroutine responsible for producing the first approximation wireframe model by subsequent calling to other routines.

LVWS : This calculates the intersection of a horizontal or vertical line with the boundary of a 2D view.

CMNSEC : This finds sections of a straight line normal to a view which lies within the first approximation model.

BRK3DE : This breaks a 3D edge produced for first approximation model if it passes a 3D node.

BRK3DSE : This breaks a 3D edge of the wireframe model if a new 3D node created

by the first approximation wireframe model is located on it.

INXYZ : This function checks if a given 3D coordinate is in the node list, in which case it returns the node number, otherwise this coordinate is entered in the table and a new node number is assigned.

GENSEG2 : This generates all acceptable combinations of 2D collinear edges and adds them to the end of the edge list.

LLNODE : This searches for the lowest left hand node.

CORD3D : This subroutine finds all 3D nodes that have a projection with specified coordinates in a view.

GENSEG3 : This produces all 3D edges so that its three projections in different views are consistent.

IPICCHK : This checks if a given 3D edge with a specified projection type in a view has any suitable projection in other views.

FINDISO : This calculates the isometric view of a 3D wireframe model for displaying on a graphic screen.

ISOTOT : This determines the coordinates of the start point for drawing the isometric picture of an object or sub-object on a graphic display in the procedure for drawing the decomposition tree.

SOBJCTNV : This routine prepares the count vectors to represent the number of 3D edges that produce a specific 2D projection. Then it activates the FAWM subroutine for

producing the first approximation wireframe model.

BNDRM3D : This finds the 3D edge difference between the wireframe and first approximation wireframe models.

OBJSEPT : This separates each set of 3D edges belonging to different sub-objects and finds their 2D views.

BRK2DS : When it is required, this breaks 2D edges into smaller sections by calling the **BRK2DS1** subroutine.

NODTR : This function gives a node number of the projection of a given 3D node in an identified view.

RMVDNGL : This looks for dangle edges and then removes them. Also it renumbers the remaining 3D edges.

FILNAME : This generates a meaningful name for saving 2D pictures of each sub-object.

5.4.1 Modification to the Current First Approximation Model Program

According to the method proposed in figure 5.1, the main item for implementation of the algorithm is the production of the first approximation model. For this purpose a subroutine is needed which, when called, operates on the boundaries of a given set of three 2D input views and generates a file containing the set of commands required for a solid modeller to build the first approximation model. In this regard, the program developed by Kargas, Cooley and Richards [1988] is used and then adapted to fit into

the necessary specification.

5.4.1.1 Changing the program into subroutine

In order to be able to use the program not only as a simple subroutine, but also for carrying out other modifications, such as increasing the number of primitives, the program was studied and a set of the required modules (figure 5.9) selected and linked together. The main routine was changed into a subroutine with suitable parameters for transferring vital information for control purposes. Input views and the output model are transferred as data files. Other parameters, vectors and tables which are used by this project are kept in COMMON BLOCKS. The call statement takes the form of :

```
CALL FRSTAPM(ICLAPP,IERROR)
```

The output parameters are:

IERROR = 0, indicates that the first approximation model has been generated successfully.

IERROR > 0, specifies the type of error occurred during the process.

ICLAPP = 0, the object is prismatic, therefore the output is the exact model for the object.

ICLAPP = 1, the object is not a prismatic object.

ICLASS = 0 (defined in common blocks), means that the object is a primitive.

The output model is written to a file called "BOXER.DAT" in the form of commands acceptable to the BOXER program (a solid modeller from Pafec ltd.). Input views are passed to this subroutine through structured files by their names "XY", "XZ" and "ZY" for each view respectively.

A short summary of each module used in the FRSTAPM is presented here:

ANGLE : finds the angle between two straight lines.

ANGLES : calculates the angles of all outgoing edges relative to an inward edge of a node.

BLOCK : writes the commands necessary to produce a block with specified geometry and location into the BOXER.DAT file.

BTREE1, BTREE2 : are used in the creation of the boolean tree, keeping track of each generated primitive.

CFLAG : assigns a flag to each node of a given loop, according to its position with reference to the nodes and sides of the surrounding rectangle.

COLIN : transforms any pair of collinear edges into one segment.

COLINA : assembles a group of collinear edges into one edge.

CYCLES : finds all the cycles existing in a view.

CYL1,CYL2 : write the commands necessary to produce a cylinder with specified geometry and location into the BOXER.DAT file.

DRAW : displays a view on graphic screen.

DRAWV : sets up different windows for any set of views.

DRGINV : initialises graphic section.

EXTREM : finds the nodes which have one of the extremum X and Y coordinate values.

FILLET : writes the commands necessary to produce a fillet with specified geometry and location into the BOXER.DAT file.

FMAXM : finds maximum and minimum values of a given vector.

INPUT : reads input data from keyboard for each view and stores them in corresponding files.

LINK : finds all nodes which are linked to a specified node through an edge.

LOCAT : finds and stores the location of each pattern.

FRSTAPM : is the main body of FRSTAPM subroutine.

MARKER : marks each traversed edge in CYCLES routine.

MERGEF : merges direct access file into the MAINDATA file.

OBNAME : allocates a name to a 3D primitive.

PATT1, PATT2 : determine the type of a loop, whether it is rectangle, sector, right-triangle, fillet or an arbitrary pattern.

PERPD : computes the coordinates (u,v) of the point P, at which a perpendicular from

a point D intersects a line.

PLOOPS : the main routine that processes a loop.

POLAR2 : converts rectangular coordinates (x,y) into polar system (R,A).

PRIMID : identifies a primitive from its patterns.

PROCLP : processes a loop to determine the pattern loop types.

RELATE : finds the relationship between different cycles (i.e. whether they are disjoint or have a common edge).

RNUMR : reorders the rectangle number and coordinates.

SETREC : sets the rectangle node coordinates, depending on the loop direction.

TEST3 : checks if a loop is a rectangle.

TESTAR : searches in the MAINDATA file for loops which have been tagged as arbitrary patterns. If such a loop is found, then its corresponding geometric and topological data are stored in a random access file called ADATA.

TOPUP : updates the topology and geometry data.

TRANSF : transfers data from file to memory or *vice versa*.

WEDGE : writes the commands necessary to produce a wedge with specified geometry and location into the BOXER.DAT file.

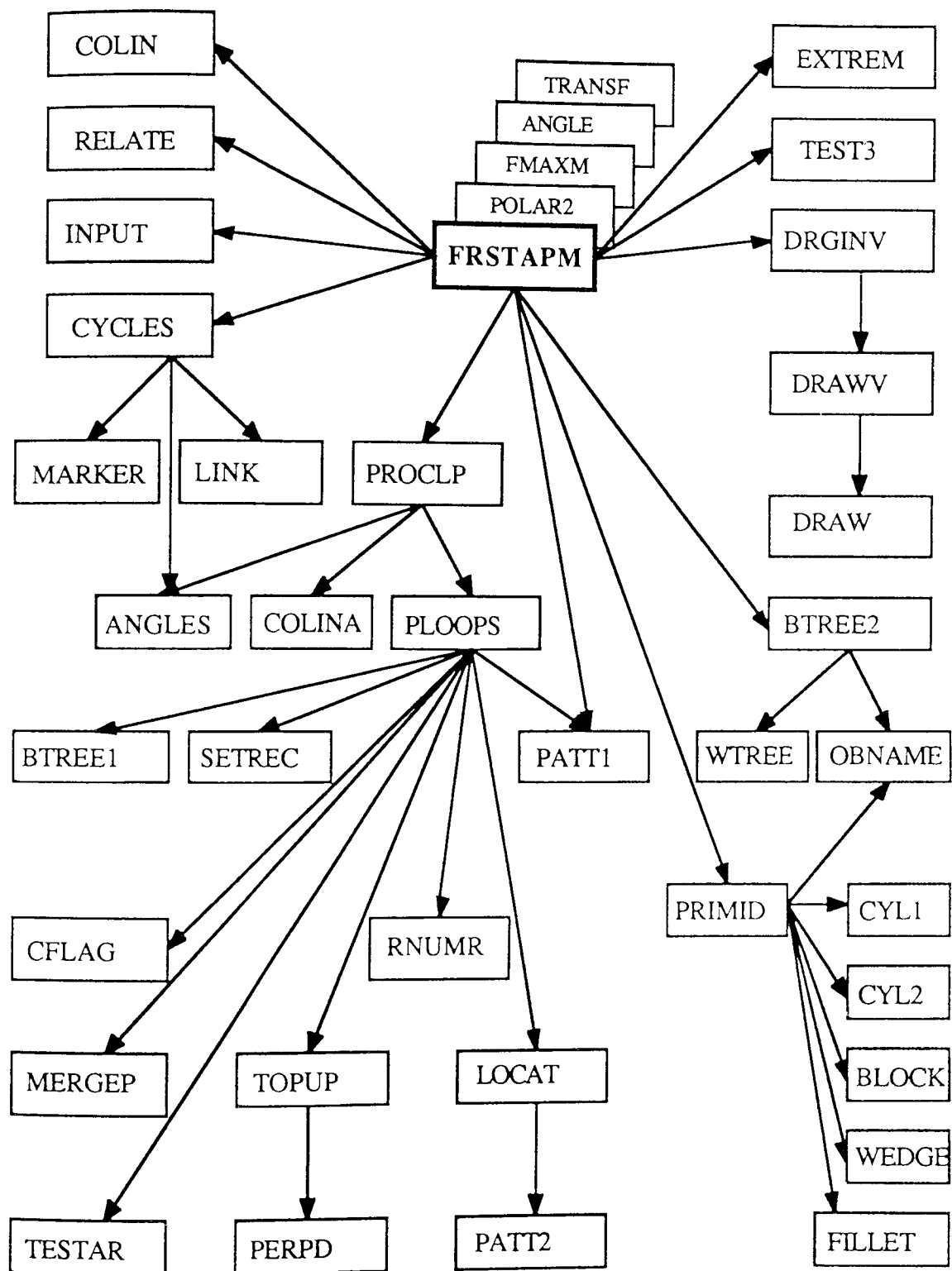


Figure 5.9 Modules used in the FRSTAPM subroutine

WTREE : reads the data generated by BTREE1 and BTREE2 to write the boolean tree into the "BOXER.DAT" file.

5.4.1.2 Modifying edge attributes in data structure

To increase the power of the program and to help in decision making in some ambiguous cases, it is better to use all the available information in an engineering drawing. One of these items is dashed line information that shows whether an edge is hidden or not. Another item which is useful for further development of the program is the different edge types in views like an ellipse rather than having only straight lines and circular arcs.

Since the only attribute available has already been defined, in order to preserve the form of the data structure at this stage, it was decided to pack all of these attributes in the same field. To read each individual item, a short subroutine with the following arguments was written to unpack them :

```
CALL DECTYPE(I,ITYPE,IDIR,INDEX,IDASH)
```

in which:

I = (input) the packed edge type

ITYPE = (output) edge type, 0 = straight line

1 = circular

2 = ellipse

3 = parabola

4 = hyperbola

IDIR = (output) direction of edge, 1 = clockwise

-1 = counter clockwise

INDEX = (output) index for non straight edges

IDASH = (output) 0 = solid edges

1 = dashed edges

The codes used for packing are selected as below :

name	solid type	dashed type
-----	-----	-----
Line	0	10000
Circular arc	+/- Index	+/(10000+Index)
Ellipse	+/(1000+Index)	+/(10000+1000+Index)
Parabola	+/(2000+Index)	+/(10000+2000+Index)
Hyperbola	+/(3000+Index)	+/(10000+3000+Index)
+ for clockwise direction		
- for anti-clockwise direction		

5.4.1.3 Modification in CYCLES routine

CYCLES is the routine that finds all of the cycles existing in a view. It starts with a node having minimum X and Y values (the lowest left most node) and selects the right most edge and traverses it to the next node. It uses the same rule for the next node and again finds the right most edge and continues until it reaches the start node. By this method the outer loop (boundary) is detected. During the process, if an edge is traversed from its start node to end node then it is tagged +1, and if it is traversed in reverse order it is marked as -1. To find inner cycles, this process is repeated in the reverse direction until all edges have been passed twice and in different directions. The

selection of the next rightmost edge is decided on their angle. Unfortunately for the case where the inward edge is a circular arc, as depicted in figure 5.10, because of ambiguity in calculation of angles, the wrong decision is made by the program at point 2 and instead of node 6 node 3 is chosen for next step. This problem is solved by a little differentiating between the angle of the tangent arc and the straight line at point 2.

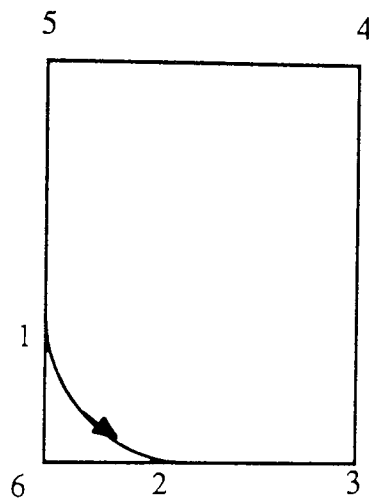


Figure 5.10 An example of cycles finding

The FORTRAN code is modified from :

$APREV = APrev - 90 * SI * IDIR$ to

$APREV = APrev - (1 / (1000 * RS) + 90) * SI * IDIR$

$APREV$ = the angle of edge relative to previous node

SI = traversing direction

$IDIR$ = arc direction

RS = radius of arc

Another modification to this subroutine is the addition of input parameters used to ask

the routine to find only the perimeter loop, without doing the rest of its function.

5.4.1.4 Modifying TRANSF

Originally, the TRANSF routine had been used to read data from files into topology and geometry arrays in the memory, and other tasks in this relation such as storing data from memory to disk are done repeatedly by codes. Therefore, to reduce the size of the program code, the subroutine is changed to the following form and used as many times as required throughout the program :

CALL TRANSF(INUNIT,OUTUNIT)

This call statement restores from the input file specified by INUNIT into appropriate arrays in the memory, and then from memory into the output file indicated by OUTUNIT. If any of these arguments is zero that function is ignored.

5.4.1.5 Modification in relation with primitives

Increasing number of primitives

As was explained, because of the nature of the method used in finding the first approximation model, each view is processed to find the base signature of the primitive separately and two other signatures are assumed to be rectangle. In this situation, primitives like sphere and cone are not used in the boolean tree. Primitives like sphere, cone, hemisphere and sectors are therefore added to the set of primitives.

Changes in PATT1 and PATT2

These subroutines are responsible for identifying patterns. In order to add new

primitives like cone, sector and others these routines are changed so that they are able to detect relevant patterns such as an equilateral triangle.

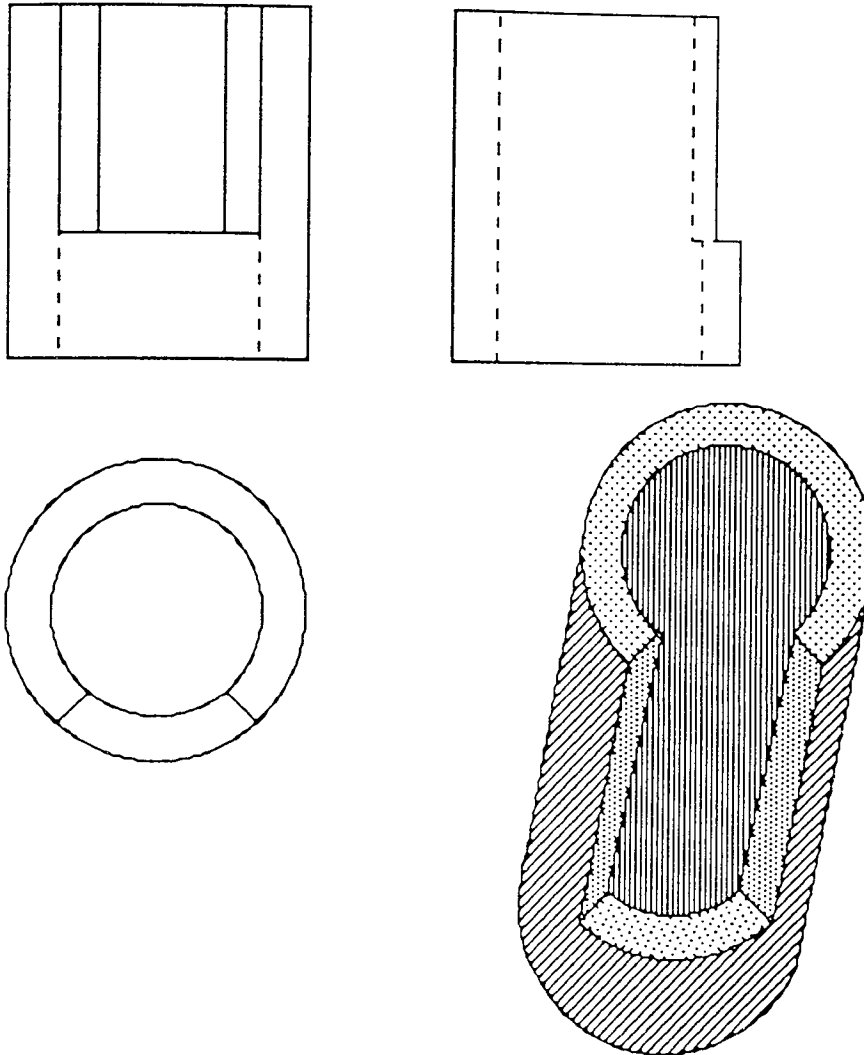


Figure 5.11 An example of circular pattern

Also in the original routines only a complete circle or exact four connected quarter circles is recognised as a circle. Thus, circles like the one shown in figure 5.11 cannot be properly identified. This problem has been cleared up in the updated version by the author.

Selection of raw prism

In the old method, it is always assumed that every part is cut out from a raw block (cuboid). In many cases starting with such a form of block would require many cuts correspondingly large number of levels in the associated boolean tree. Hence, in the modified version of the program, if the root is a primitive it is left unchanged. For example, in figure 5.11 the object can be cut from a cylindrical rod.

Initial boundary testing

Considering the example of figure 5.12, since the object has internal details, the old method of finding the first approximation model processes three views and the object is recognised as non prismatic. In this case, three views are processed separately, then primitives are generated for them and they are passed to the solid modeller for calculation of the intersection between these three objects, which takes time. In the revised version of the first approximation model subroutine, on the first pass the boundaries of views are tested for any primitive: if successful then it is regarded as a primitive, and by this means processing time is saved.

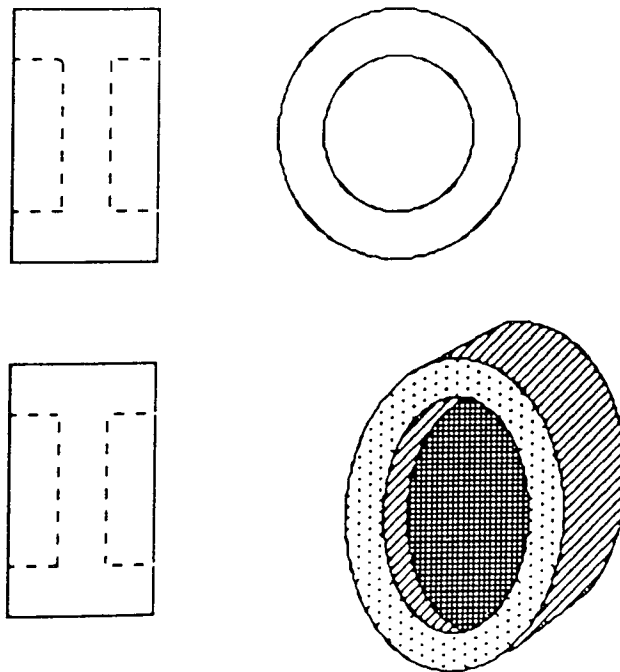


Figure 5.12 Checking of the boundary for primitives

5.5 Discussion

In this chapter a mathematical model based on the expansion of an object in terms of some simpler objects is presented. At the first level the object is assumed to be its first approximation model, from which a set of sub-objects is taken. An algorithm is proposed and implemented for finding sub-objects, by comparing two wireframe models: the object's wireframe and its first approximation wireframe. A new fast method has been developed for calculation of the first approximation wireframe model. A subset of modules taken from earlier work were selected and modified to form a subroutine to generate the first approximation model.

Many issues have been addressed in this chapter. To increase the accuracy of the model, each sub-object is again treated as an individual object and expanded to a level further in terms of its own first approximation model, and another set of sub-objects. By these means a set of volume is subtracted from the first approximation model, while in the second level some volumes are added. Subtraction and addition of objects then alternates at successive levels. The process may be continued until the volume of sub-objects are less than a specified value or they are primitives or prismatic objects.

Another issue is the speed of the process which mainly depends on the amount of backtracking and the number of produced 3D edges. Since the algorithm is designed so that it either prevents ghost edges being produced right at the beginning (using dashed line information and edge types) or reducing its effect on sub-objects (by comparing two identically produced wireframes). Therefore, both the amount of backtracking and the number of 3D edges in wireframe have been reduced; consequently the processing time is reduced dramatically. In this regard, another important criterion is the availability of the number of different primitives which reduces the number of boolean operations in the CSG tree. This affects both the construction time of the output file and the solid modeller's processing time.

When the decomposition tree is completed, the leaves are wireframe representations of primitive solids or approximations there to. Hence, in constructing a CSG tree all the elements are solids, and consequently the output model produced in the root can be expected to be a true solid object.

Chapter Six

WIREFRAME MODEL **GENERATION**

Chapter Six

WIREFRAME MODEL GENERATION

6.1 Introduction

To generate a wireframe model requires many steps. First of all 3D nodes corresponding to 2D vertices must be created. Since projections of many 3D edges may be superimposed on 2D orthographic views, all combinations of 2D edges which are collinear must be considered in the generation of 3D elements. Some 3D edges may be false ones, so some effort must be made to prevent their creation, for example by using dashed line information.

A subroutine called WIRFRAME has been developed for this purpose. It accepts names of input files containing view data. Different parts of the algorithm are described in the following sections.

6.2 Nodes

Projections of any 3D node $P(x,y,z)$ appear in each view as $P_{xy}(x,y)$, $P_{xz}(x,z)$ and $P_{zy}(z,y)$. In order to recover the 3D nodes from the two-dimensional representations, for each vertex $P_{xy}(x_1,y_1)$ in a selected view XY , the other views are searched for vertices having coordinate values of $P_{xz}(x_1,z')$ and $P_{zy}(z'',y_1)$ respectively and which also satisfy the relation $z'=z''$. In this case all 3D nodes $P(x_1,y_1,z)$ that have a projection in the XY view (e.g. (x_1,y_1)) are found. This process is repeated for other

views as well. As an example, in figure 6.1 vertex 1 in each view is the projection of 3D node P. If vertex 1 in view XY is selected then vertices 1, 12, 11, 10 in the ZY view have the same value for y ("y1"), and vertices 5 and 6 in the XZ view have the same x value ("x1"). Therefore two nodes $P(x1, y1, z1)$ and $Q(x1, y1, z2)$ are found that have a projection $P_{xy}(x1, y1)$ in view XY.

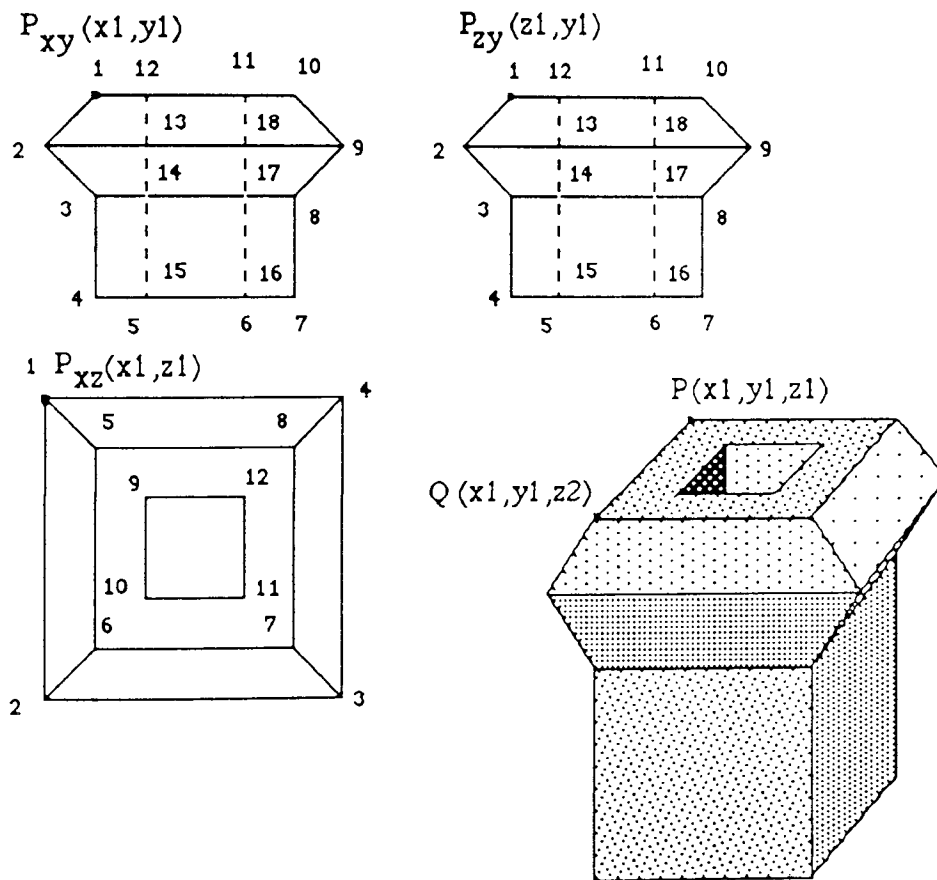


Figure 6.1 Relationship of 3D and 2D nodes (first angle orthographic views)

In this process many cases may arise, including:

Case 1 : If a 2D vertex in a view is a projection of only one 3D node, then it will produce the exact location of the 3D node when it is processed like vertices 1, 2, 3 and 4 in XZ view of figure 6.1.

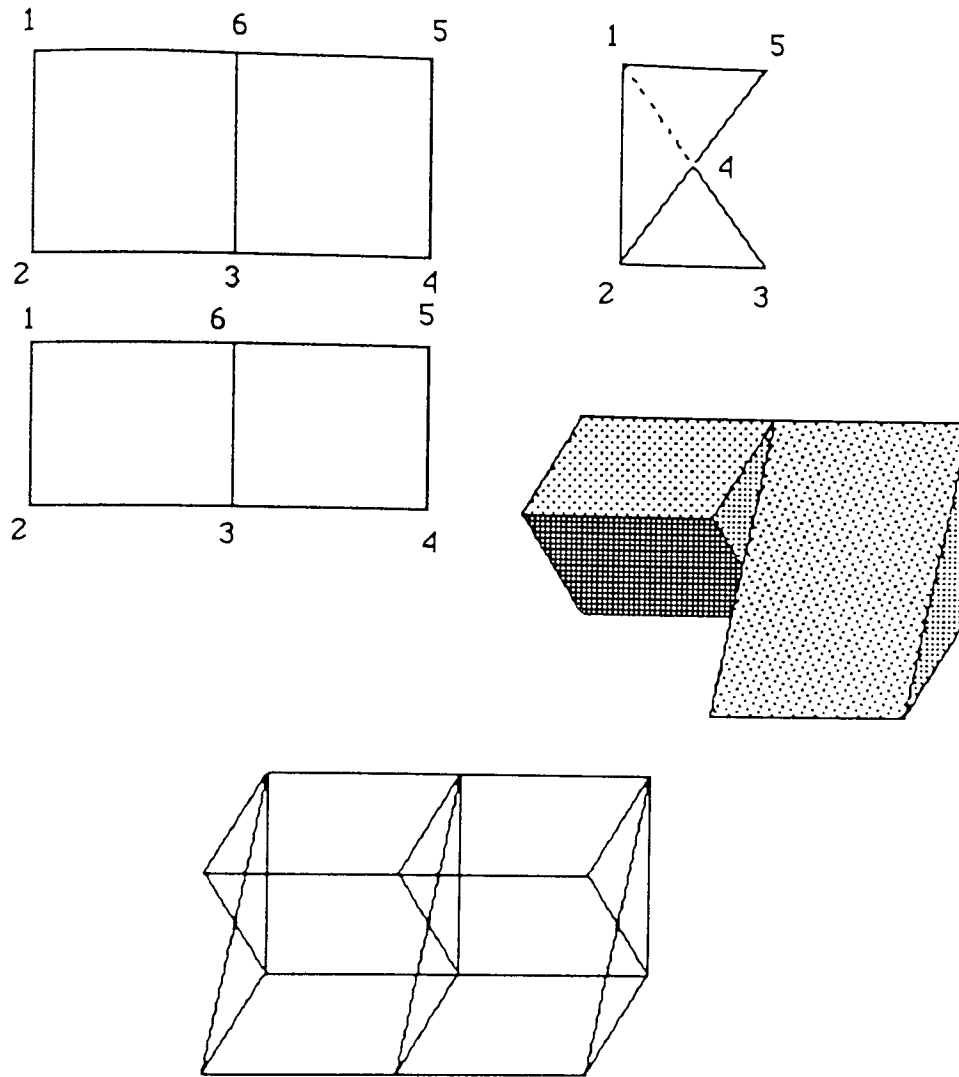


Figure 6.2 An example for ghost node that has not been produced

Case 2 : If a pseudo 2D vertex is not really a projection of any 3D node, but by chance there are some vertices in other views whose corresponding coordinates conform with the current 2D vertex being processed, then it will generate 3D ghost node(s) such as vertices 13, 14, 17 and 18 in view XY. In this case they will be checked and eliminated during the generation of 3D edges, and they have no effect in the process.

Case 3 : If a 2D vertex is not really a projection of any 3D node, then a 3D ghost node will not be created such as vertex 4 in ZY view of figure 6.2.

In summary, the method extracts all real 3D nodes, plus some extra "ghost" nodes.

6.3 Analysis of Superimposed Edges on Engineering Drawing

As explained earlier, any three-dimensional edge is represented by its three orthogonally produced 2D projections in an engineering drawing. In some cases, when more than one 3D edge is located in a plane orthogonal to the projection plane, then the pictures of the edges are mixed or, possibly, the projection of the edge is superimposed by part or all of the projection of the other edges. Some of these cases are shown in figure 6.3.

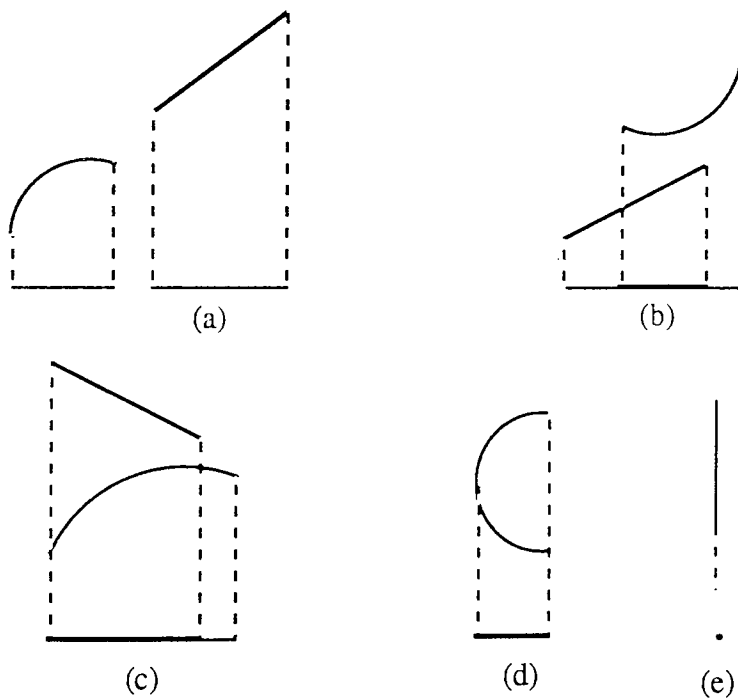


Figure 6.3 Different cases of superimposing of projections

These cases may also happen in a cylindrical plane when its axis is orthogonal to the projection view, as in figure 6.4. In such cases, when there are collinear edges in a 2D view, any connected combination of sub-edges may constitute a projection of a 3D edge. To decide which part belongs to a 3D edge projection, other views must be considered (figure 6.5).

In some cases, only two views need to be considered for decision making, while other cases, like the one in figure 6.5, need all three views to be processed. There are special

cases where even three views do not reflect the distinction of 3D edges. In such events other properties like dashed line and edge type information may be used. If this still is not conclusive, it means that the object cannot be represented by only three orthographic views, and an auxiliary view for that part of object is required.

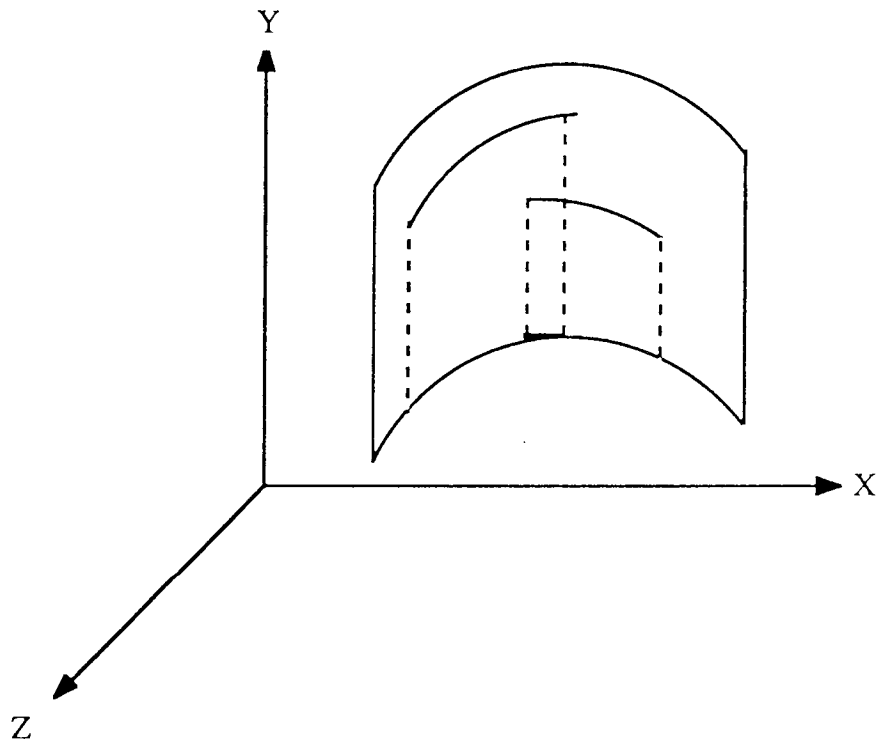


Figure 6.4 Overlapping the projections of two arcs

To distinguish a projection of a 3D edge in different views, and to determine its type, a subroutine called IPICCHK has been developed. This subroutine accepts a 3D edge located between 3D points N1 and N2, that has a projection in a view IV ($IV=1..3$) with a certain type of ITYPE. The routine verifies if there is any suitable representation for the projections of such a given condition among the other two views. If there is such a possibility, then it determines the type of this 3D edge. Depending on the type of the 2D element and the situation of start and end nodes selected for 3D edges there will be many different possibilities that must be decided by this routine. At present, arcs are assumed to be located in a flat plane parallel to one of the main planes, therefore in subroutine IPICCHK, a 3D arc having an arc projection in more than one view is

rejected. Thus, for example, case C in figure 6.7 is not acceptable. For future development, this routine can be modified easily to consider desired types of 3D arcs, including those which are not parallel to any main plane. Figure 6.6 shows different cases of 3D edges that are located in a plane parallel to one of the main planes and produce a straight line projection in a view. Figure 6.7 is the same as figure 6.6 but for the 3D edges located in a non-parallel plane; the case of a 3D arc is shown in figure 6.8.

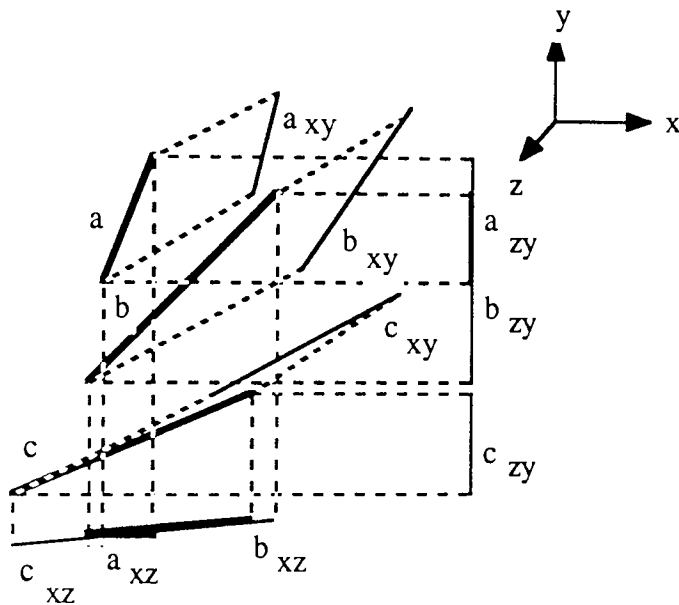


Figure 6.5 Distinction of 3D edges by three view projections

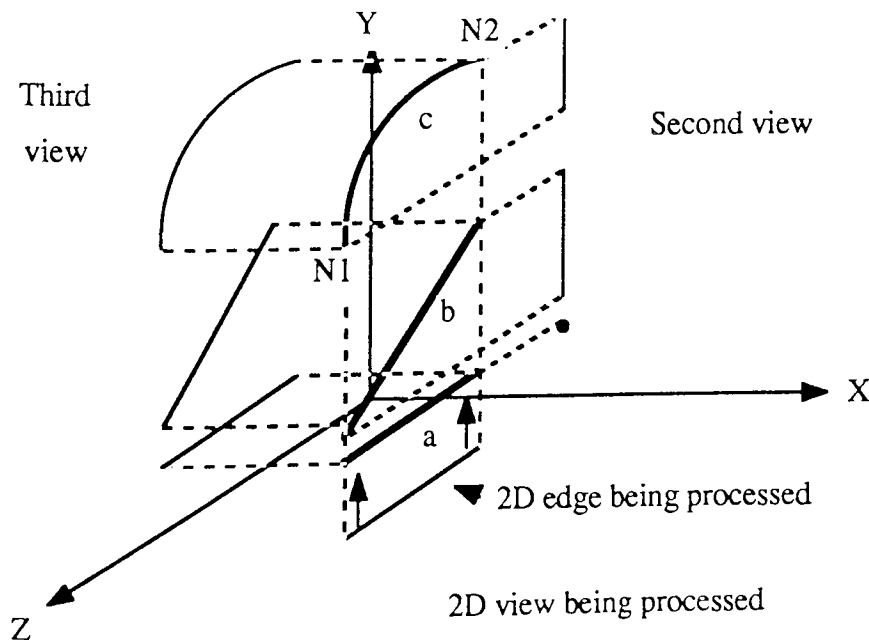


Figure 6.6 3D edges parallel to a main plane, having a straight line projection

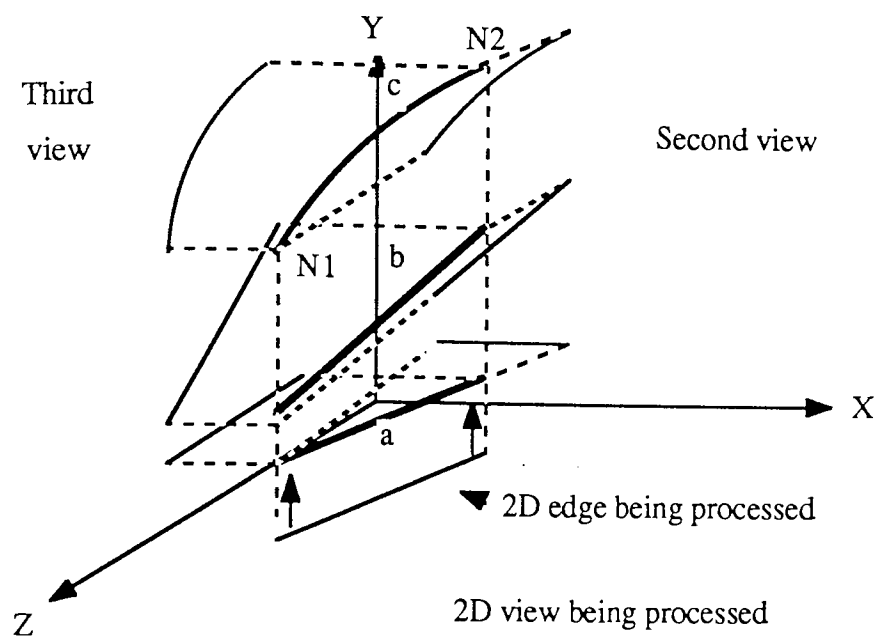


Figure 6.7 3D edges not parallel to any main plane, having a straight line projection

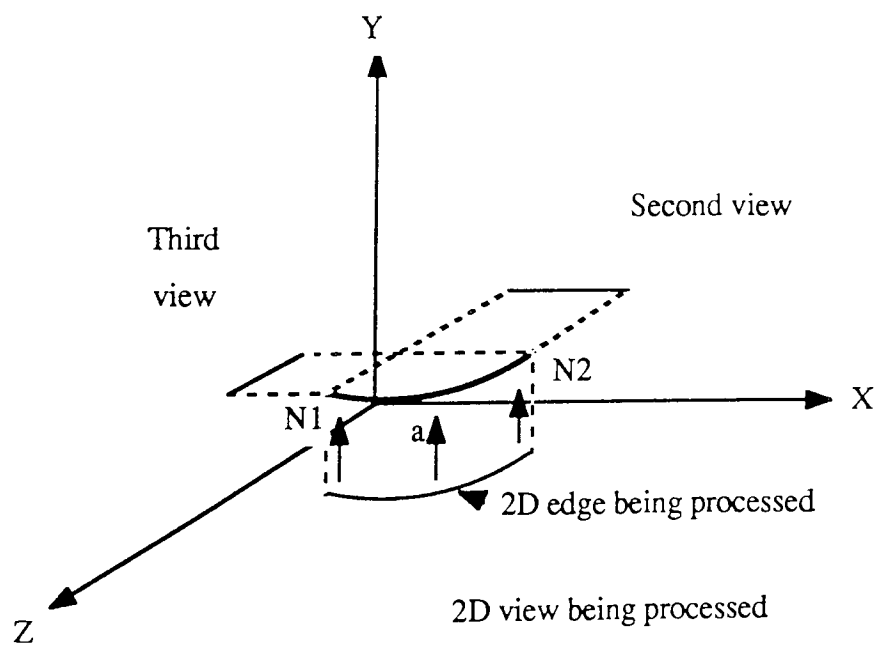


Figure 6.8 3D arc parallel to a main plane

6.4 Generation of 3D Edges

The second step towards wireframe model generation, after creation of 3D nodes, is the production of 3D edges, whether real or ghost ones. The method is implemented in a subroutine called GENSEG3 and it consists of the following steps:

- 1) Generation of all possible 2D edges. Because of the problem explained in section 6.3, at the beginning all possible combinations of connected sub-edges constituting a collinear edge are produced for each view and they are saved at the end of the corresponding tables. Actually some decision making in combining different parts of collinear edges can be important in reducing the number of generated ghost 3D edges. For example, if a selected combined segment has a sub-segment with dashed type or which is located on the boundary as part of it, then selecting the type of the resulting segment will be significant. A subroutine called GENSEG2D has been written to do all of these functions concerning the production of all possible segments for 2D collinear edges.
- 2) Each 2D element I in view IV, including those elements added in step one, that has not been marked as an isolated edge is selected. The isolated edges may exist because of having 2D pictures of sub-objects, which is described in the appropriate section. For the chosen 2D edge the subsequent functions are applied.
- 3) All 3D edges which can produce the current 2D element being processed are selected as probable 3D candidates.
- 4) A 3D element from the candidate list is selected and the following rules are tested.
- 5) Coordinates of start and end points of the 3D edge are checked to see if they lie on a straight line orthogonal to one of the main planes. This information is used in a routine called IPICCHK which verifies the presence of the projections of the candidate edge in other views. This check is to prevent irrelevant 3D edges being produced. Depending on the results of the output generated by this routine, it is then decided to reject or continue with the selected 3D candidate.

6) If the candidate 3D element has at least one projection on the boundary of any view, then its type is set as a boundary element and it is known that it is located on the boundary of the real solid provided that it is not a ghost edge, and it is definitely located on the boundary of the first approximation model.

7) A check is also made with the list of 3D elements generated so far to see if the edge has been produced or not. Because of different cases (as shown in figure 6.9), this check includes complete testing of start and end nodes, edge type and radius and direction of arcs in the case of circular arcs.

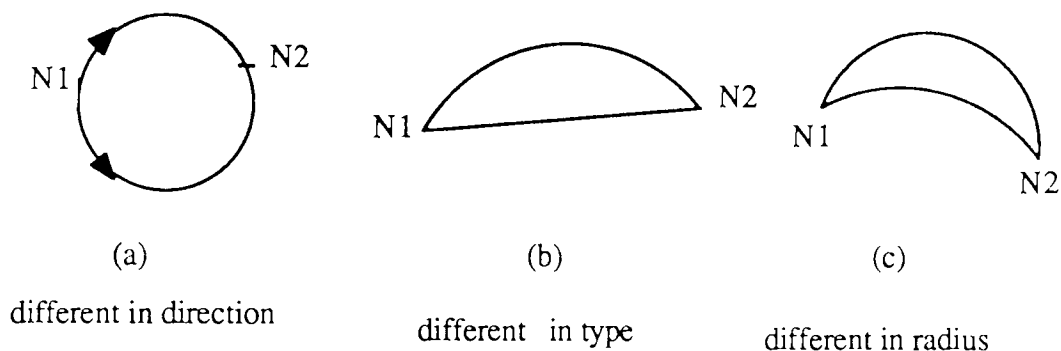


Figure 6.9 Different cases of 3D edges with common start and end points

8) Check if the approved 3D edge, passed from the last stage, has any common section with any other produced 3D edge. If there is no intersection between them, then the item is added to the 3D edge list without any changes, otherwise it verifies some kind of inconsistency. In other words, either the 3D intermediate node is a ghost one or one of the two edges, which covers the other one, is a ghost one.

In short, in these cases 3D edges are broken into common and non-common sections and all of the sections are stored.

In the general case, for the main problem, since the same algorithm is used for both processes of wireframe and the first approximation wireframe models between which

differences are selected, ghost nodes and edges have no effect, if they are located on the boundary.

9) Go to step 4 and select next item from candidate list.

10) Continue step 2 with next 2D element.

6.4.1 Dashed Lines

As a convention in engineering drawing, any edge which is obscured by at least one surface in the direction of sight is shown by a dashed line in that view. In the reverse transformation, from 2D to 3D, if a real 3D edge can uniquely be built from its 2D pictures, then this dashed line information is redundant. Figure 6.10 provides an example.

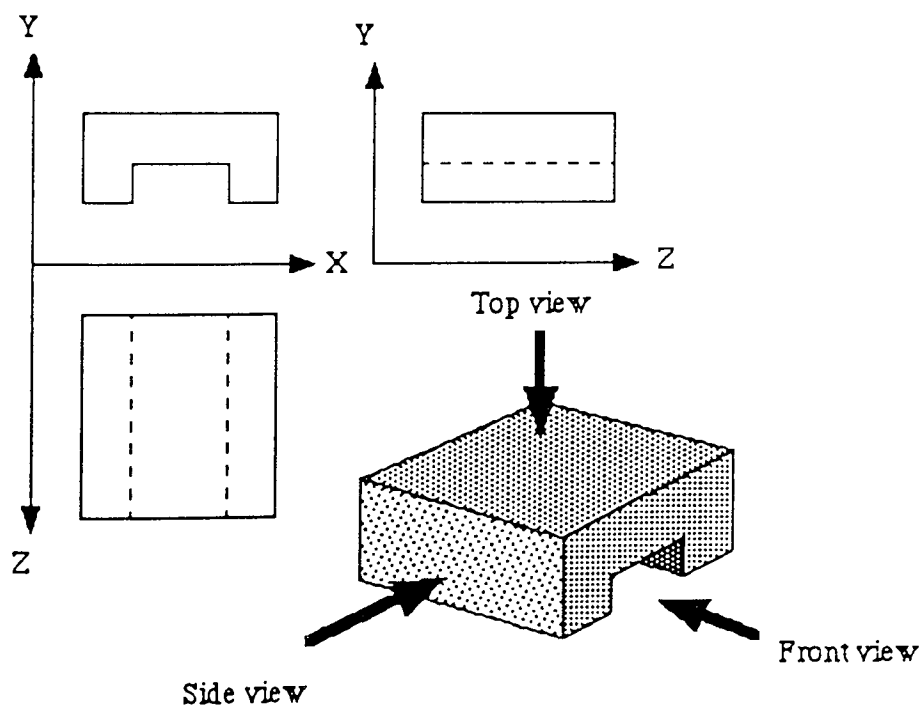


Figure 6.10 Example where dashed line information is not used

In some cases, however, dashed lines are very helpful in good decision making for the selection of the correct corresponding 3D edges and rejection of ghost ones.

Considering the example shown in figure 6.13, two 3D edges **a** and **b** are created in the wireframe model, corresponding to the 2D edges of **a** and **b** shown in the orthographic views of figure 6.12. Obviously if a 2D segment, which has a dashed line type, is used to create a 3D edge that is located on a surface closest to the viewer, then this process is not acceptable and the produced 3D edge is regarded as a ghost one. Edges **a** and **b** will therefore be removed from the wireframe model of figure 6.13.

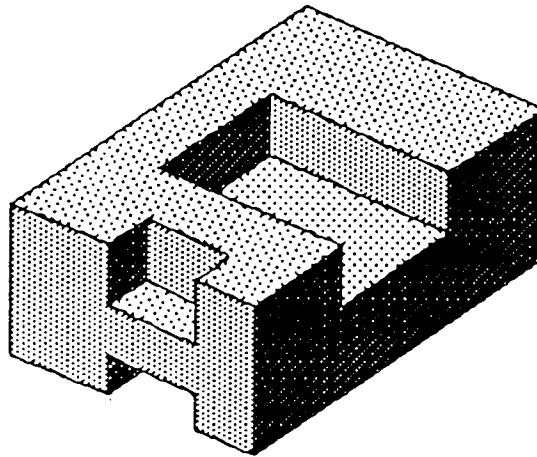


Figure 6.11 Solid object for dashed line example

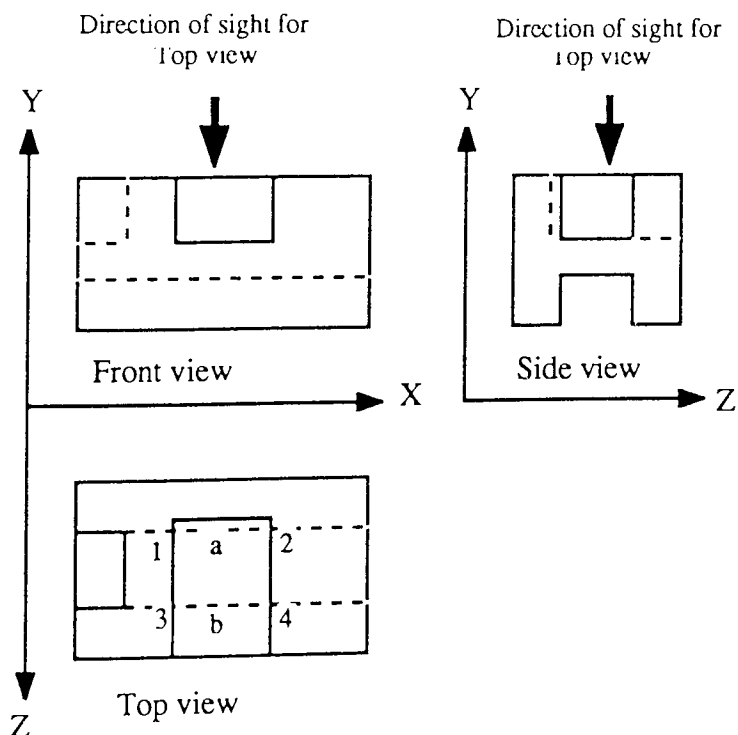


Figure 6.12 Orthographic views of object in figure 6.11

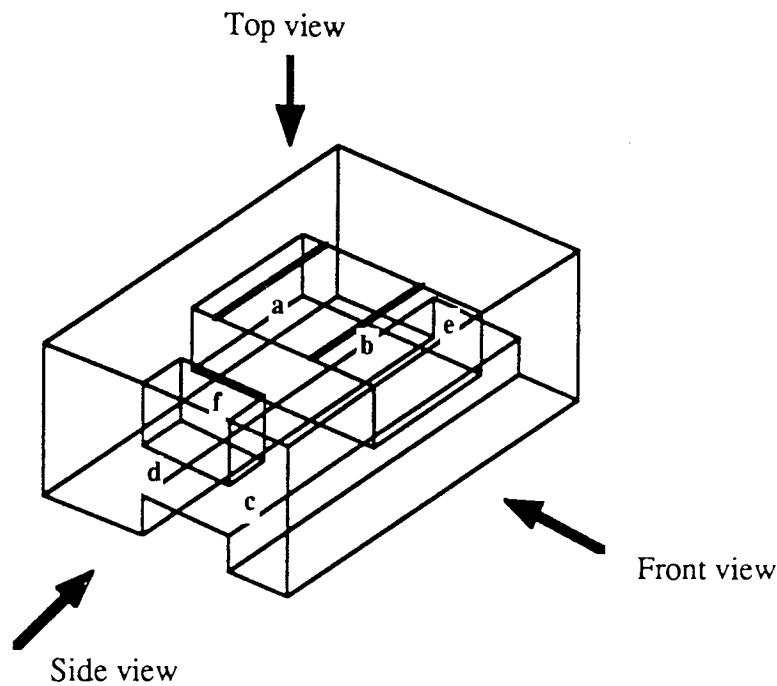


Figure 6.13 Wireframe model of the example

It should be mentioned that although this method does not work completely, because it can only prevent the ghost edges from being created in the front surface, but nevertheless it is helpful. This is not the only place that dashed line information is used: it is used in another section, which will be explained later. A subroutine called DASHFN is implemented for this purpose, which is applied on 3D edges after of their creation.

It may be noted that after each deletion of a 3D edge, its start and end nodes are checked for validity of the Euler equation: if the number of edges connecting to a node is less than three, except when they are two collinear edges, then they are removed, and so on.

6.5 Generation of Isometric View

The generation of an isometric view is needed to view the 3D results produced in different stages of the program such as the wireframe model of input data, first approximation wireframe model, and 3D differences of these two models, and so on. For this reason a subprogram called FINDISO has been developed which accepts 3D data in the form of a wireframe model and maps this into 2D coordinates, creating an isometric view (section 3.5.1). 3D arcs are approximated by small line segments.

6.6 First Approximation Wireframe Model

Another vital element as a part of the tools for development of the algorithm is having 2D orthographic views or/and wireframe models of the first approximation model as a reference for comparison. The first approximation model which is found by the subroutine FIRSTAPM is not useful for this purpose directly, because it is in the form of a set of instructions applicable to a solid modeller for building the object. Therefore one option is to execute these instructions by the solid modeller and request it to produce orthographic views and wireframe model of the object. Another option is to find them within the program. Although it would be easy to select the first choice, for many reasons, including its relative slowness and inconvenience of use, it was not used.

Since this problem is a simple case of wireframe modelling and most of the items, like 3D nodes, most 3D edges and other useful tables are already available, it was decided to choose the second option and implement this section as a part of whole package, and so minimise the number of calls to the solid modeller. The main routine which does this function is called FAWM, and output produced by it is referred to as "First Approximation Wireframe Model".

6.6.1 Overall Description

According to the definition, the first approximation model is the intersection of the extrusion of three views. So, to find the first approximation wireframe model, only boundaries of three views are used. Extrusion of each view means passing a flat or circular plane from each boundary element and orthogonal to that view. Obviously the intersection of two consecutive planes is also a straight line normal to the same view. In this case calculation of the intersection becomes much easier than the general case. Therefore, instead of finding the intersection of each boundary plane from a view with boundary planes of other views, the pictures of this normal line in other views which are straight lines parallel to the main axis are used. Then the intersections of these lines with other views are found and the common sections which are located inside or on the boundaries of both views are selected. By this means are found pieces of the 3D normal line passing through the 3D vertices, corresponding to 2D vertices, which belong to the boundary of the first approximation model. By repeating this process for every vertex of the 2D boundary and for all views, the 3D wireframe model for the first approximation model is found. In figure 6.16 this method has been demonstrated for the object shown in figure 6.14, for which the orthographic views are given in figure 6.15. In implementating the method many problems were confronted and many special cases found; some of these will be discussed in the following sections.

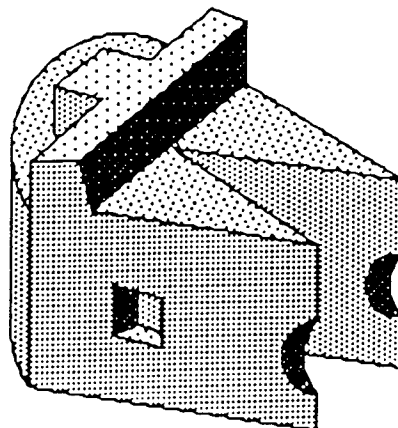


Figure 6.14 A sample object

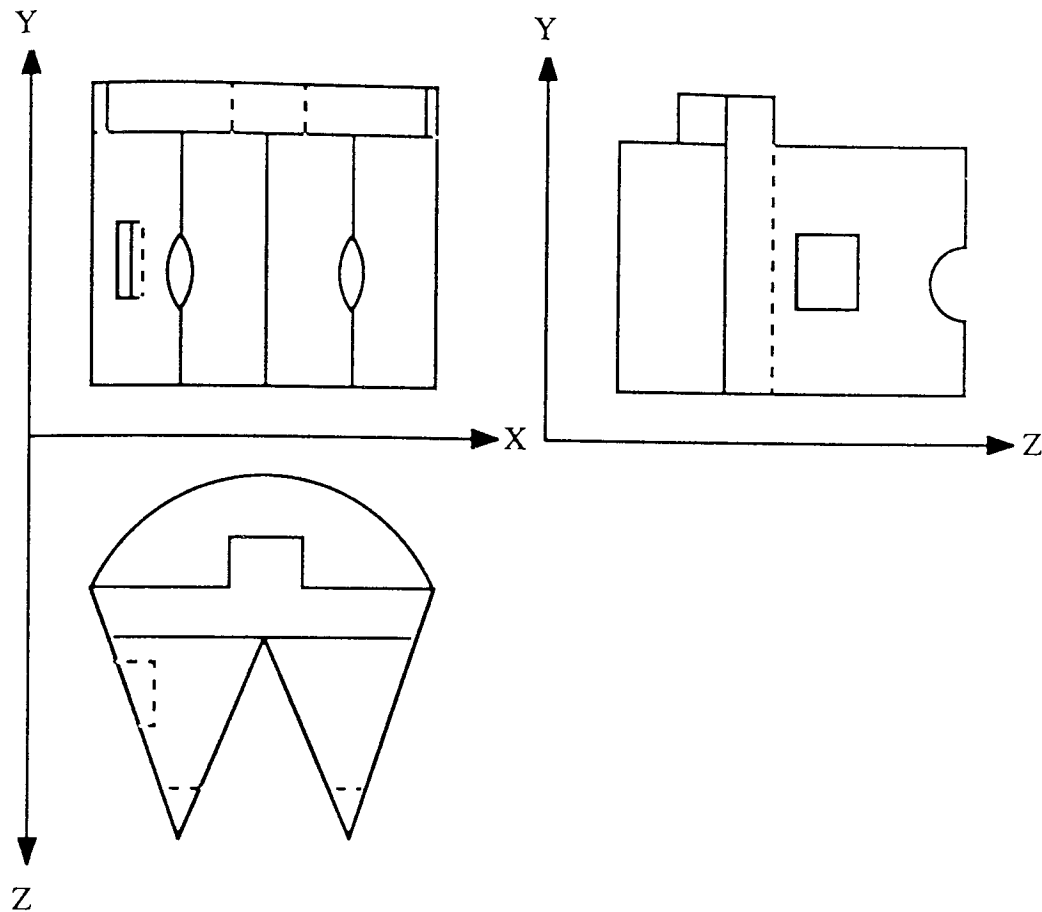


Figure 6.15 Orthographic views of object in figure 6.14

Briefly the method can be summarised into the following steps :

- 1) Select a view.
- 2) Select a vertex from the 2D boundary of the current view.
- 3) Find pieces of 3D lines normal to the current view and having the coordinates of the current vertex, so that their pictures are within or on two other boundary views.
- 4) Find if there are any inclined or circular 3D edges whose projection lies on the 2D boundary edge between the current vertex and next one.
- 5) If any new 3D node is created, then add it to the node list.
- 6) Check if a new generated node is located on any 3D edge; if so, then split that edge into two at this point. This is used to simplify the comparison section. Also, the projection of these new nodes may create new vertices on the boundary of views.

- 7) If any 3D edge belonging to the FAWM passes through any 3D node belonging to the wireframe model of the object, then break that edge at this node. This is also due to aid the comparison section.
- 8) Add produced 3D edges belonging to the FAWM to the separate list.
- 9) Continue step 2 with next 2D vertex.
- 10) Remove redundant created 3D edges of FAWM.
- 11) Go to step 1 for next view.
- 12) In case of new vertices on boundaries created in step 6, then the whole process (steps 1-11) is repeated for new vertices.

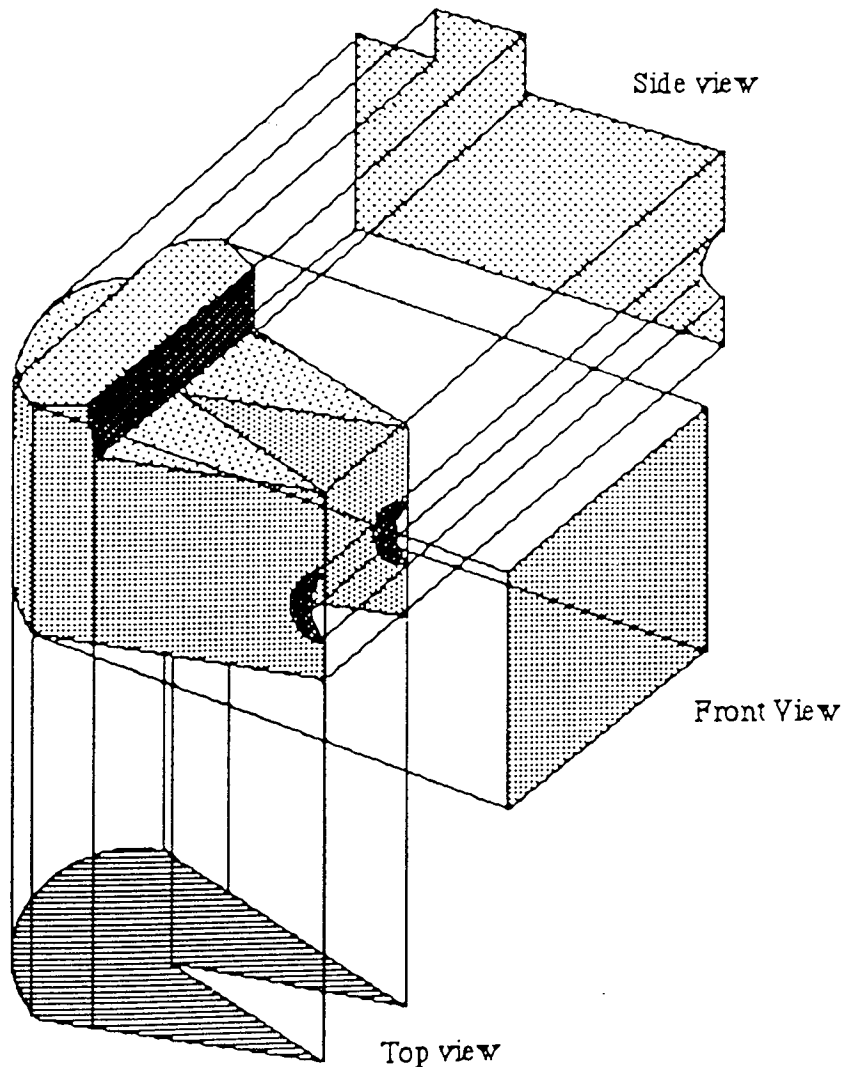


Figure 6.16 Extrusion of three views

6.6.2 2D boundary intersection with horizontal/vertical lines

As indicated in the previous section, the problem of finding the intersection of a flat or circular plane normal to a view with extrusion of the boundary of other views (figure 6.16), is reduced to the calculation of the intersection of a 2D view with a vertical or horizontal line parallel to one of the coordinate axis (figure 6.17). By this mean, the three-dimensional problem is converted into two-dimensional one and the line equation will be $x=c$, $y=c$ or $z=c$. In this equation the constant parameter c is equal to one of the coordinate values of the vertex on the 2D boundary, from which the normal line is drawn. In this calculation, which is done by a subroutine called LNVWS, different sections of the line are recognized as within, outside or on the boundary of that view. Then in subroutine CMNSEC, these sections of lines from two views are merged in order to specify the common pieces which are located on the boundary or inside of both views.

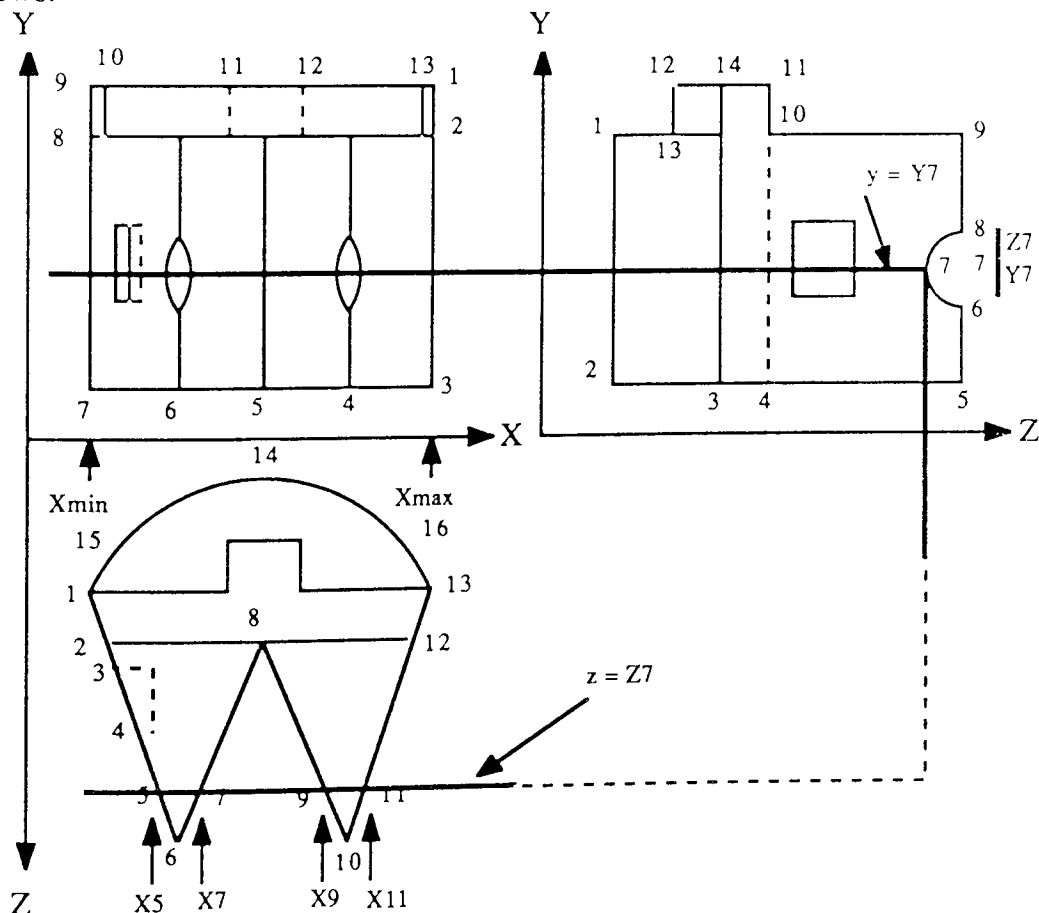


Figure 6.17 Intersection of a line with boundary of a view

To find the intersection of a line $y=c$ or $x=c$ with a 2D boundary which is a polygon including arcs, the value of x or y is set to a minimum so that the point on the line is certainly outside the boundary. As the value of the x or y is increased across the line, at some point it intersects with an element of the boundary of polygon. Therefore for each occurrence of an intersection the status of the line changes alternatively from outside to inside and *vice versa* (figure 6.19f). But in cases where the line $y=c$ passes through a vertex i , however, as shown in figure 6.19(a-e), the process of finding the inside/outside depends on the situation of the inward and outward edges and requires more testing.

To clarify the problem, orthographic views and the FAWM generation process examples are repeated in figures 6.17 and 6.18 with some annotations. Considering point 7($Z7, Y7$) of the Z-Y view in the figure 6.17, the intersection of line $y=Y7$ with the X-Y view gives a line starting from $Xmin$ and extending to $Xmax$, which is inside the view boundary. The intersection of line $z=Z7$ with the X-Z view will produce two pieces of lines inside the boundary of that view, which are from $X5$ to $X7$ and from $X9$ to $X11$. Now the common sections between these two sets of lines are $X5-X7$ and $X9-X11$. Therefore the 3D straight lines generated will be from points 12($X5, Y7, Z7$) to 13($X7, Y7, Z7$) and 14($X9, Y7, Z7$) to 15($X11, Y7, Z7$) of the 3D part of figure 6.18. All straight lines normal to each view are produced in this manner. At the same time, using some other techniques, all the inclined lines and arcs are added to the 3D edge list of the FAWM: for example, arcs 10-12, 10-13, 11-14 and 11-15 in the same figure. As an example new 3D nodes 1 and 2 (shown in figure 6.18) are generated, which create two new vertices 15 and 16 on the boundary of the top view. The 3D edge between nodes 1 and 2 of the FAWM passes through 3D nodes 3 and 4 belonging to the object wireframe model, so it is broken by subroutine BRK3DE into edges 1-3, 3-4 and 4-2 (example of step 7).

The final output of this process, which is done by subroutine LNVWS, is a vector

containing the start and stop values of x or y of the sections of the line which are located inside or on the boundary of the view in order of increasing values.

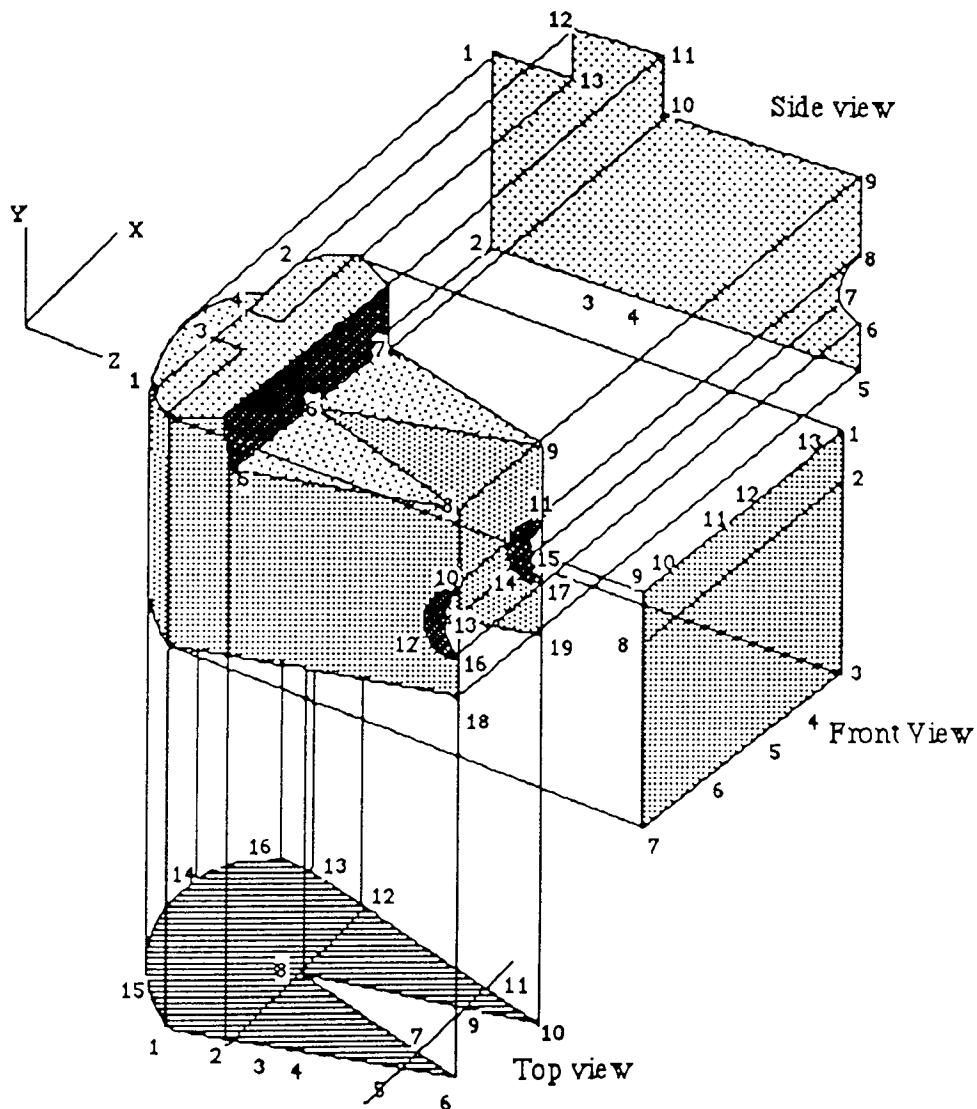


Figure 6.18 Example of creating 3D edges of FAWM

Note: If within a view there are any isolated loops which may indicate a hole through the object, they are not considered and only the outer boundary is processed. This is because in the main algorithm holes will be detected as separate sub-objects which are subtracted from the first approximation model.

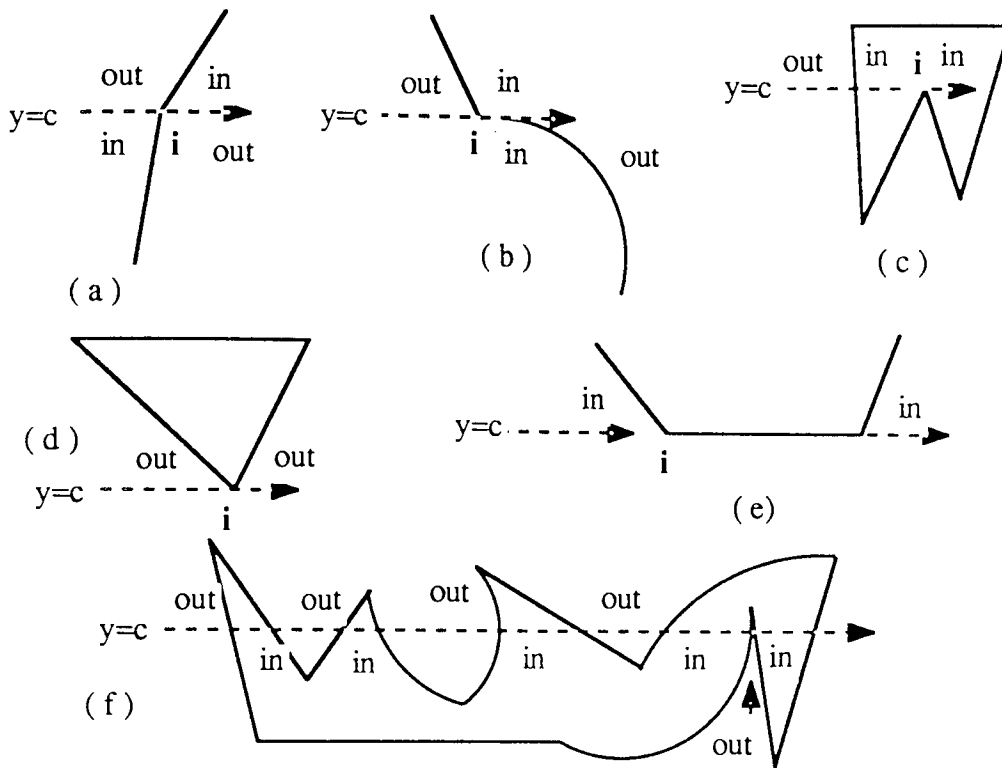


Figure 6.19 Different cases of finding the inside sections of a line

6.6.3 Removing extra lines from FAWM

During the procedure of finding 3D edges for FWAM, some vertices from the boundary produce extra 3D edges on a surface which are redundant, as shown in figure 6.21 for the boundary views of figure 6.20.

Some additional rules are applied to suppress such redundant edges and the final result will appear as drawn in figure 6.22.

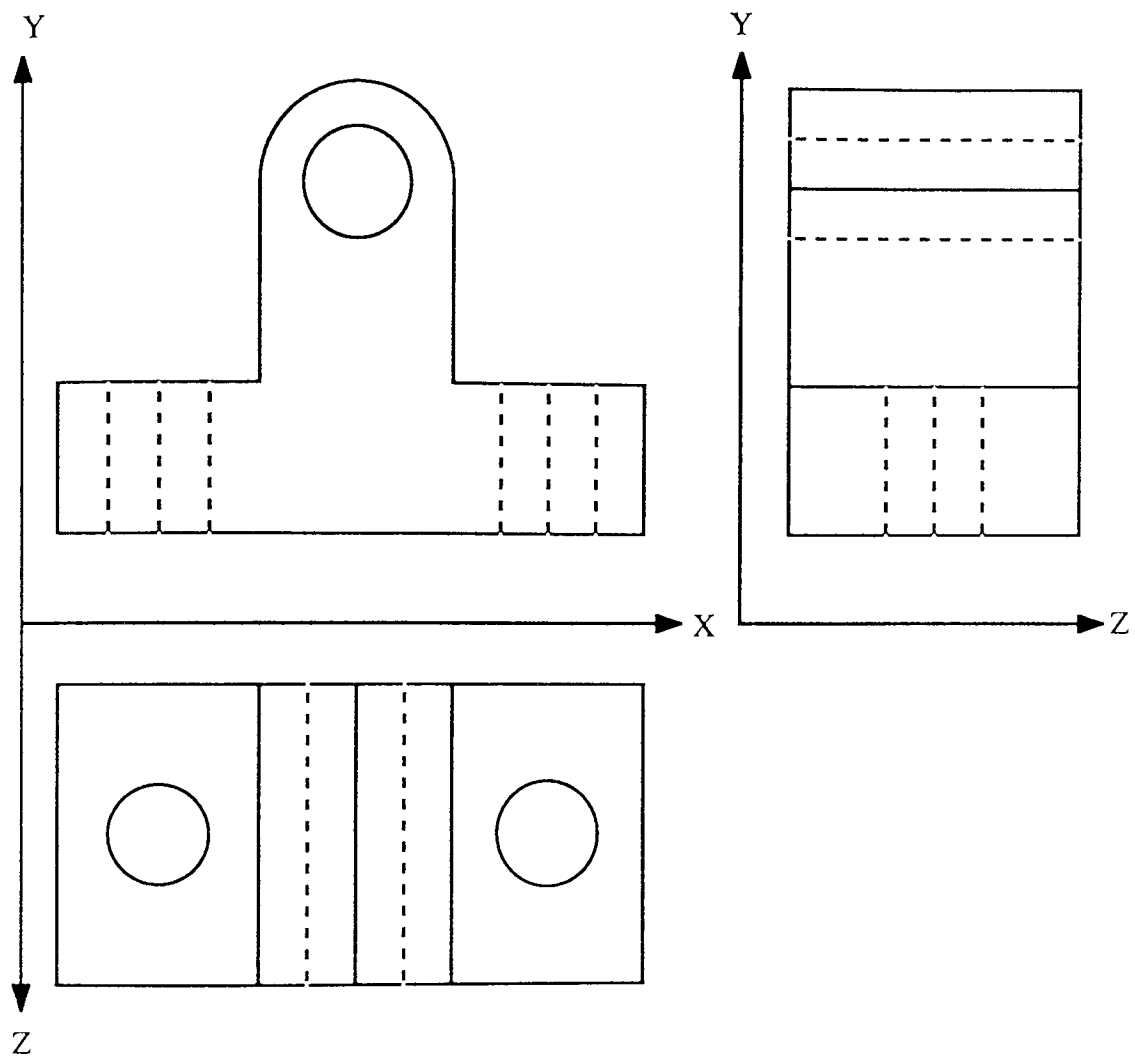


Figure 6.20 Orthographic views

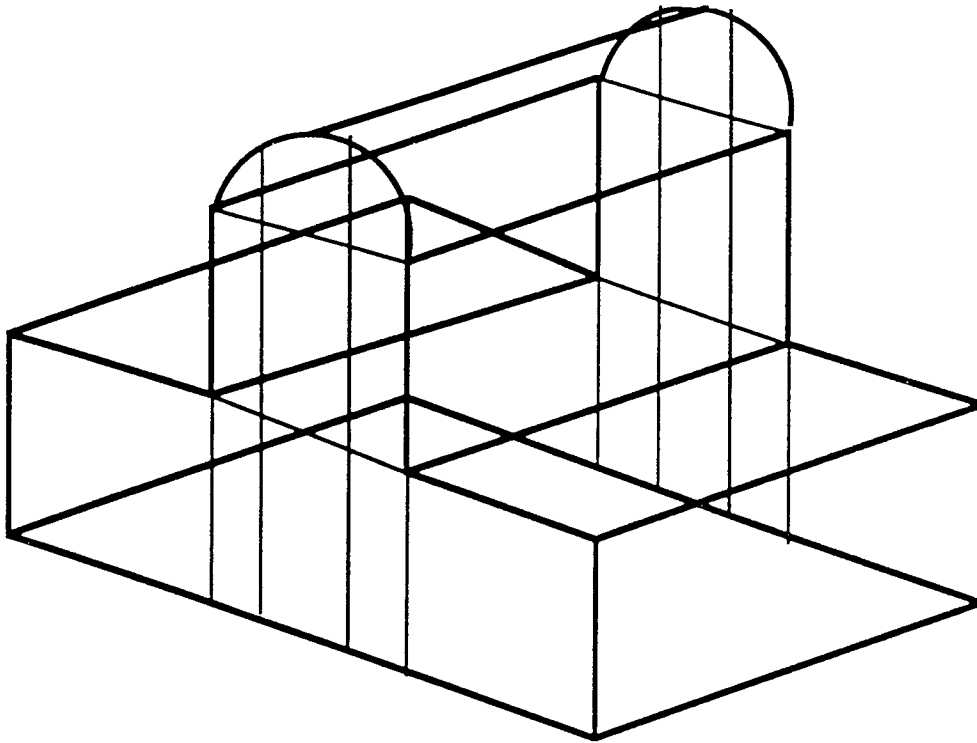


Figure 6.21 First approximation wireframe model with extra lines

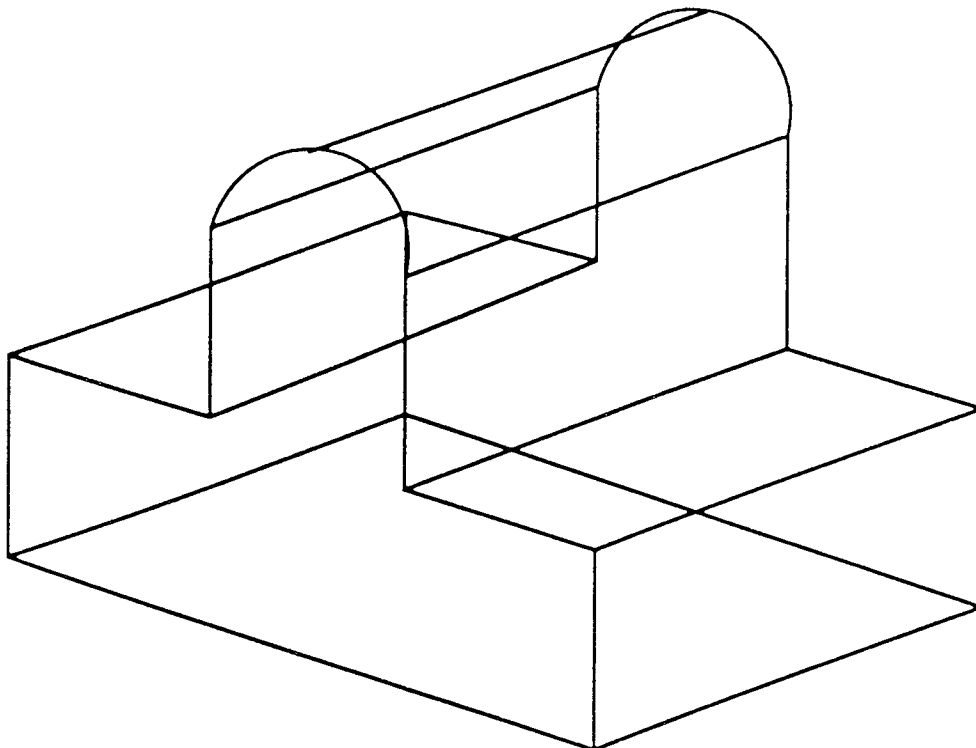


Figure 6.22 Final result of first approximation wireframe model

Chapter Seven

METHOD OF COMPARISON

Chapter Seven

METHOD OF COMPARISON

7.1 Introduction

Chapter six has explained the methods of creating the wireframe and the first approximation wireframe models of an object. According to the main algorithm, these two items are used to extract the differences between the main object and the first approximation model in terms of a set of sub-objects. Then, by repeating the same method for sub-objects, the process is continued until sub-objects are recognised as primitives, prismatic objects or objects with volume less than some defined error. Therefore in this chapter the comparison method, the extraction of sub-objects, and the way that they are combined to constitute the model for the object are discussed.

7.2 Overall Description

3D elements which have been produced by the boundary elements of 2D views will appear on the boundary planes of the first approximation model. Therefore, if there exists any sub-object as a difference between the object and the first approximation model, then certainly some of its 3D edges will not be on the boundary planes of the first approximation model. In this case their 2D projections will fall inside the boundary of views.

It is clear that if the edges from wireframe model are selected such that they do not

belong to the boundary of the first approximation model, then they will constitute all or part of the edges of sub-objects, depending on whether they are completely inside the first approximation model (figure 7.1) or have common edges or plane contacts with its boundary (touching the boundary) as shown in figure 7.2. In the case of having common elements with the boundary, some edges from the boundary of the first approximation model must be selected in such a manner that they complete the wireframe representation for sub-objects. In this selection process some information such as dashed line information may be used. Of course, some of these edges may not be the true one, and also from these 3D differences, each set of edges belonging to individual sub-objects must be separated. This will be discussed later.

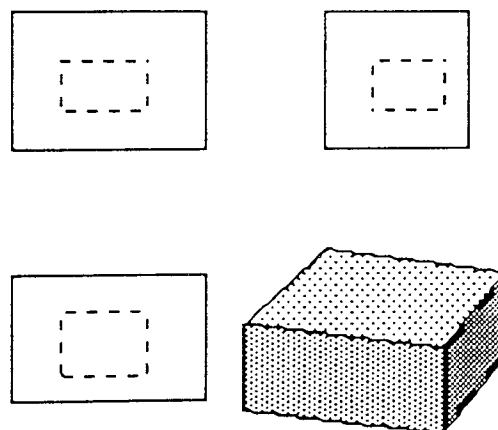


Figure 7.1 Sub-object is completely within the first approximation model

After finding the wireframe model for a sub-object, its 2D projection is found in order to identify the boundaries of views, which in turn are used to calculate the first approximation wireframe model. For each object or sub-object, the related command file for the solid modeller, for generation of its first approximation model, together with its position in the expansion tree (decomposition tree) are saved. At the final stage, according to the structure of decomposition tree, the final file containing the instructions for producing the object model are prepared for the solid modeller.

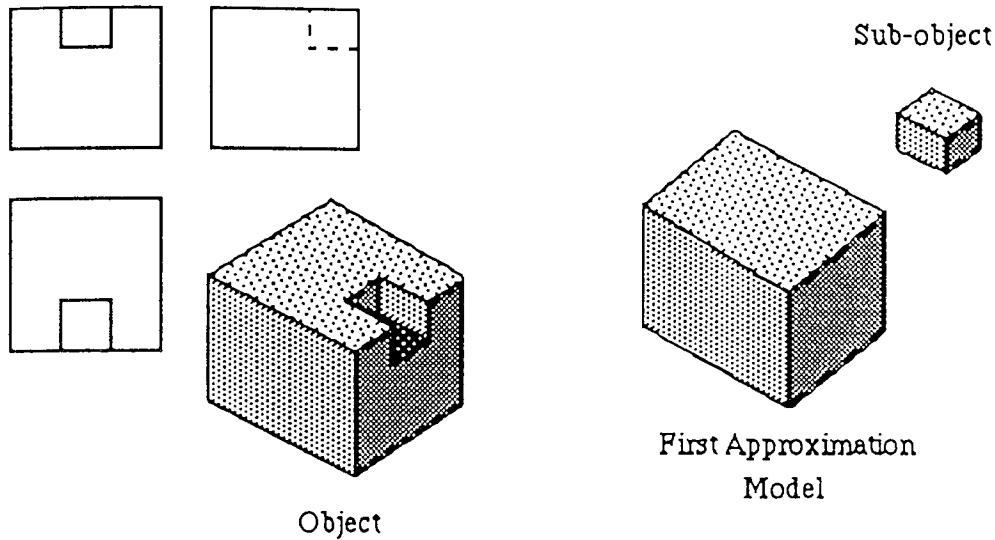


Figure 7.2 Sub-object has common planes with the first approximation model

7.3 2D Comparison

One method of extracting sub-objects, which must be taken from the first approximation model to produce the object model, is comparing the 2D projections of the object and first approximation models. To find the 2D projection of the first approximation model, the output file taken from the FRSTAPM subroutine is passed to the solid modeller and it is requested to produce the orthographic views. There is another easier way to achieve this and that is the use of first approximation wireframe model. For the reasons mentioned earlier (section 5.3), the second method has been preferred in this project. In figure 7.3, the first approximation wireframe model has been drawn for the object shown in figure 6.11.

In the process of finding the wireframe and first approximation wireframe models, some arrays have been defined to maintain the relationship between 3D edges and

nodes, and their corresponding items in 2D views. Therefore the 2D orthographic views of the first approximation model can easily be found from its 3D wireframe model. Also another array called the count vector, is used to show the number of 3D edges that map to a particular 2D edge for every edges in all views.

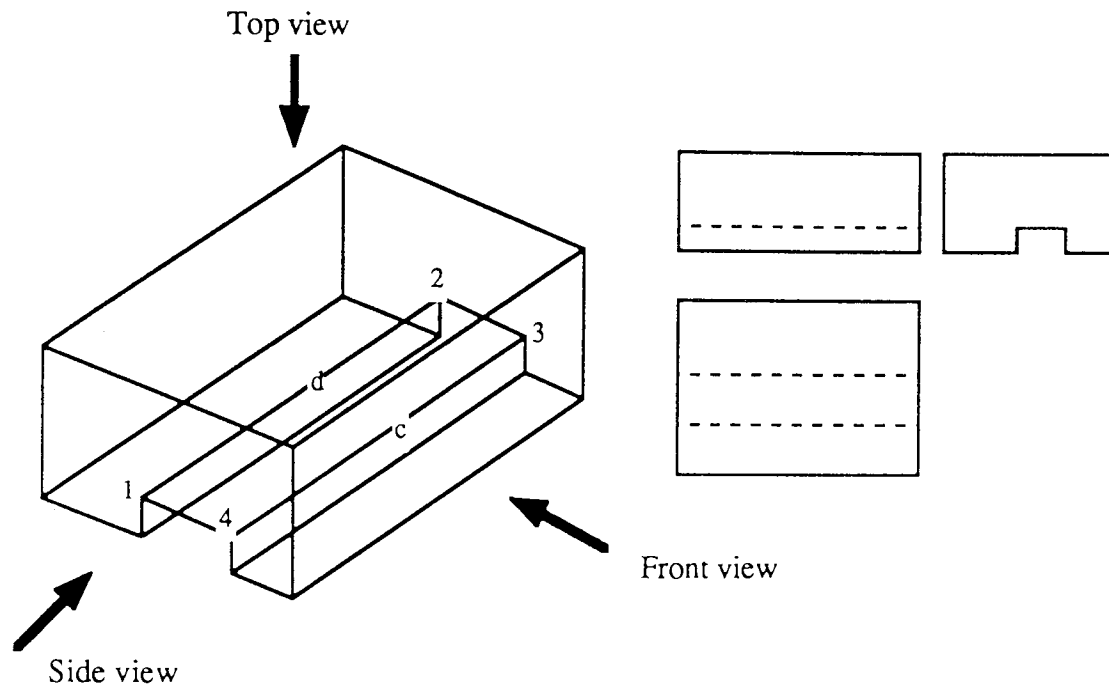


Figure 7.3 First approximation wireframe and its orthographic views for the object of figure 6.11

To find the differences, all of the edges from the wireframe model which are not located on the boundary of the first approximation model must be selected together with those edges from the first approximation wireframe model which are not part of the wireframe model. Common edges between the wireframe and first approximation models are deleted and the pictures of the others are selected as the differences.

Since 2D edges are compared and some of them may be the projection of more than one 3D edge, the extended 2D edge list, which includes all possible combination of collinear edges, must be first of all used. To remove the common edges, it is required to remove the boundary edges of the 2D views together with their corresponding pictures in other views and updating the count vector until its values for all boundary

elements(edges) are zero, for these 2D boundary edges in fact make up the boundary of the first approximation model. It must be noted that 3D edges which have a point as a 2D projection on the boundary of a view are still considered as a boundary element, like edges **c** and **d** in figure 7.3, so their pictures in the front and top views (long horizontal dashed lines) of figure 6.12 will also be removed in the 2D differences given in figure 7.4. As may be seen, because of many problems, including separation of the different sets of 2D edges belonging to individual sub-objects, which exist in the 2D extraction of differences, this method is abandoned at this point and the 3D comparing method, which is explained in the next section, is used instead. However this part has been implemented by subroutine BNDRMV for comparison.

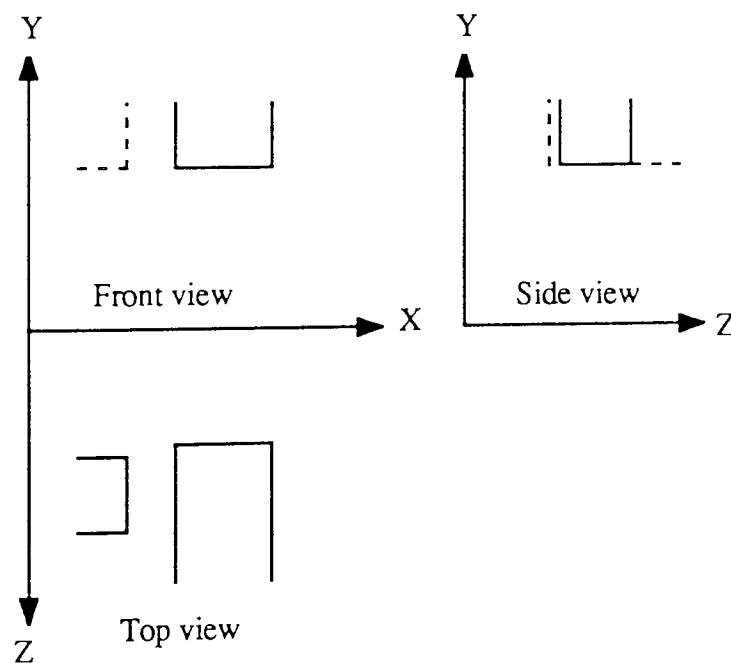


Figure 7.4 Two-dimensional differences of two models

7.4 3D Comparison

As was mentioned earlier in section 6.4, during the production of the first approximation wireframe model, each new 3D node that has been created is checked to see if it is located on any 3D edges and, if so, that edge is split at the same point. Similarly, if a produced 3D edge passes through a 3D node, it is also broken at that point (figure 7.6c). Therefore, two sets of 3D edges (wireframe and first approximation wireframe) are compared easily, the common edges are deleted and the remaining edges will belong to sub-objects, as shown in figure 7.5 and 7.6d. The name of the main subroutine which is implemented for this purpose is BNDRM3D.

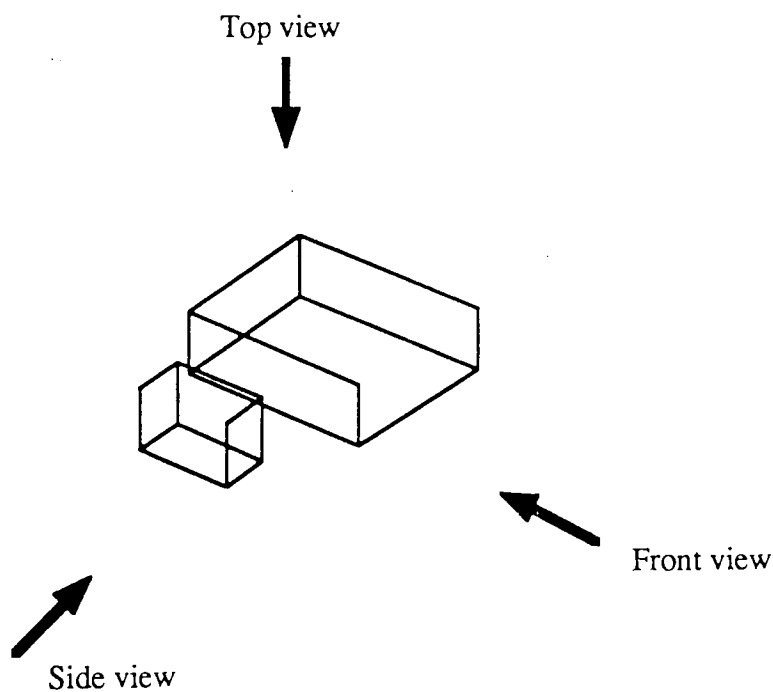
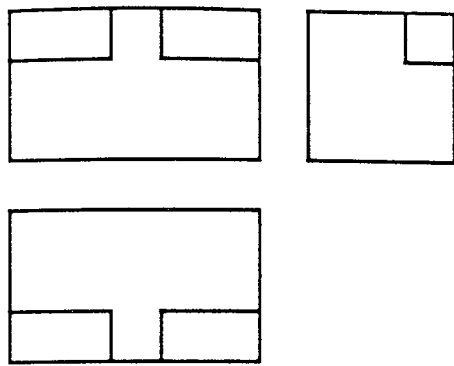
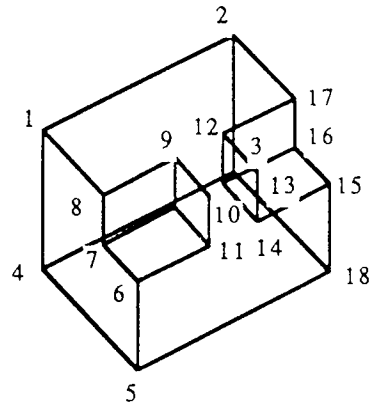


Figure 7.5 Three-dimensional differences of two models

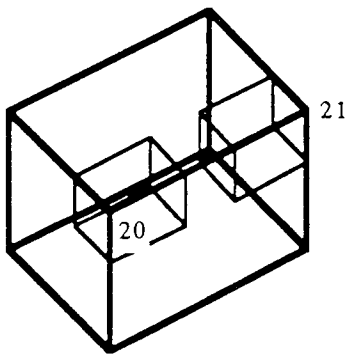
In this method, separation of edges belonging to different objects is achieved much more easily than the previous one; just by clustering the connected groups. Note that some sub-objects may have a common node or an edge, but this will not create a serious problem as is shown later.



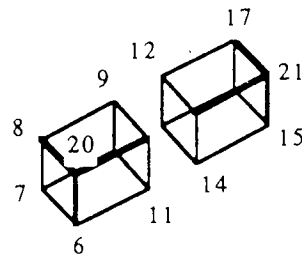
(a) Orthographic views



(b) Wireframe model



(c) Wireframe and First Approximation
Wireframe Model (in bold)



(d) Three dimensional
differences

Figure 7.6 Another example of three-dimensional differences extraction

7.5 Mapping of 3D differences to 2D

In previous section the 3D edges which constitute most parts of sub-objects were determined. Since it is intended to apply the same algorithm of finding wireframe and first approximation wireframe models and other related calculations for sub-objects as well as the main object, it is necessary to find the 2D orthographic views of sub-

objects.

As was explained before, because of the relations between 3D edges and their 2D corresponding elements are stored in transformation vectors, the mapping process from 3D to 2D is very easy and fast. Moreover, for most of the cases this transformation frustrates the effect of ghost edges produced in last stage (provided that ghost edges are created on the boundary of the first approximation model) and has caused the corresponding edges from 3D differences to be removed. For example, ghost edges *e* and *f* created in the wireframe model of figure 6.12 have caused the loss of two edges from the 3D differences shown in figure 7.5. But as may be seen, when transformed into 2D, the pictures of the other edges, which in their 2D pictures in original views have caused the production of ghost edges, will complete the 2D loops as depicted in figure 7.7a. The other example in figure 7.7b shows that the 3D differences are representing the complete wireframe model of sub-objects.

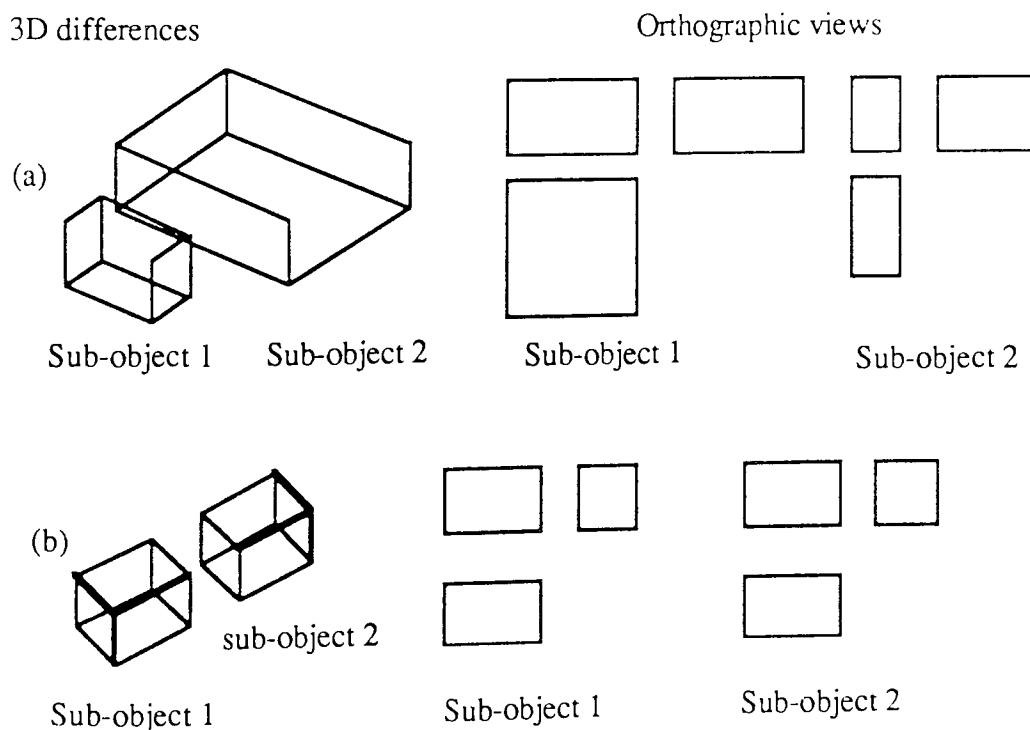


Figure 7.7 Mapping 3D differences into 2D orthographic views

It must be mentioned, that the 3D differences do not always represent sub-objects in so straightforward a manner, therefore some further processing must be done on them or their 2D pictures, which will be discussed in the next section.

7.6 Extraction of 2D Loops

3D extraction of edge differences calculations are done between two wireframe models. In this condition the results are a set of 3D edges and there is no information regarding the surfaces. Since the available information for reference base is the original orthographic views, these 3D edges are mapped into 2D. It is clear that the projection of the boundary of a surface which is a closed loop is also a closed loop. Even in the special case where this projection in a view is a line, it graphs as a closed loop in at least one other view. Using this property, and searching in the 2D projections of 3D edge differences for incomplete loops, will reveal those parts of the sub-object which have been in common with the boundary of the first approximation model, and have been removed in the comparison process. So they must be completed again with some elements of the boundary (the projection of the touched section).

As a further example 3D edge differences between two wireframe models of figure 5.3 and their 2D pictures are shown in figure 7.8a and 7.8b respectively. By using a method, which has been illustrated in figure 5.5, two sets of orthographic views belonging to two sub-objects are separated. Then the boundary loops of views are closed, using some parts of the object first approximation model boundary, which are shown as bold in figure 7.9. Further difficulties here are the rules concerning the selection of boundary pieces; these are discussed to some extent in the next section.

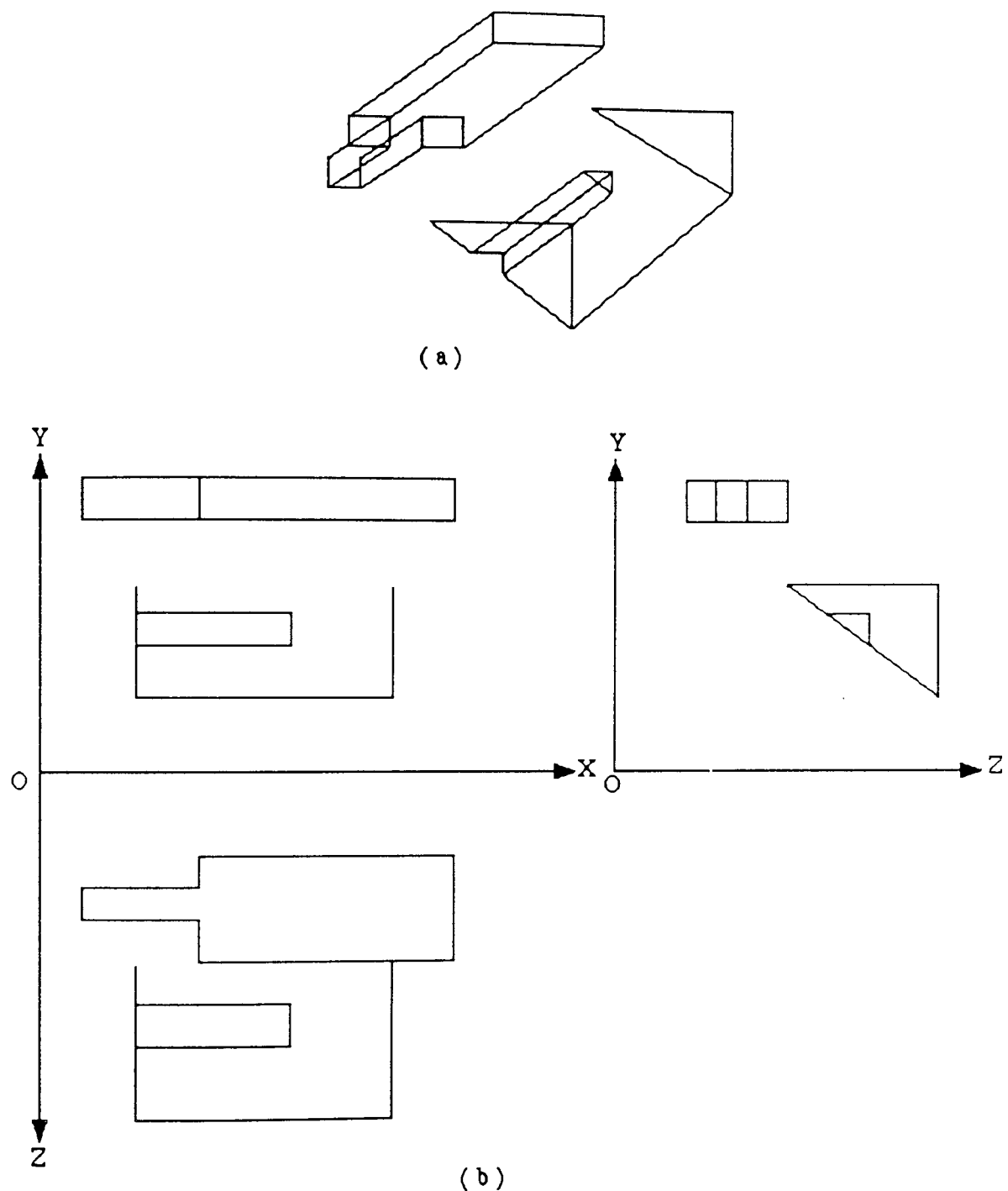


Figure 7.8 3D differences and their 2D orthographic views

Although applying Euler's rule on 3D edges meeting at each node belonging to the set of differences will also detect these missing parts, the 2D extraction of loops is preferred here. But the main task of separation of 2D views belonging to different sub-objects is done by utilisation of 3D edges.

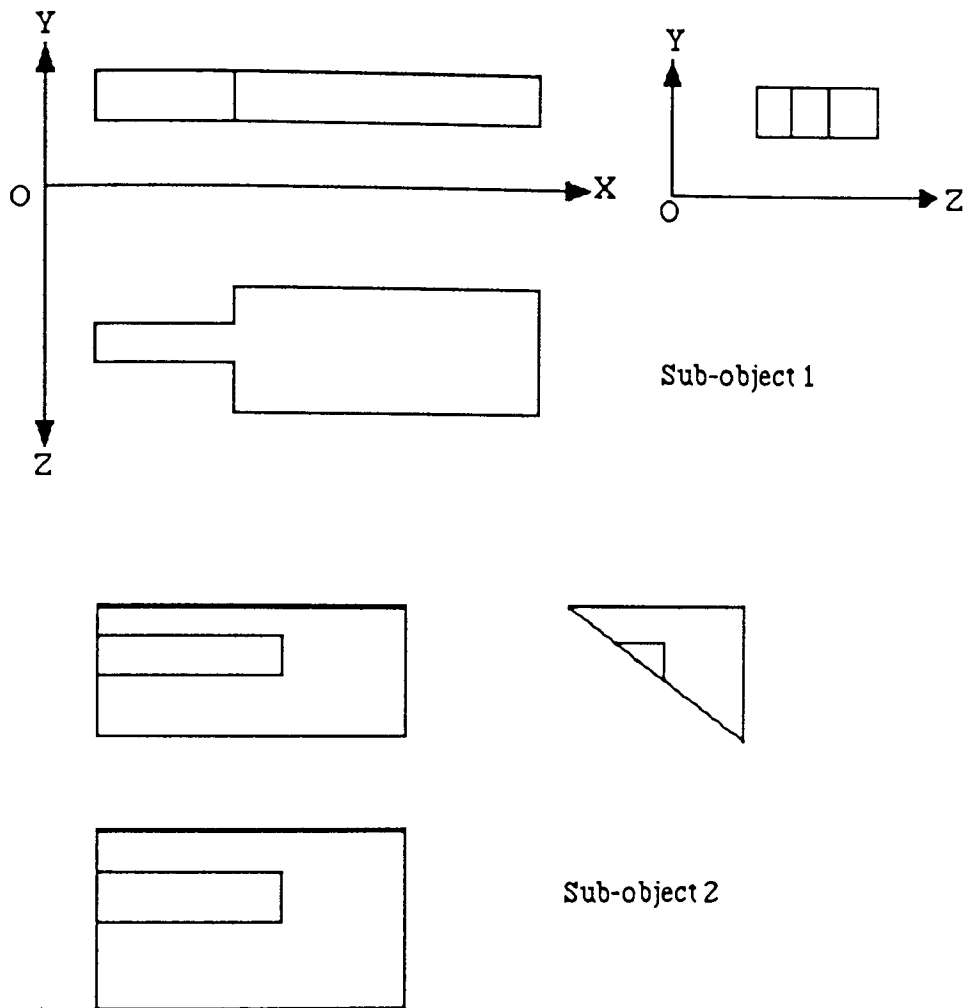


Figure 7.9 Completed orthographic views of sub-objects

7.6.1 Boundary elements

At this stage, two sets of 2D elements are available. One is the 2D graph resulting from the projection of 2D differences, and the other is the original object's views, from which only the boundary elements are noticeable. The aim is the selection of the part of boundaries of object views that also complete the boundary of sub-object views. It should be noted that this is required for the case where the sub-object has a common edge with the first approximation model; for the other cases (e.g. sub-object is completely within the first approximation model) the boundary of views already form a

closed loop.

Partial selection of the boundary elements to complete the boundary of sub-object views mostly depends on the shape of the object. At present the program can handle a certain class of objects. But this restriction does not create any severe problems for the main algorithm structure. By finding a good solution to the boundary completion for the general sub-object case and implementing it, the domain of objects covered by the program can be extended.

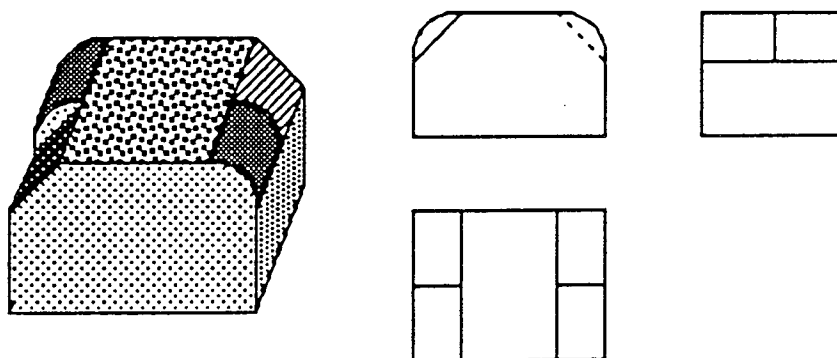


Figure 7.10 Dashed line application example

7.6.2 Dashed lines

Some times it is difficult to choose the correct elements from the boundary to close the boundary views of sub-objects. In this situation using other information such as dashed lines can ease the decision making. For example, figure 7.10 shows an object with its orthographic views. The wireframe model generated and the first approximation wireframe model on top of it (in bold) are shown in figure 7.11. The 2D projections of

the resultant 3D differences are also shown in the same figure. Since there are two separate loops in front view, perimeter loops corresponding to these two loops must be accomplished in other views. But there are confusions in the selection of 2D edges (a,b,c), (d,e,f) or (g,h,i) as part of boundary elements, to constitute top and side views for two sub-objects.

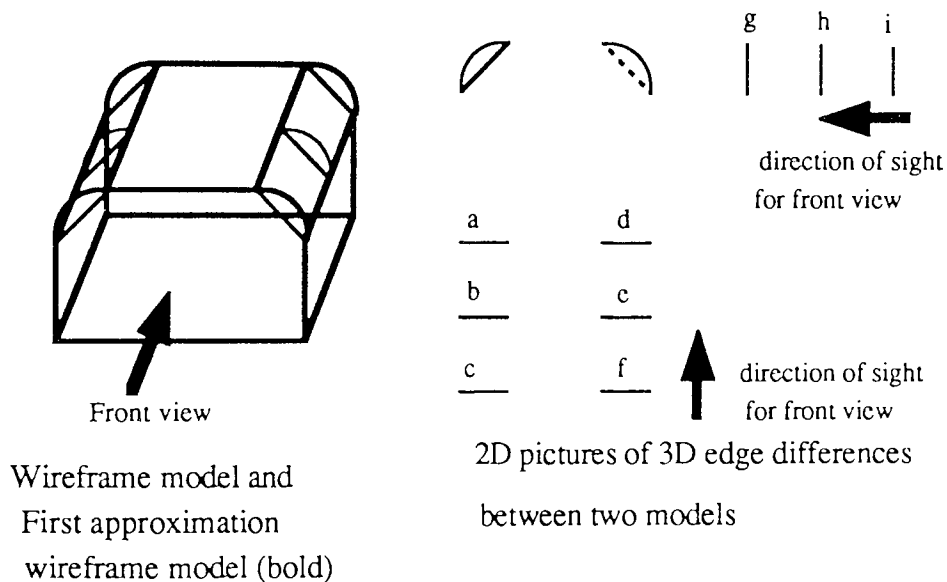


Figure 7.11 Wireframe and first approximation wireframe models

In this case, by considering the position of the dashed line in the front view, it is obvious that its corresponding pictures in top and side views in the direction of sight cannot be those closest to the observer. So 2D edges (e,d) from the top view and (h,g) for the side view are selected for sub-object 2, and the rest of the loop is completed (shown in figure 7.12). Using the same explanation for true lines, corresponding loops for sub-objects 1, are also found, as shown in figure 7.12.

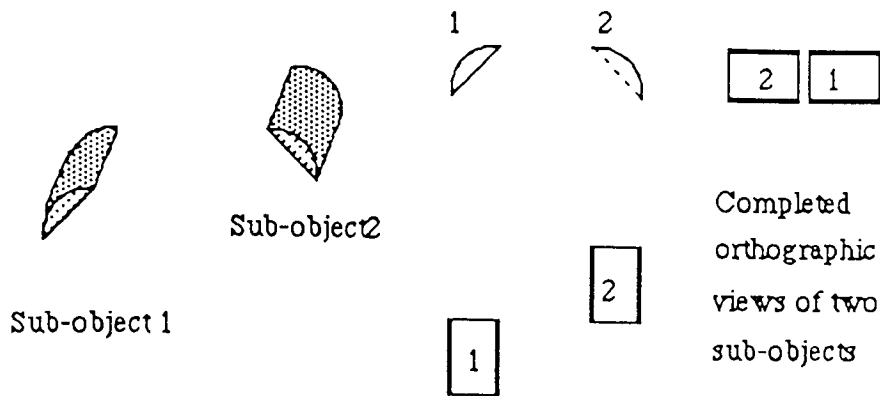


Figure 7.12 Completed boundaries for sub-objects

7.7 Identification of Primitives

At this point the processing may be reviewed. The initial 2D input views are given to the FRSTAPM subroutine which calculates the first approximation model in form of a data file containing commands to a solid modeller to build the object model. It also indicates the status of the output model: whether the input views belong to a primitive, prismatic, or nonprismatic object. In case of a primitive object, the output file for the solid modeller contains only a command for producing that primitive, which can be any of the primitives shown in figure 3.7. If the object is prismatic, then the contents of the output file is a set of regularized boolean operations to make an extrusion of a base view from a set of primitives. Naturally the primitives used in this condition are of prismatic type, such as cylinders, cubes, sectors, wedges and fillets. For these two cases the output model represents the exact model of the object and no further processing is required. For nonprismatic cases, however, the first approximation model produced by FRSTAPM subroutine is far from the real object.

7.8 Use of Recursion to Improve the Accuracy of the Model

As explained in preceding sections, instead of accepting the first approximation as a model for the object, the following actions are taken to produce a more sophisticated model.

The 3D edge differences are calculated and then transformed to 2D spaces. Those 2D boundaries which are not closed loops are completed by some elements of the boundary, and they are stored in the order that they have been produced for later processing, as well as those which already consist of closed loops. The list which keeps these orderings is called the "active list". It does not make any difference if the whole loop has not been completed. Even if a partial loop from each view which is consistent with matching box, has been selected as a graph of the part of the sub-object, then the rest of the sub-object will be detected in subsequent stages.

However, some of the sub-objects may not be primitives or prismatic objects. In this case, although the final model is closer to the real object, in comparison with the initial first approximation model, it is still not an exact representation of the object. Therefore, each sub-object can be treated as a main object and it is again expanded in term of its own first approximation model and a set of other sub-objects. These sub-objects are also added to the same active list in the same order for further processing. This method of expansion of each node of the expansion tree in order of their production is known as the "breadth first method" and applying the same rule for a child in a tree as its father is called "recursion". This recursive method is applied to any child selected from the active list, and is continued until the leaves of the tree are either primitives or prismatic objects, or very small defined first approximation models. In this way, each time a recursive call is made, the final model will be improved.

To demonstrate the method described so far, 2D orthographic views of figure 7.13 are given to the program as input views, while its real object is shown in figure 7.15. The 3D wireframe model of figure 7.14 is produced by the program.

After calculation of the first approximation wireframe model, extracting 3D differences, and transforming them into 2D, three sets of orthographic views belonging to three sub-objects are found. As explained earlier these views are stored and the sequence of generated sub-objects, which are shown in figure 7.16, is also saved in a vector called the "active list". The entities of this vector are used as an indices to sub-objects. Therefore after the first level the contents of active list are as follows:

Active list : 1 , 2 , 3 pointer =1 ;

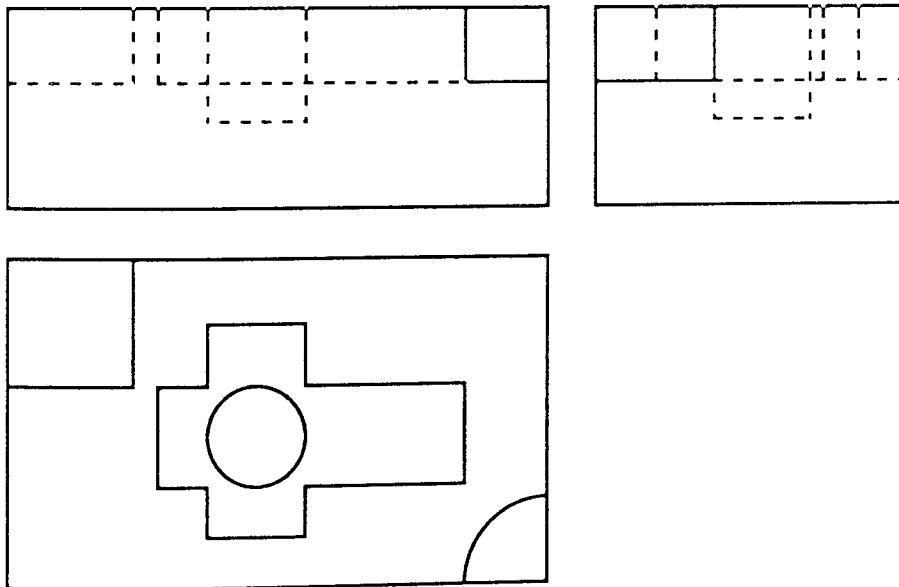


Figure 7.13 Input first angle orthographic views

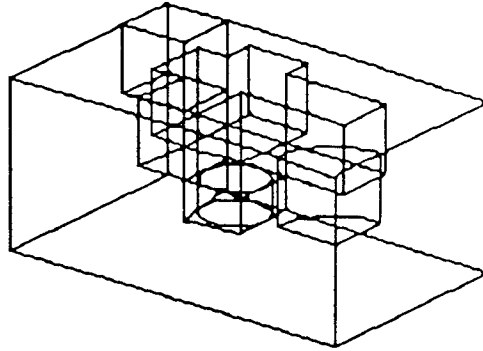
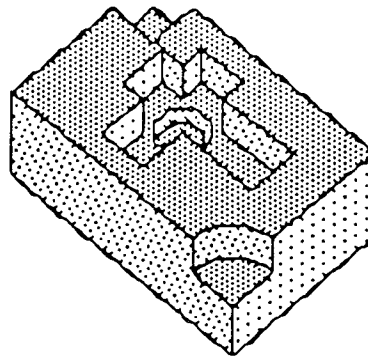


Figure 7.14 Wireframe model for solid object of fig. 7.15

The pointer value indicates the next element of the active list which is going to be processed. 2D views corresponding to the first element (pointer=1) of the active list are selected as new input and the process is repeated. Since sub-object 1 is a primitive cube, only the equivalent command for its generation is put into the BOXER file. At this stage :

Active list : 1 , 2 , 3 pointer =2;



Main object

Figure 7.15 A complex example

This time, second 2D views (pointer=2) corresponding to sub-object 2 are chosen. Execution of the same process will result the production of the wireframe model of figure 7.17. Because this is not a primitive or a prismatic object, a new sub-object 4 is found, which is stored as element four of the list :

Active list : 1 , 2 , 3 , 4 pointer=3;

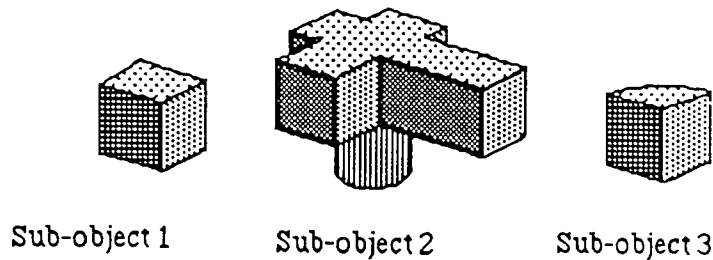


Figure 7.16 Sub-objects created at level 1

Continuation of the process will identify sub-objects 3 and 4 as primitive and prismatic objects respectively. The CSG tree for sub-object 2 together with wireframe model of sub-object 4 are given in figure 7.18.

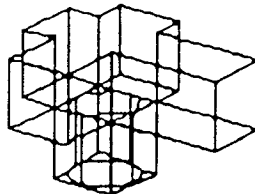


Figure 7.17 Wireframe model of sub-object 2

This recursive process is stopped when the active list is empty. The program then assembles the final file for the solid modeller. Finally the decomposition tree is displayed on the graphic screen, as illustrated in figure 7.19. It should be mentioned that the created file for the solid modeller (BOXER.DAT) has been fed to the solid modeller (BOXER), and figure 7.15 is its output result. Also, the solid modeller was asked to find the first angle orthographic views of the assembled object, which matched

completely with the original input views of figure 7.13.

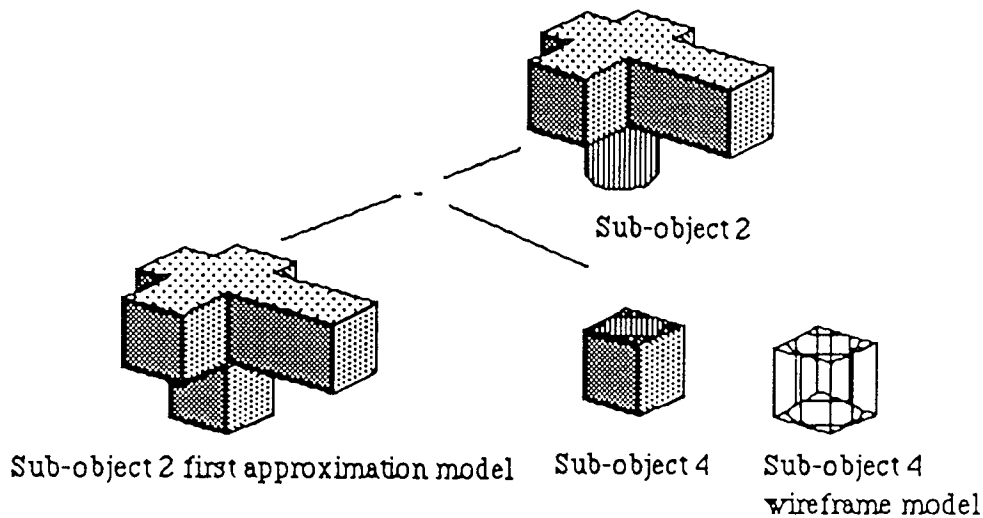


Figure 7.18 CSG tree for sub-object 2

Here, by looking at the figure 7.20, the effect of recursive processing in the improvement of the output model is quite obvious. If the process is stopped at level 0 (only finding the first approximation model of initial input views), the output model will be 7.20a. Object model of figure 7.20b will be the result if the program stops after level 1 (using only the first approximation model of sub-object 2). But, from figure 7.20c it is clear that after completion of level 2 (for this example), the exact model of the object is gained.

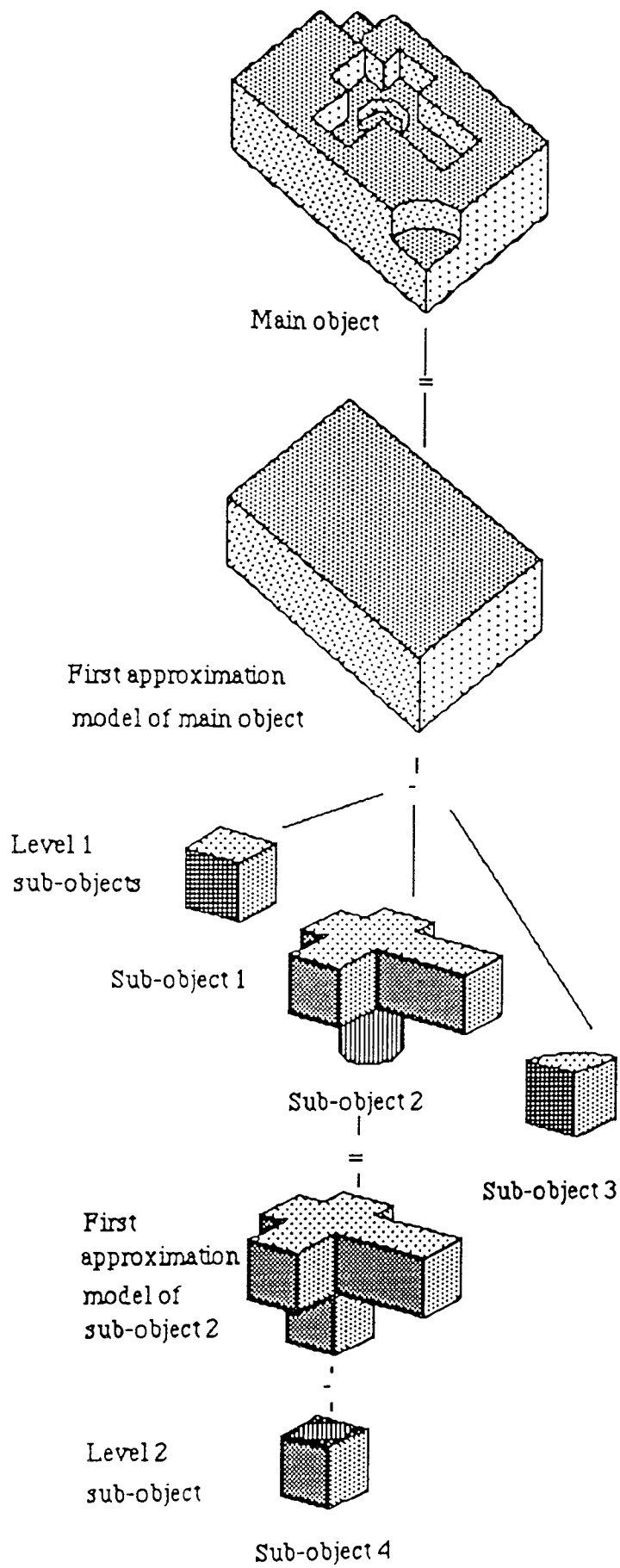
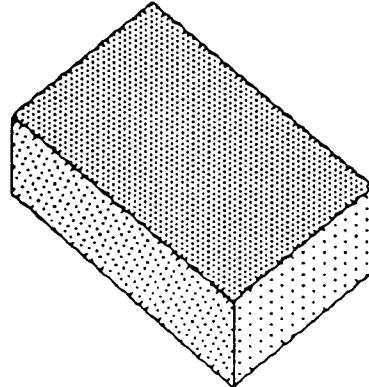


Figure 7.19 Decomposition tree

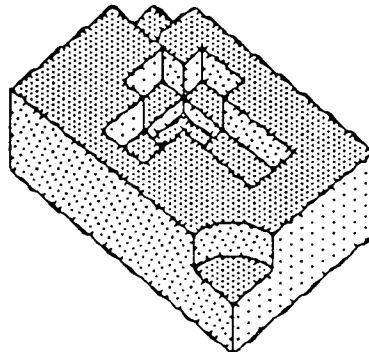
The MAIN section of the program (figure 5.8) is responsible for coordinating the calls to different modules, keeping track of produced sub-objects and monitoring all activities needed for recursive decomposition. Finally it assembles the ultimate BOXER file, according to the decomposition tree structure, from each individual sub-object BOXER file.

Output model

(a) at level 0



(b) at level 1



(c) at level 2

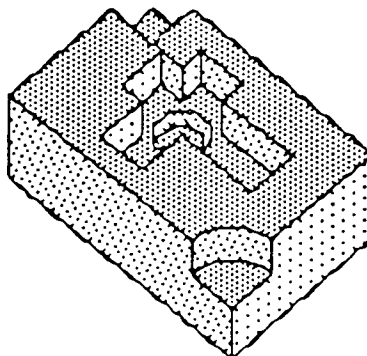


Figure 7.20 Improvement of the model from top to bottom

Chapter Eight

PRACTICAL RESULTS AND **DISCUSSION**

Chapter Eight

PRACTICAL RESULTS AND DISCUSSION

8.1 Programming Requirements

This package is written in FORTRAN 77 on an Apollo workstation (DN3000) and the GMR graphic library [Apollo, 1986] is required for interfacing with the graphic display to enable viewing of the results. The program consists of several modules, each of which contains many other subroutines/functions. Each time a module which is under development has been altered it is modified by the editor and then it is compiled to create the new object module. A text file called "LINKX" is used to keep the list of object modules, which must be linked together to make an executive program. To make the program ready for execution, the following instruction is required after logging on to the system:

```
INLIB /LIB/GMRLIB ; only for the first time after each logging on
```

```
....
```

```
EDIT modulename.FTN
```

```
FTN modulename.FTN
```

```
....
```

```
other editing and compilation
```

```
....
```

```
LINKX ; to build executive program
```

8.2 Input/Output

8.2.1 Input Options

For each view a separate filename is allocated, under which name a data file is created.

To run the program, its name is simply entered as follows :

CIEDSM

the program responds :

input filename of the view XY

The user then inputs the filename for that view; if it already exists on the disk then the next view filename is asked for, otherwise this message is printed:

THIS FILE DOES NOT EXIST

DO YOU WANT TO CREATE IT ?(Y/N)

At this stage, if the user has entered a wrong filename, he will type "N" and the program will respond :

PLEASE TRY AGAIN

input filename of the view XY

If the program is asked to create it, the INPUT routine is called and the data for corresponding view is read from the keyboard [Kargas, 1988] and stored on the file.

This process is repeated for other views.

When the program asks the user to input the first filename, if "/" is entered then it inquires about options :

DO YOU WANT LIST OF DATA ?

If answer to this question is YES, then during processing the desired data are printed.

DO YOU WANT IGES OUTPUT FOR GRAPHICS ?

This option will create an IGES [Bloor, Dodsworth & Owen, 1984] format file equivalent to each frame of the graphic display. These standard files can be used to transfer graphics data from one program/machine to another one (e.g. Apollo to VAX). Here it is used to read the image created on the screen of the Apollo to a drafting program called DOGS [Pafec ltd., 1984] and then produce the output on an HP colour plotter. Also this IGES file is accepted by the same program (DOGS) on VAX, and the output is captured on the Macintosh screen by emulating the Tektronix 4014, and may then be pasted to documents.

8.2.2 Output format

Output from the system appears into two form: text and graphics. Some outputs are printed as text, while others that relate to the shape of object, for better representation, are displayed on the graphic screen. To avoid unwanted data, options are available to skip some part of output. In the following sections these are explained.

8.2.2.1 Text output format

When the program is run and the input files containing the orthographic views are entered, depending on the option selected by the user, in addition to the graphics frames, a series of other output is printed as text. The complete output will contain the following information :

- A message for the start of the analysing section, indicating which sub-object at which level is processing.
- Giving the processed input class type (i.e. primitive, prismatic or non-prismatic), and in the case of primitive indicating its type.
- Informing the start of wireframe section.
- Number of 2D edges and vertices in each view.
- Geometry and topology information, specifying the edge type as well as those which are located on the perimeter loop for each view.
- Coordinate values of the 3D generated nodes.
- Geometry and topology information and edge types of 2D edges which may be added to each view for collinear cases.
- Topology of produced 3D edges, and the conversion table which relates each element in 3D space into its corresponding projection in 2D views.

These outputs are repeated for each individual sub-object and at the end the final BOXER file is printed.

8.2.2.2 Graphic screen format

The most important part of the program is the graphic part, which communicates with the user. This interfacing section reflects the pictorial results on the graphic screen for

better perception of the user, and acquainting him with what is going on during the process. Good design of this section is related to the art of computer graphics, and depends on the attitude of program designer. The following factors are effective in designing of a good graphic program :

- What results to be displayed ?
- At what stages of the program ?
- Form of representation.
- Utilisation of the full available capabilities of the graphic display.

There are five different graphic frames selected for displaying the results in the program. They are as follows :

Frame 1 - This frame is allocated to display the input views, as shown in figure 8.1. The boundary elements are drawn in yellow . The solid edges inside the perimeter are shown in white, while the dashed ones are in purple. After display of the screen, the program waits for user's response to either of the following questions :

ONLY FINAL RESULTS ?

ARE THE VIEWS CORRECT ?

If views are not correct, then the user has to type any character except <CR> and Y. If the views are correct and <CR> is hit (answer No to the first question), then frames 1 to 4 will be repeated for all sub-objects as well as the main object. Entering Y in answer to the first question causes skipping of the display of any of the frames for subsequent sub-objects until the end, where the final decomposition tree is displayed in frame 5.

Frame 2 : After calculation of the wireframe from input views, its isometric view is shown again with input views in the form of a new frame, as seen in figure 8.2. Here

the 3D edges which are located on the boundary faces of the wireframe are shown in yellow, and the others are drawn in white.

Frame 3 : By finding the first approximation wireframe model, its isometric view is drawn in purple on top of the wireframe model of frame 2. Also instead of input views, 2D differences are selected and the result is displayed under frame 3, as in figure 8.3.

Frame 4 : Isometric view of 3D edge differences of two wireframe models together with the 2D views of next selected sub-object are shown in this frame (figure 8.4) .

Frame 5 : This is the last frame, which shows the decomposition of the object, as in figure 8.8.

8.2.3 IGES output file

The graphic parts of output are only visible on the screen of the workstation (Apollo DN3000). In order to use the visible section of the output in the thesis, a set of routines were developed to generate the same output in IGES format, which is a text file that can be read into other programs using the same standard. Therefore, if this option has been selected, an IGES format file is created for each frame with the names : iges1, iges2, . . ., igesn. Obviously the last file (igesn) belongs to the decomposition tree.

8.2.4 Error messages

To reflect the status of the program during execution , relevant messages are printed out. Some of them are as follows :

**** THIS FILE DOES NOT EXIST **** ; when a new file name is given for input

INPUT VIEWS ARE NOT CORRECT ; when visual check fails.

IS THE DIRECTION OF LOOPS IN "viewname" VIEW IS IN NON

STANDARD WAY ? (Y/N): ; this is asked only when arc edges appear in X-Z view.

INDEX OF ISOMETRIC VIEW > 1000 ; when the number of 3D edges exceeds limit.

THIS IS NOT CONSISTENT ==> ERROR; loops in different views don't match

8.3 Running different examples

8.3.1 Illustrative example

To demonstrate the program capabilities, different example have been selected. Because of the huge output, only for the following simple illustrative example is shown full detail :

```
$ inlib /lib/gmrlib
$ FTN ZDRAWINVIEW.FTN
no errors, no warnings in DRGINV, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in INIGMR, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in COLOR_ENTRY1, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in ERROR, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in COLOR_ENTRY, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in DRAWV, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in PLOTIG, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in TEXTIG, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in CIRCLIG, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in PROPERTY, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in DPIOUV, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in II2C, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
no errors, no warnings in R2D, Fortran version 9.66 1991/02/11 15:13:39 GMT (Mon)
```

```
$ linkx
LINKING CIEDSM VERSION Z10UNF & Z11
All Globals are resolved.
$ ciedsm
Type / for option or
```

input filename of the view xy

```
/
Do you want list of data ?
y
```

Do you want IGES output for graphics?

y
Type / for option or

input filename of the view xy

ex11

input filename of the view xz

ex12

input filename of the view yz

ex13


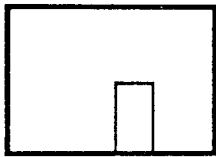
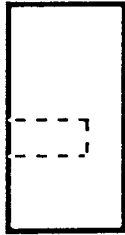
	
	<div>CONVERSION OF 2D DATABASES TO 3D By E.A.Allabad Only final result? type Y ARE THE VIEWS CORRECT? <i>Solid</i> _____ HIT <CR> FOR YES <i>Bdry</i> _____ <i>Dash</i> - - -</div>

Figure 8.1 Frame 1 : Input views for visual check

ANALYSING INPUT DATA

OBJECT 0 AT LEVEL 0

1. IDENTIFYING CLASS OF OBJECT

OBJECT IDENTIFIED AS * NON-PRISMATIC *
But the FAM is primitive : BLOCK

WIREFRAME MODEL SECTION

OBJECT 0 AT LEVEL 0

VIEW xy :

NUMBER OF NODES = 6

NUMBER OF EDGES = 7

NODE NO.	X	Y
1	0.100E+01	0.100E+01
2	0.300E+01	0.100E+01
3	0.400E+01	0.100E+01
4	0.400E+01	0.500E+01
5	0.100E+01	0.500E+01
6	0.100E+01	0.300E+01

EDGE NO. S.NODE->E.NODE TYPE

1	1	2	20000	LINE,BDRY
2	2	3	20000	LINE,BDRY
3	3	4	20000	LINE,BDRY
4	4	5	20000	LINE,BDRY
5	5	6	20000	LINE,BDRY
6	6	1	20000	LINE,BDRY
7	2	6	10001	CRCL,DASH

ARC NO X CENTER Y CENTER

1	0.300E+01	0.300E+01
---	-----------	-----------

VIEW xz :

NUMBER OF NODES = 8

NUMBER OF EDGES = 9

NODE NO.	X	Z
1	0.100E+01	0.100E+01
2	0.400E+01	0.100E+01
3	0.400E+01	0.700E+01
4	0.100E+01	0.700E+01
5	0.100E+01	0.500E+01
6	0.100E+01	0.400E+01
7	0.300E+01	0.400E+01
8	0.300E+01	0.500E+01

EDGE NO. S.NODE->E.NODE TYPE

1	1	2	20000	LINE,BDRY
2	2	3	20000	LINE,BDRY
3	3	4	20000	LINE,BDRY
4	4	5	20000	LINE,BDRY
5	5	6	20000	LINE,BDRY
6	6	1	20000	LINE,BDRY
7	5	8	10000	LINE,DASH
8	8	7	10000	LINE,DASH
9	7	6	10000	LINE,DASH

VIEW yz :

NUMBER OF NODES = 8
NUMBER OF EDGES = 9

NODE NO.	Z	Y
1	0.100E+01	0.100E+01
2	0.400E+01	0.100E+01
3	0.500E+01	0.100E+01
4	0.700E+01	0.100E+01
5	0.700E+01	0.500E+01
6	0.100E+01	0.500E+01
7	0.500E+01	0.300E+01
8	0.400E+01	0.300E+01

EDGE NO. S.NODE->E.NODE TYPE

1	1	2	20000	LINE,BDRY
2	2	3	20000	LINE,BDRY
3	3	4	20000	LINE,BDRY
4	4	5	20000	LINE,BDRY
5	5	6	20000	LINE,BDRY
6	6	1	20000	LINE,BDRY
7	3	7	0	LINE,SOL.
8	7	8	0	LINE,SOL.
9	8	2	0	LINE,SOL.

3D NODE COORDINATES

NODE NO.	X	Y	Z	NODE PIC IN VIEWS:		
				xy	xz	yz
1	0.100E+01	0.100E+01	0.100E+01	1	1	1
2	0.100E+01	0.100E+01	0.700E+01	1	4	4
3	0.100E+01	0.100E+01	0.500E+01	1	5	3
4	0.100E+01	0.100E+01	0.400E+01	1	6	2
5	0.300E+01	0.100E+01	0.400E+01	2	7	2
6	0.300E+01	0.100E+01	0.500E+01	2	8	3
7	0.400E+01	0.100E+01	0.100E+01	3	2	1
8	0.400E+01	0.100E+01	0.700E+01	3	3	4

9	0.400E+01	0.500E+01	0.100E+01	4	2	6
10	0.400E+01	0.500E+01	0.700E+01	4	3	5
11	0.100E+01	0.500E+01	0.100E+01	5	1	6
12	0.100E+01	0.500E+01	0.700E+01	5	4	5
13	0.100E+01	0.300E+01	0.500E+01	6	5	7
14	0.100E+01	0.300E+01	0.400E+01	6	6	8

2D COLINEAR EDGES ADDED TO EACH VIEW :

VIEW xy :

EDGE NO. S.NODE->E.NODE TYPE

1	1	2	20000	LINE,BDRY
2	2	3	20000	LINE,BDRY
3	3	4	20000	LINE,BDRY
4	4	5	20000	LINE,BDRY
5	5	6	20000	LINE,BDRY
6	6	1	20000	LINE,BDRY
7	2	6	10001	CRCL,DASH
8	1	3	20000	LINE,BDRY
9	1	5	20000	LINE,BDRY

VIEW xz :

EDGE NO. S.NODE->E.NODE TYPE

1	1	2	20000	LINE,BDRY
2	2	3	20000	LINE,BDRY
3	3	4	20000	LINE,BDRY
4	4	5	20000	LINE,BDRY
5	5	6	20000	LINE,BDRY
6	6	1	20000	LINE,BDRY
7	5	8	10000	LINE,DASH
8	8	7	10000	LINE,DASH
9	7	6	10000	LINE,DASH
10	1	5	20000	LINE,BDRY
11	1	4	20000	LINE,BDRY
12	6	4	20000	LINE,BDRY

VIEW yz :

EDGE NO. S.NODE->E.NODE TYPE

1	1	2	20000	LINE,BDRY
2	2	3	20000	LINE,BDRY
3	3	4	20000	LINE,BDRY
4	4	5	20000	LINE,BDRY
5	5	6	20000	LINE,BDRY
6	6	1	20000	LINE,BDRY
7	3	7	0	LINE,SOL.
8	7	8	0	LINE,SOL.
9	8	2	0	LINE,SOL.
10	1	3	20000	LINE,BDRY
11	1	4	20000	LINE,BDRY
12	2	4	20000	LINE,BDRY

3D EDGE LIST

EDGE NO S.NODE->E.NODE TYPE EDGE PIC IN VIEWS :

EDGE NO	S.NODE	E.NODE	TYPE	xy	xz	yz
1	3	6	20000	1	7	0
2	4	5	20000	1	9	0
3	7	9	20000	3	0	6
4	8	10	20000	3	0	4
5	9	11	20000	4	1	0
6	10	12	20000	4	3	0
7	13	3	20000	6	0	7
8	14	4	20000	6	0	9
9	5	14	10001	7	9	9
10	6	13	10002	7	7	7
11	1	7	20000	8	1	0
12	2	8	20000	8	3	0
13	1	11	20000	9	0	6
14	2	12	20000	9	0	4
15	7	8	20000	0	2	11
16	9	10	20000	0	2	5
17	2	3	20000	0	4	3
18	3	4	20000	0	5	2
19	13	14	20000	0	5	8
20	4	1	20000	0	6	1
21	6	5	20000	0	8	2
22	11	12	20000	0	11	5

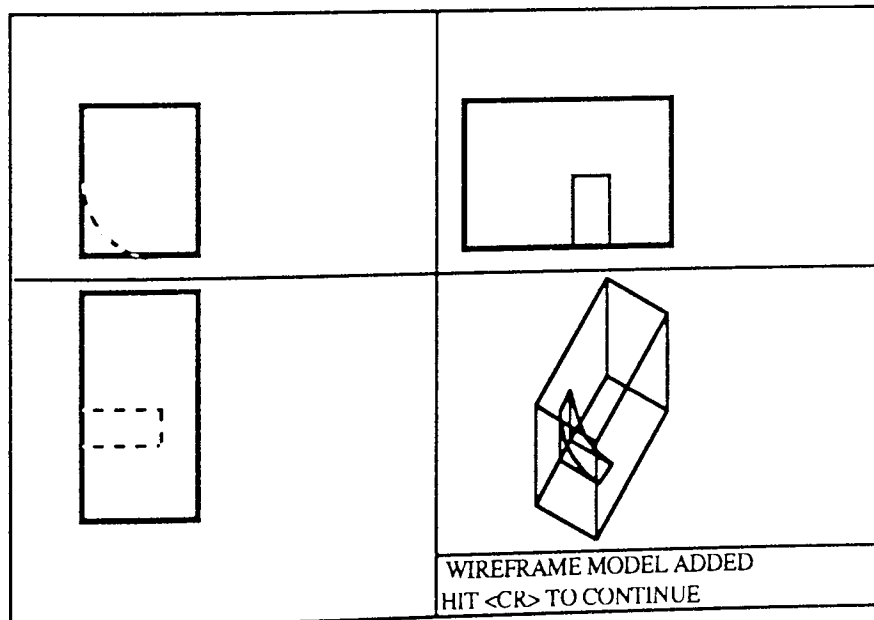


Figure 8.2 Frame 2 : Input view and wireframe model

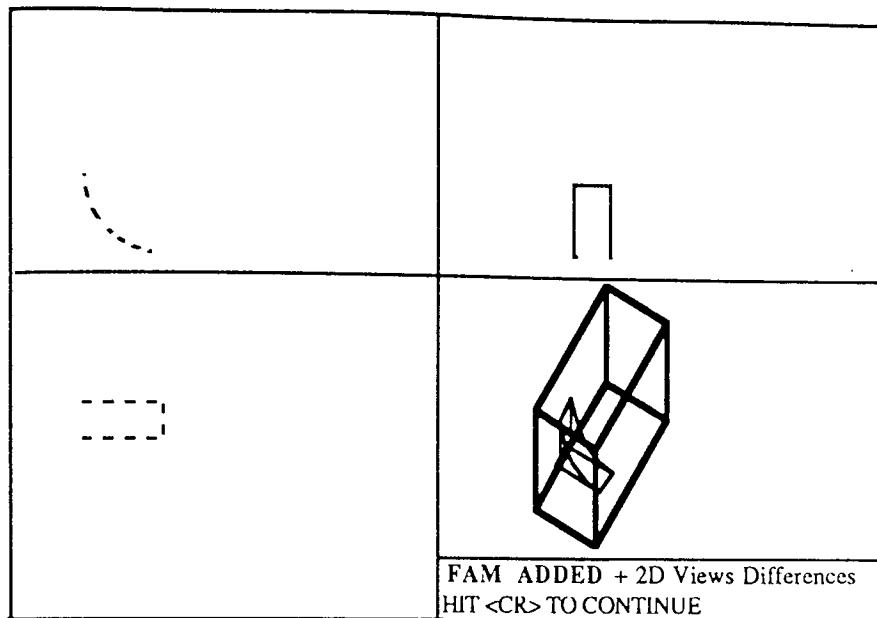


Figure 8.3 Frame 3 : First approximation wireframe model on top of W.M.

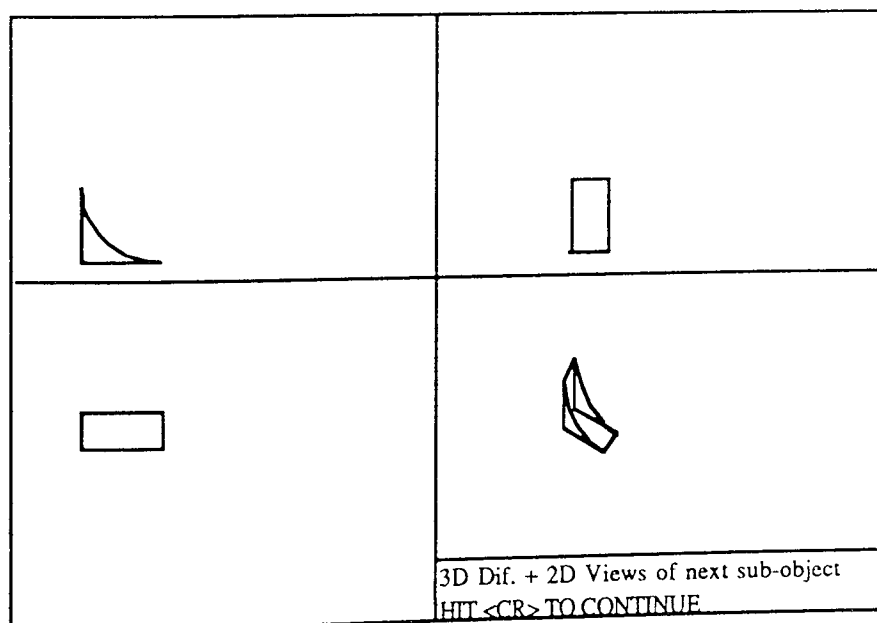


Figure 8.4 Frame 4 : 3D edge differences and orthographic views of next sub-object




	
	<p>CONVERSION OF 2D DATABASES TO 3D</p> <p>By E.A.Allabad</p> <p>Only final result? type Y</p> <p>ARE THE VIEWS CORRECT? <i>Solid</i> ____</p> <p>HIT <CR> FOR YES <i>Bdry</i> ____ <i>Dash</i> - -</p>

Figure 8.5 Frame 1 for sub-object 1

ANALYSING INPUT DATA

OBJECT 1 AT LEVEL 1

1. IDENTIFYING CLASS OF OBJECT

OBJECT IDENTIFIED AS A * PRIMITIVE *
(FILLET)

2. IDENTIFYING 3D PRIMITIVES

WIREFRAME MODEL SECTION

OBJECT 1 AT LEVEL 1

VIEW xy :

NUMBER OF NODES = 3

NUMBER OF EDGES = 3

NODE NO.	X	Y
1	0.100E+01	0.100E+01
2	0.300E+01	0.100E+01
3	0.100E+01	0.300E+01

EDGE NO. S.NODE->E.NODE TYPE

1	1	2	20000	LINE,BDRY
2	3	1	20000	LINE,BDRY
3	2	3	20001	CRCL,BDRY

ARC NO X CENTER Y CENTER

1	0.300E+01	0.300E+01
---	-----------	-----------

VIEW _{xz} :

NUMBER OF NODES = 4

NUMBER OF EDGES = 4

NODE NO. X Z

1	0.100E+01	0.500E+01
2	0.300E+01	0.500E+01
3	0.100E+01	0.400E+01
4	0.300E+01	0.400E+01

EDGE NO. S.NODE->E.NODE TYPE

1	2	1	20000	LINE,BDRY
2	1	3	20000	LINE,BDRY
3	3	4	20000	LINE,BDRY
4	4	2	20000	LINE,BDRY

VIEW _{yz} :

NUMBER OF NODES = 4

NUMBER OF EDGES = 4

NODE NO. Z Y

1	0.500E+01	0.100E+01
2	0.500E+01	0.300E+01
3	0.400E+01	0.300E+01
4	0.400E+01	0.100E+01

EDGE NO. S.NODE->E.NODE TYPE

1	1	2	20000	LINE,BDRY
2	2	3	20000	LINE,BDRY
3	3	4	20000	LINE,BDRY
4	4	1	20000	LINE,BDRY

3D NODE COORDINATES

NODE NO.	X	Y	Z	NODE PIC IN VIEWS:		
				xy	xz	yz
1	0.100E+01	0.100E+01	0.500E+01	1	1	1
2	0.100E+01	0.100E+01	0.400E+01	1	3	4

3	0.300E+01	0.100E+01	0.500E+01	2	2	1
4	0.300E+01	0.100E+01	0.400E+01	2	4	4
5	0.100E+01	0.300E+01	0.500E+01	3	1	2
6	0.100E+01	0.300E+01	0.400E+01	3	3	3

2D COLINEAR EDGES ADDED TO EACH VIEW :

VIEW xy :
NONE

VIEW xz :
NONE

VIEW yz :
NONE

3D EDGE LIST

EDGE NO S.NODE->E.NODE TYPE EDGE PIC IN VIEWS :

EDGE NO	S.NODE->E.NODE	TYPE	xy	xz	yz
1	1 4	20000	1 1	0	
2	2 4	20000	1 3	0	
3	5 1	20000	2 0	1	
4	6 2	20000	2 0	3	
5	3 5	20001	3 1	1	
6	4 6	20002	3 3	3	
7	1 2	20000	0 2	4	
8	5 6	20000	0 2	2	
9	4 3	20000	0 4	4	

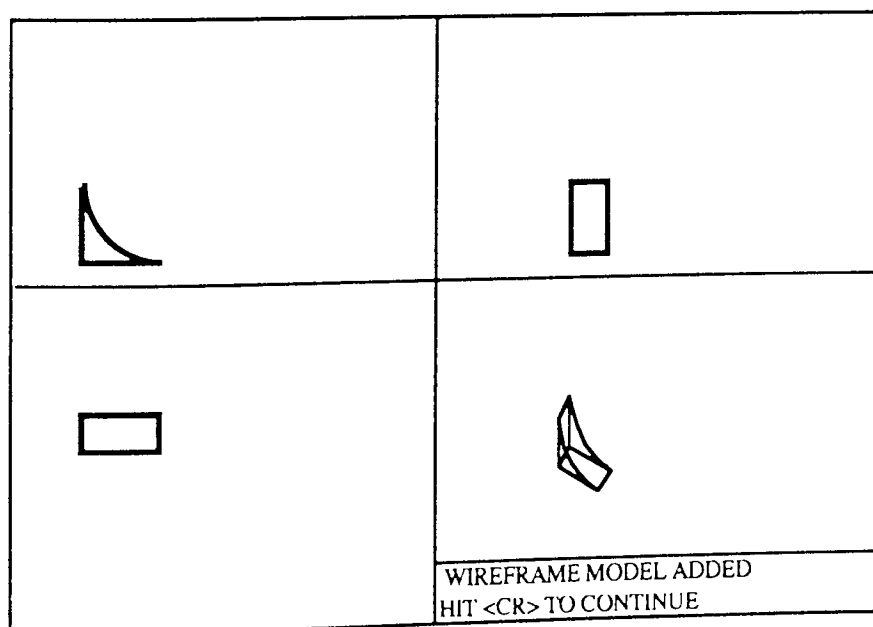


Figure 8.6 Frame 2 for sub-object 1

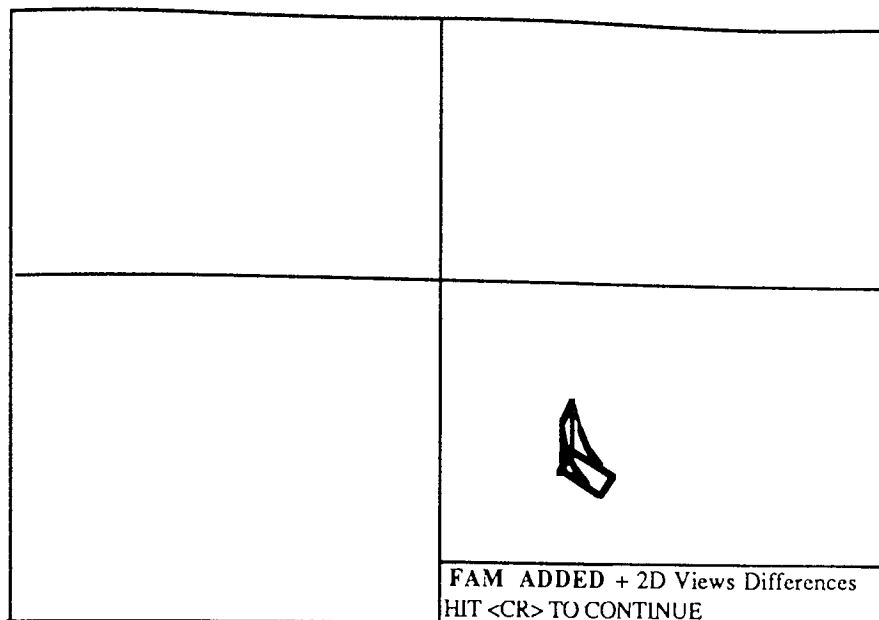


Figure 8.7 Frame 3 for sub-object 1

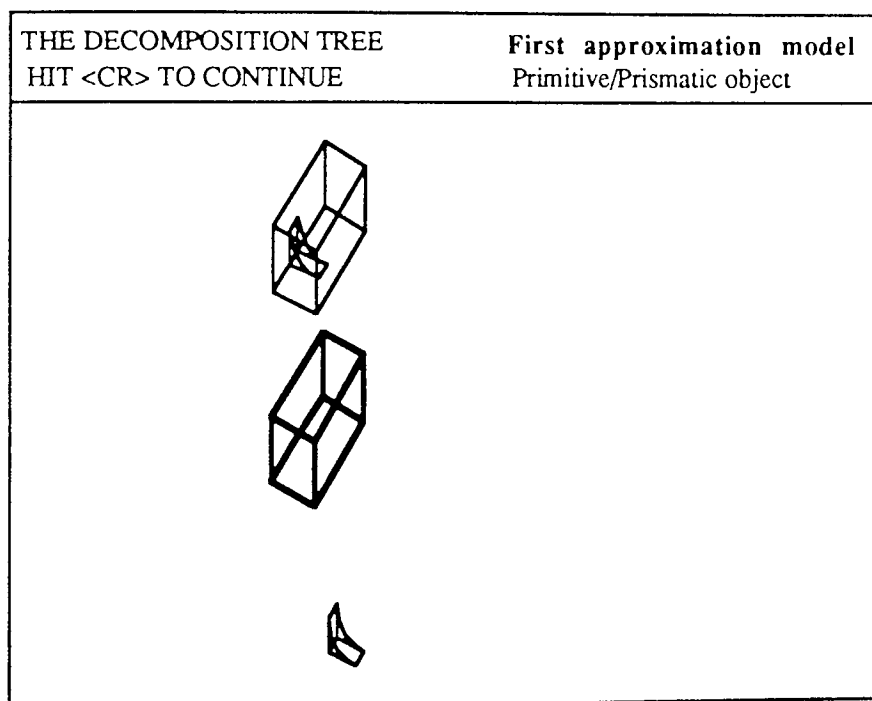


Figure 8.8 Frame 5 : Decomposition tree

CONTENTS OF CREATED BOXER FILE

```

SCOPE COMMON
  XYL0001<- BLOCK( 3.00, 4.00, 6.00) AT ( 2.50, 3.00, 4.00)
F00 <- XYL0001
  XYL0101<- ^SFIL1( 1.00, 1.00, 4.00, 3.00, 1.00, 4.00, 1.00, 3.00 $
, 4.00,0,0,0, 1.00,0,-1)
F01 <- XYL0101
F02 <- F00 - F01
OBJ <- F02
Fortran STOP

```

8.3.2 Program testing

The above file is executed by the BOXER program on VAX and the solid object created by it is captured on the Macintosh screen, by emulating the Tektronix 4014, as illustrated in figure 8.9. To verify the functioning of the implemented program, once again the solid modeller is asked to produce the first angle orthographic views of the built object, the result of which is shown in figure 8.10. Comparing this output with the original input of figure 8.1 shows the correct acting of the program for this example. The only differences are the presence of tangency lines in input views, which have been added for the purpose of specifying tangency vertices.

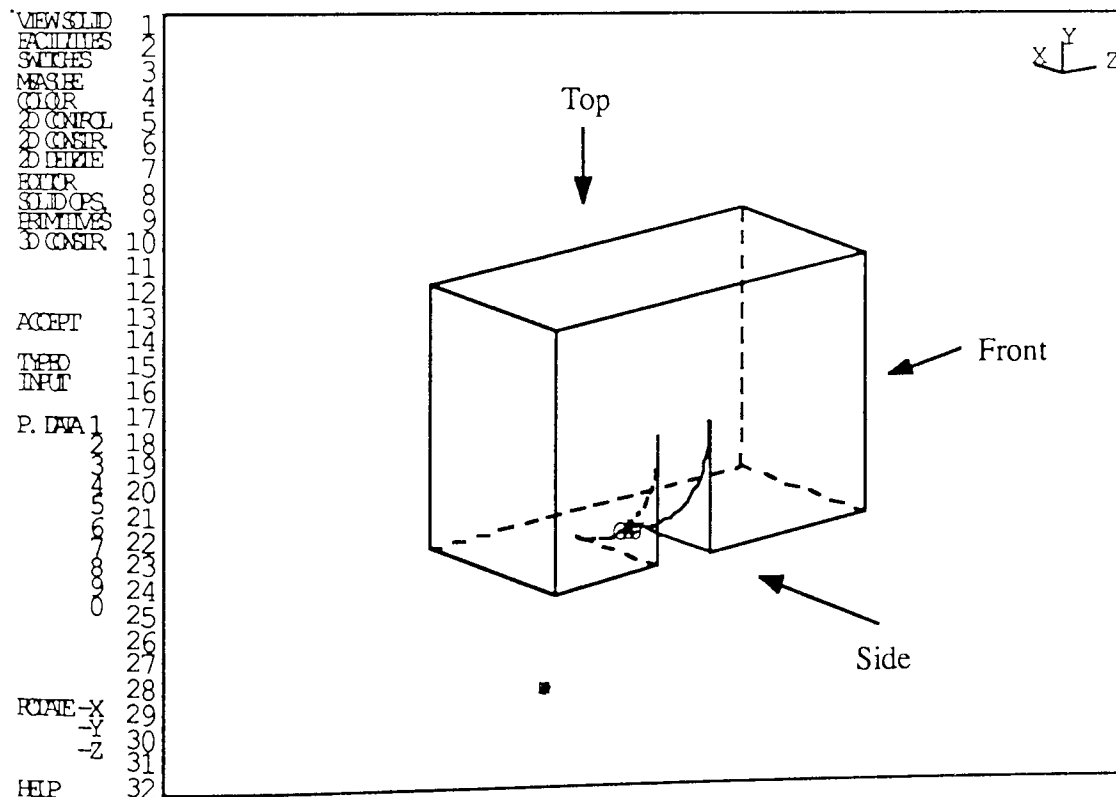


Figure 8.9 Solid object built by solid modeller

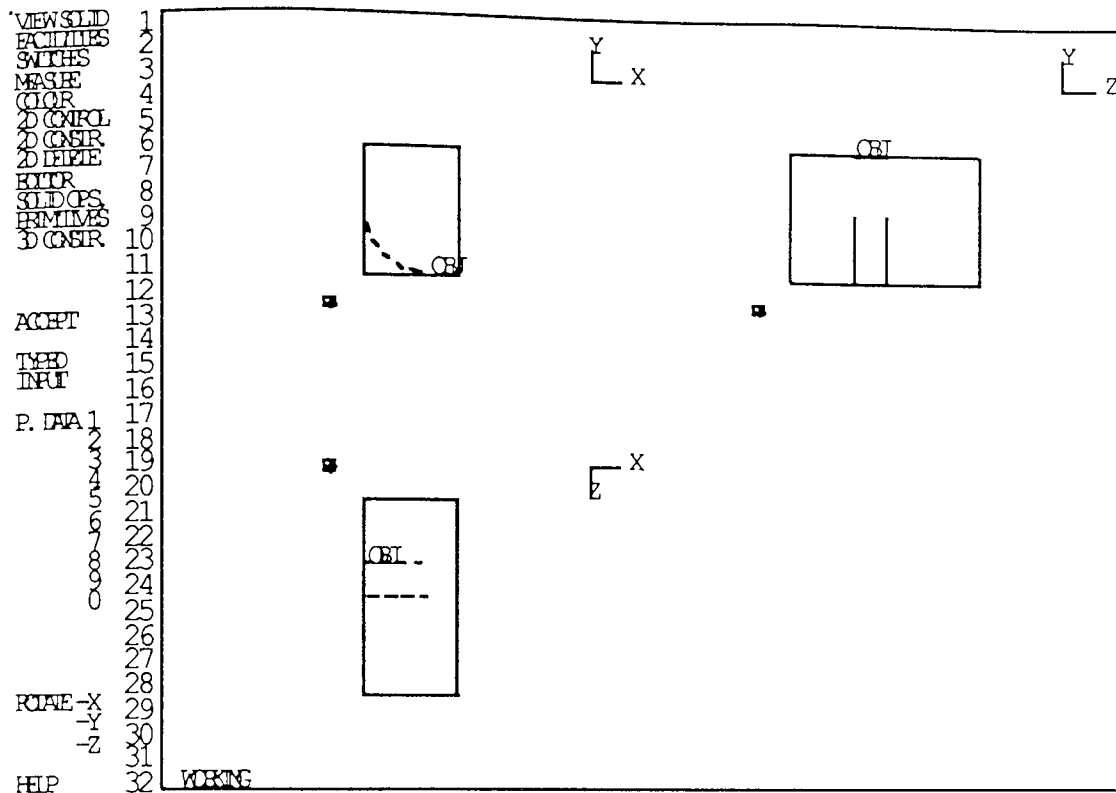


Figure 8.10 First angle orthographic views of the produced object by solid modeller

8.4 Discussion

In the last section a simple example was selected to demonstrate the functionality of the program. For complex examples with the print option selected, the amount of output generated will be very huge, especially if the example has to reach the solution in many iterations. Since this example has a sub-object it has 2D and 3D differences and does not end at frame 3 (figure 8.3). So after completion of the first stage, it starts the second stage, with the extracted 2D views of the sub-object (figure 8.5). This time, because there are no differences between the wireframe and the first approximation wireframe in frame 3 (figure 8.7), the process is stopped and the decomposition tree is displayed (figure 8.8), followed by the contents of solid modeller command file (BOXER.DAT). More examples are run by the program and results are presented as appendix A. To save time and space, only original input views, the wireframe model of the built object produced by solid modeller (with and without dashed lines removed),

and decomposition tree generated by the program are shown for each example. When the decomposition tree is created, then from bottom to top, using leaves of the tree which are solids, the CSG tree can be build. Each example is briefly explained below:

Example 1: This is an example of an orthoprismatic object processed by the program. Since the first approximation model and object's wireframe are the same, the expansion tree consists of only one item.

Example 2: This is a simple nonprismatic object.

Example3: Figure A.3 shows for the case where dashed line information is used, which gives an acceptable solution; figure A.4 uses the same conditions, except that the dashed line information is ignored. Comparing these two output models shows how effective consideration of dashed line information can be. It should be mentioned that in this example the total boundary loops, which are views resulting from 3D differences, are not completed and only the middle loops of the views are considered. Therefore, only one level of recursion is reached. In this case we can also reach the final solution by comparing the orthographic views produced from the model with the input views (figure A.3). In the next example it is shown that if the recursion is continued, the final output model will be achieved, without calling upon the solid modeller.

Example 4: This is an object which requires more than one level of recursion to achieve the correct output model.

Example 5: This is a simple example for a sphere primitive.

Example 6: This is a simple example for a cylinder primitive.

Example 7: This is a prismatic object pierced by holes. Since it has been identified as a

prismatic object by the FRSTAPM subroutine, and the instruction for its production by the solid modeller includes holes, its expansion tree consists of only one item.

Example 8: This shows an orthoprismatic object with holes. This time the holes are represented by separate objects which must be taken from the first approximation model. This means that, unlike example 7, the first approximation model does not contain any holes.

Examples 9-11: These are additional examples for nonprismatic objects.

Chapter Nine

SUGGESTIONS FOR FURTHER WORK

Chapter Nine

SUGGESTIONS FOR FURTHER WORK

9.1 Discussion

Building a solid object from its 2D engineering drawings is a formidable task because vital information is routinely omitted from the orthographic views. The task becomes even more complex when the vast domain of mechanical engineering components are considered. Many other researchers have tackled this problem in different ways. Each has been successful in handling some classes of objects. None of them has yet proposed a fully automatic method to cover a wide range of objects. Recently, a new method has been developed by A. Kargas, P. Cooley and T.H.E. Richards, which produces a first approximation model using constructive solid geometry concepts. Although this model in many cases does not correspond to a real object, because of the many advantages of this method over others, the current project has been undertaken to improve the deficiencies of the method and refine its output model.

The basic idea prescribed here for tackling the problem is: since the object is bounded by its first approximation model the differences between the object and this model is a set of volumes or masses called sub-objects. This is similar to casting an object, where the volume of the surrounding frame is the first approximation model, the mould material is the object, and the cast materials are sub-objects. In this scheme, two items were required to find the object. One was the first approximation model, which was prepared by simplifying and changing the original package developed in the last project into subroutine form. Sub-objects are another item that must be found. To reach this

goal, an easy and fast method that could lead the program to extract these items was selected, and that was to find two wireframe models, apart from their ghost edges and nodes. Wireframe representation of input views and its first approximation model constitute these items. Since the first approximation model computed by the subroutine consisted only of commands for a solid modeller, it first had to run the solid modeller to build the model, and then request the calculation of wireframe, which was also not in acceptable form for the program. Therefore, this process was time consuming and could be a disadvantage to the program. To solve this problem, a new efficient algorithm was developed for this special case (first approximation model), where many common features exist between the two wireframes, e.g. 3D node coordinates.

When these items are ready the 3D edges which are not common between the two models are identified. Clearly, these differences are due to the sub-objects. This means that these edges constitute part of the sub-objects. The other parts are sought among boundary elements. At this stage, two other points are considered: the effect of ghost edges during the process, and the method of selecting the boundary elements to complete the wireframe model of sub-objects.

It has been shown that cases of ghost edges being created on the boundary, do not produce any problem. Because the two wireframes are identically calculated therefore, ghost edges do not appear in 3D differences. Also, it is seen that if 3D edge differences are mapped into 2D and used to produce the wireframe of sub-objects, then the ghost edge phenomenon is reduced to some extent. Completing the boundary of each view for sub-objects within a matching box is also done in 2D, and it is shown that using the dashed line information can be helpful in many cases. In such a situation, if sub-objects are actually separate elements within the first approximation model, then their 3D wireframe difference appeared as a separate set of connected edges.

After finding 2D views for each sub-object they are then treated as main objects, and

these procedures are repeated recursively. In any of these cases, if a set of 2D views represents primitive signatures, prismatic objects or if its first approximation model has a volume less than a specified amount, then the process for that particular item or branch of tree is terminated. It is deduced that in each step the object is broken into many simplified sub-objects and first approximation models. The tree representing this process is called the decomposition or expansion tree, which is similar to the expansion of a function in the form of a power series.

9.2 Suggestions for Further Work

It is not easy to find a final solution for any research problem. Always new ideas are initiated and old techniques are replaced by enhanced newer ones, using other experiences and newly emerging facilities. In this project a new concept is used to improve the first approximation model, which has been produced by applying a novel technique to an orthographic view.

Although the implemented program shows that the new method can improve the model, much work must still be done to attain an advanced commercial package. Some areas remaining for future work are discussed here:

9.2.1 Input section

The ideal case for the input section is to have a system to accept the drawings and provide a data file with the suitable format, required by the program. This process involves the scanning of 2D views and conversion of the drawings from raster to vector form. Different techniques of image processing, pattern and character recognition, and expert systems may be employed to interpret figures, symbols,

recognize line styles, edge types and so on. A good system must be able to check the dimensions with the geometric values, and decide on the correct items by referring to different views.

Another subject in this section is the ability to check and correct input data. For example, tangency lines and vertices may be detected automatically and added to the input data. Also checks should be included for incomplete edges or extra lines on the drawings, caused by poor quality drawings or badly scanned data. To correct the input drawings, a good interactive graphic editor is recommended.

9.2.2 Increasing the object range

To increase the range of objects which can be manipulated by the program, different sections must be altered:

The data structures of the algorithm must be able to handle different types of 2D and 3D edges. Since most mechanical parts are made by special tools, they normally have a regular shape. Therefore, accepting circular, elliptic, parabolic, and hyperbolic arc type edges, rather than free form curve, is quite adequate to respond to these requests. The current status of the program is such that it can only accept these different curved type edges in 2D input.

3D edge generation is the main stage at which transformation from 2D to 3D takes place. Regardless of producing true or ghost edges, this is where the edges of 3D items including primitives are built. Therefore, here is one of the key areas where the capability of the program by extending the range of input, can be improved. The appropriate parts of the program must be altered to generate quadric 3D curves from corresponding 2D edges in different views.

Freedom of the number of views is another criteria. Sometimes two views are enough to represent a simple object, while in complex cases, in addition to many orthographic views, a cross section and auxiliary views may also be needed. Therefore the program must be flexible so that it can complete its information regarding the geometry and topology from as many views as are available. To achieve multiple view choice, the input section must also be modified.

The number of primitives and their orientations are also very important, not only in increasing the range of input, but also since they affect the number of boolean operations in the final CSG tree, to produce the output model. In the current project cones, spheres, hemi spheres and sectors have been added to the set of primitives detectable by the program. Some others, like toroids and pyramids, must still be added to the set. Also, primitives are currently recognised from their 2D signature, and they are assumed to be aligned with a coordinate axis. This restriction on the direction of the primitive can lead to the bigger CSG trees. To remove this constraint, the suggestion is made that it would be easier to recognise primitives from their 3D wireframe, regardless of the orientation.

9.2.3 Implementation

Some precautions must be taken or checks made for exceptional cases in the implementation process. For example, an object may be such that if a first extracted sub-object is processed again, the newly found sub-object may be the same as the object. This would cause an infinite loop, which must be detected and avoided.

Another problem which has not yet been solved completely is that of selecting the 2D boundary elements to complete the sub-object views. This needs more mathematical

study of the problem. The subject of ghost edges is also another item that merits further review in order to widen the scope of the cases that can be processed.

9.2.4 CSG tree optimization's

Every effort must be made to minimise the number of boolean operations, in the final CSG tree, in order to reduce the solid modeller processing time. Some steps have already been taken, as in the selection of the initial primitive, rather than simply choosing a cuboid as a raw block.

Chapter Ten

CONCLUSION

Chapter Ten

CONCLUSION

Modern design techniques within computer integrated manufacturing (CIM) mainly use constructive solid geometry (CSG) models for defining objects, and gradually traditional methods of design are becoming obsolete. In this period of transition, an automatic procedure which can convert engineering drawings into the 3D solid models used in the new technology will be a vitally important tool for industry. Hence, problems of redesigning or entering large amounts of data for existing designed parts can be avoided.

A new technique of expanding objects has been developed in this project, using the first approximation model produced by recent work [Kargas, Cooley & Richards, 1988]. Finding a way of improving the first approximation model produced in this regard, gives a great advantage to this method of transferring 2D engineering drawings into 3D solids, based on CSG concepts. The method proposed and implemented in this project is designed in such a way that it can easily be extended to handle a vast range of objects in an efficient way. Its use of most information produced during the process, and ability to avoid unnecessary calculations (i.e. 3D nodes and edges for consecutive sub-objects), gives the program the ability to cope with complex objects more efficiently. A novel use of dashed line information for any 2D edge type is employed to aid decision making when solving the problem.

Since the algorithm breaks down the object to simpler ones until generating a solid primitive or prismatic object, the backtracking problem has been reduced considerably. Briefly, the idea of decomposing objects into a hierarchy of the first approximation models, coupled with the use of dashed line information provides a powerful method for conversion of 2D engineering drawings into 3D solid models with many advantages over existing techniques.

References

References

Aldefeld, B. (1983), "On Automatic Recognition of 3D Structures from 2D Representations ", *Computer-Aided Design*, **15**(2), pp.59-64.

Aldefeld, B. & Richter, H. (1984), "Semiautomatic Three-Dimensional Interpretation of Line Drawings", *Computers and Graphics (GB) J.*, **8**(4), pp. 371-380.

Apollo Computers Inc., (1986), "Programming with 2D Graphic Meta File Resources: Software version 9.5", USA.

Baer, A., Eastman, C. & Henrion, M. (1979), "Geometric Modelling : A Survey", *Computer-aided Design J.*, **11**(5), pp.253-271

Baumgart, B.G. (1975), "A Polyhedron Representation for Computer Vision", in : *AFIPS Proc. Nat. Comp. Conf. 44*, pp.589-593.

Bezier, P. (1972), *Numerical Control: Mathematics and Applications*, John Wiley & Sons Ltd., New York.

Bezier, P. (1974), "Mathematical and Practical Possibilities of UNISURF", in Barnhill, R.E. & Riesenfeld (Eds), *Computer Aided Geometric Design*, Academic Press, New York.

Bin, Ho (1986), "Inputting Constructive Solid Geometry Representation Directly from 2D Orthographic Engineering Drawings", *Computer-aided Design J.*, **18**(3) pp. 147-155.

Bloor, M.S., Dodsworth, R. & Owen, J. (1984), *Computer Aided Design Interchange of Data: Guideline for the Use of IGES*, London:NEDO.

Braid, I.C. (1986), "Geometric Modelling", in Enderle, G., Grave, M. & Lillehagen, F. (eds), *Advances in Computer Graphics I*, Berlin: Springer-Verlag, pp.325-361.

Braid, I.C. & Lang, C.A. (1973), "Computer-Aided Design of Mechanical

Components with Building Blocks", *Proceedings of the Second IFIP/IFAC Int. Conf. on Programming Languages for Machine Tools*, Budapest:PROLAMAT73, pp. 109-118.

Coons, S.A. (1964), "Surfaces for Computer-aided Design of Space forms", MIT MAC-TR-41, Cambridge, MA.

Coons, S.A. (1974), "Surface Patches and B-spline Curves", pp. 1-16 in Barnhill, R.E. & Riesenfeld, R.F. (eds), *Computer Aided Geometric Design*, Academic Press, New York.

Davies, B.L. & Yarwood, A. (1986), *Engineering Drawing and Computer Graphics*, Reinhold(U.K.):Van Nostrand.

Ferguson, J.C. (1964), "Multivariate curve interpolation", *J. Assoc. Comput. Mach.* vol 11, pp221-228.

Field, D.A. (1987), "Mathematical Problems in Solid Modelling: A Brief Survey", in Farin, G.E.(ed), *Geometric Modelling : Algorithms and new Trends*, SIAM, pp.91-105.

Forrest, A.R. (1972), "Interactive interpolation and approximation by Bezier polynomials", *Computer J.*, vol.11, pp. 71-79.

Goldman, R.N. (1987), "The Role of Surfaces in Solid Modelling", in Farin, G.E.(ed), *Geometric Modelling : Algorithms and new Trends*, SIAM, pp.69-90.

Gordon, W.J. (1969), "Spline-blended Surface Interpolation through curve Networks", *J.Math. Mech. E.*, pp. 931-952.

Gu, Kaining , Tang, Z. & Sun, J. (1985), "Reconstruction of 3D Objects from Orthographic Projections", *Proceedings of COMPINT85 Computer aided Technologies*, pp. 807-811.

Idesawa, M. (1973), "A System to Generate a Solid Figure from Three View", *Bulletin of the JSME*, 16(92), PP. 216-225 .

Idesawa, M., Soma, T., Goto, E. & Shibata, S. (1975), "Automatic Input of Line

Drawing and Generation of a Solid Figure from Three-View Data", *Proceedings of the International Joint Computer Symposium*, pp. 304-311.

Iwata, K., Yamamoto, M. & Iwasaki, M. (1988), "Recognition System for Three-view Mechanical Drawings", *Pattern Recognition, 4th International Conference*, Cambridge, March 28-30, pp. 240-249.

Kargas, A. (1988), "Computer Interpretation of Engineering Drawings as Solid Models", Ph.D. thesis, Aston University, Birmingham, U.K.

Kargas, A., Cooley, P. & T. H. E. Richards, T.H.E. (1988), "Interpretation of Engineering Drawings as Solid Models", *Computer-Aided Engineering J.*, April 1988, pp.67-78.

Lafue, G. (1976), "Recognition of Three-Dimensional Objects from Orthographic Views ", *Proceedings 3rd Annual Conference on Computer Graphics, Interactive Techniques and Image Processing*, ACM/SIGGRAPH, pp.103-108.

Lequette, R. (1988), "Automatic Construction of Curvilinear Solids from Wireframe Views", *Computer-Aided Design*, 20(4), pp.171-180.

Light, R.A. & Gossard, D.C. (1982), "Modification of Geometric Models through Variation Geometry", *Computer-Aided Design*, 14(4), pp.209-214.

Markowsky, G. & Wesley, M.A. (1980), "Fleshing Out Wire Frames", *IBM J. Res. Develop.*, 24(5), pp. 582-597.

Mortenson, M.E., (1985), *Geometric Modelling*, John Wiley & Sons, Inc., United States of America, New York.

Pafec Ltd., (1984), "DOGS user's manual level 3.2", Nottingham, U.K.

Pratt, M.J. (1984), "Solid Modelling and the Interface between Design and Manufacture", *IEEE Computer Graphics and Application*, 4(4), pp.52-59.

Preiss, K. (1981), "Algorithms for Automatic Conversion of a 3-View Drawing of a Plane-Faced Part to the 3D Representation", *Computer in Industry*, 12(2), pp.133-139.

Preiss, K. & Kaplansky, E. (1983), "Solving CAD/CAM Problems by Heuristic Programming", *Computer in Mechanical Engineering*, Sept. 1983, pp. 56-60.

Requicha, A.A.G. (1977), "Mathematical Models of Rigid Solid Objects", Technical Memo No-28, Production Automation Project, University of Rochester, Rochester, NY.

Requicha, A.A.G. (1980), "Representations for Rigid Solids: Theory, Methods, and Systems", *Computing Surveys*, **12**(4), pp.437-464.

Requicha, A.A.G. & Voelcker, H.B. (1982), "Solid Modelling: A Historical Summary and Contemporary Assessment", *IEEE Computer Graphics and Appls.*, **2**(2), pp.9-24.

Requicha, A.A.G. & Voelcker, H.B. (1983), "Solid Modelling: Current Status and Research Directions", *IEEE Comp. Graphics & App.*, **3**(7), pp.25-37.

Riesenfeld, R.F. (1973), "Applications of B-spline approximation to geometric problems of computer aided design", Ph.D. thesis, Dept. Systems and Information Science, Syracuse Univ., Syracuse, NY.

Sakurai, H. & Gossard, D.C. (1983), "Solid Model Input Through Orthographic Views", *Computer Graphics J.*, **17**(3), pp. 243-252.

Sutherland, I. (1974), "Three-Dimensional Data Input by Tablet", *Proc. IEEE* **62**(4), pp.453-461.

Thornton, R.W. (1978), "Interactive modelling in Three Dimensions Through Two-Dimensional Windows", *3rd Int. Conf. on Computers in Engineering and Building Design*, BRIGHTON, pp.204-211.

Vossler, D.L. (1985), "Sweep-to-CSG Conversion Using Pattern Recognition Techniques", *IEEE Computer Graphics and Appls.*, **5**(4), pp. 61-68.

Wesley, M.A. & Markowsky, G. (1981), "Fleshing Out Projections", *IBM J. Res. Develop.*, **25**(6), pp. 934-953.

Wesley, M.A. & Markowsky, G. (1984), "Generation of Solid Models from Two-

Dimensional and Three-dimensional data", in Pickett, M.S. & Boyse, J.W. (Eds), *Solid Modelling by Computers: from Theory to Applications*, New York: Plenum, pp. 23-50.

Woo, T.C. & Hammer, J.M. (1977), "Reconstruction of Three-Dimensional Designs from Orthographic Projections", *Proceedings of 9th CIRP Conference*, ENGLAND: Cranfield Institute of Technology, pp.247-255.

Xie, Shun-en & Calvert, T.W. (1988), "CSG-EESI: A New Solid Representation Scheme and a Conversion Expert System", *IEEE Transactions on Pattern and Machine Intelligence*, **10**(2), pp.221-234.

APPENDICES

APPENDICES

A. EXAMPLES

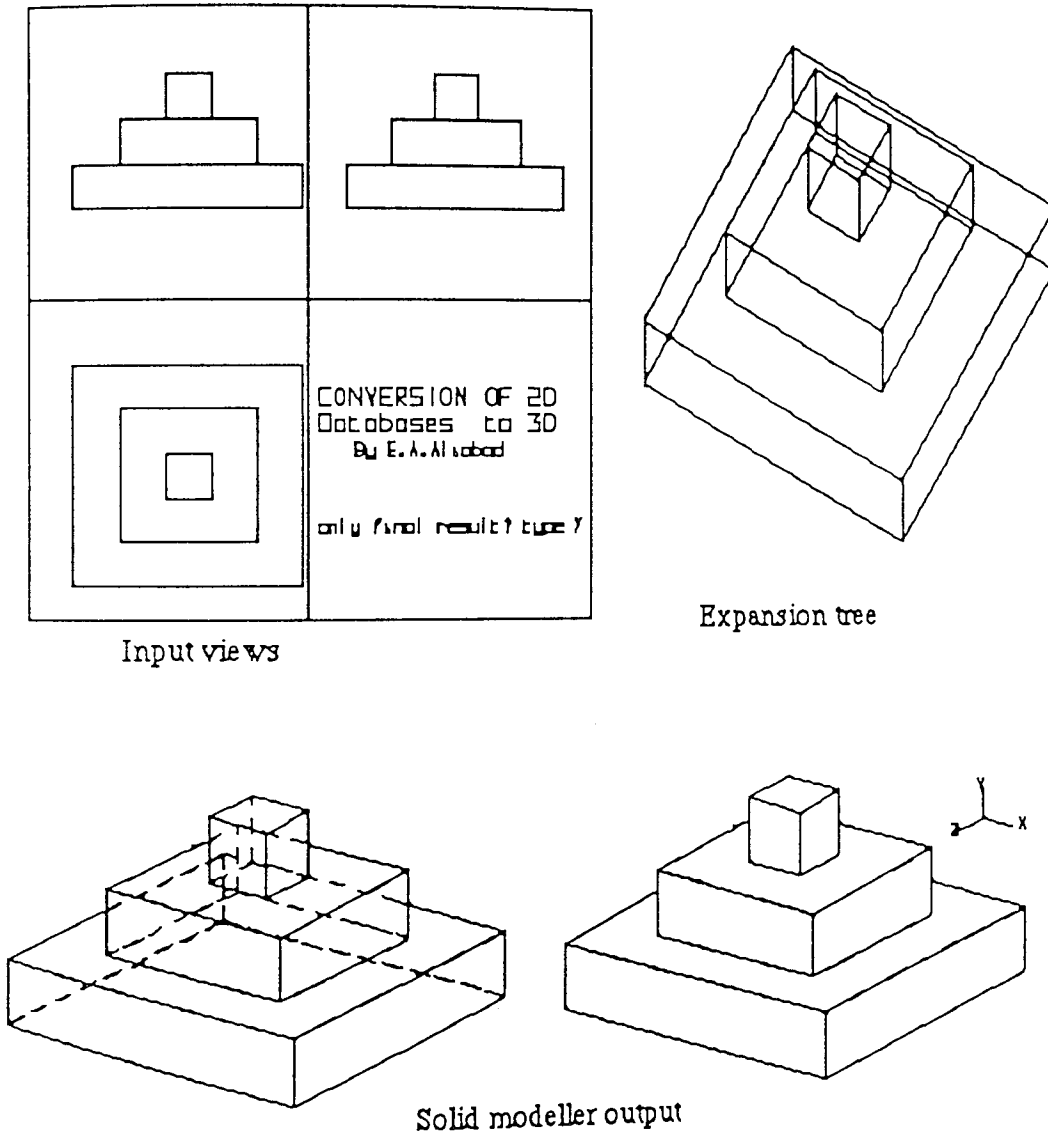
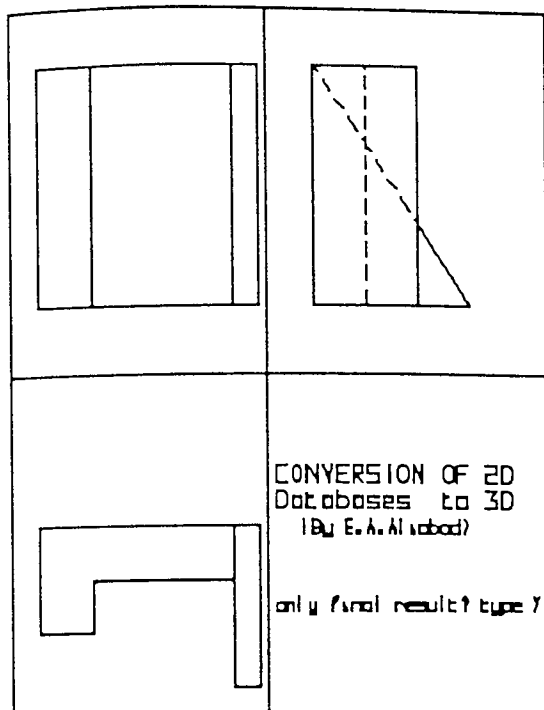
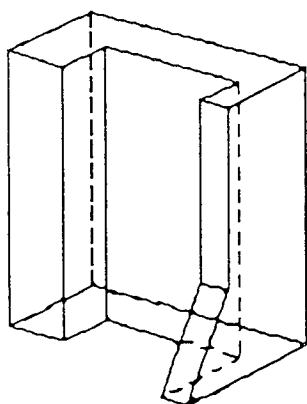


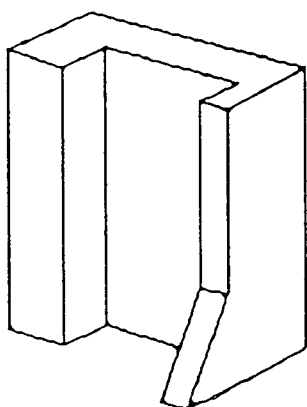
Figure A.1 Example 1



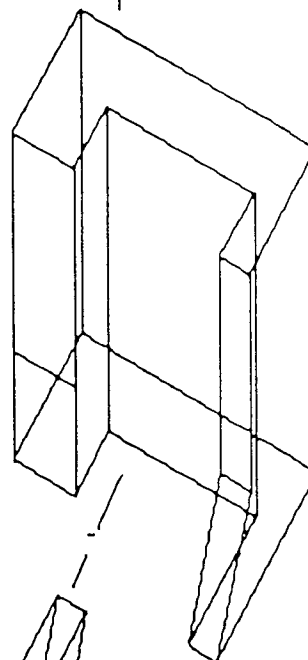
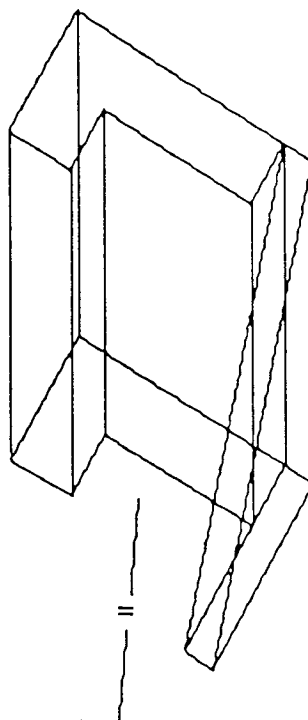
Input views



Solid modeller output



Solid modeller output



Expansion tree

Figure A.2 Example 2

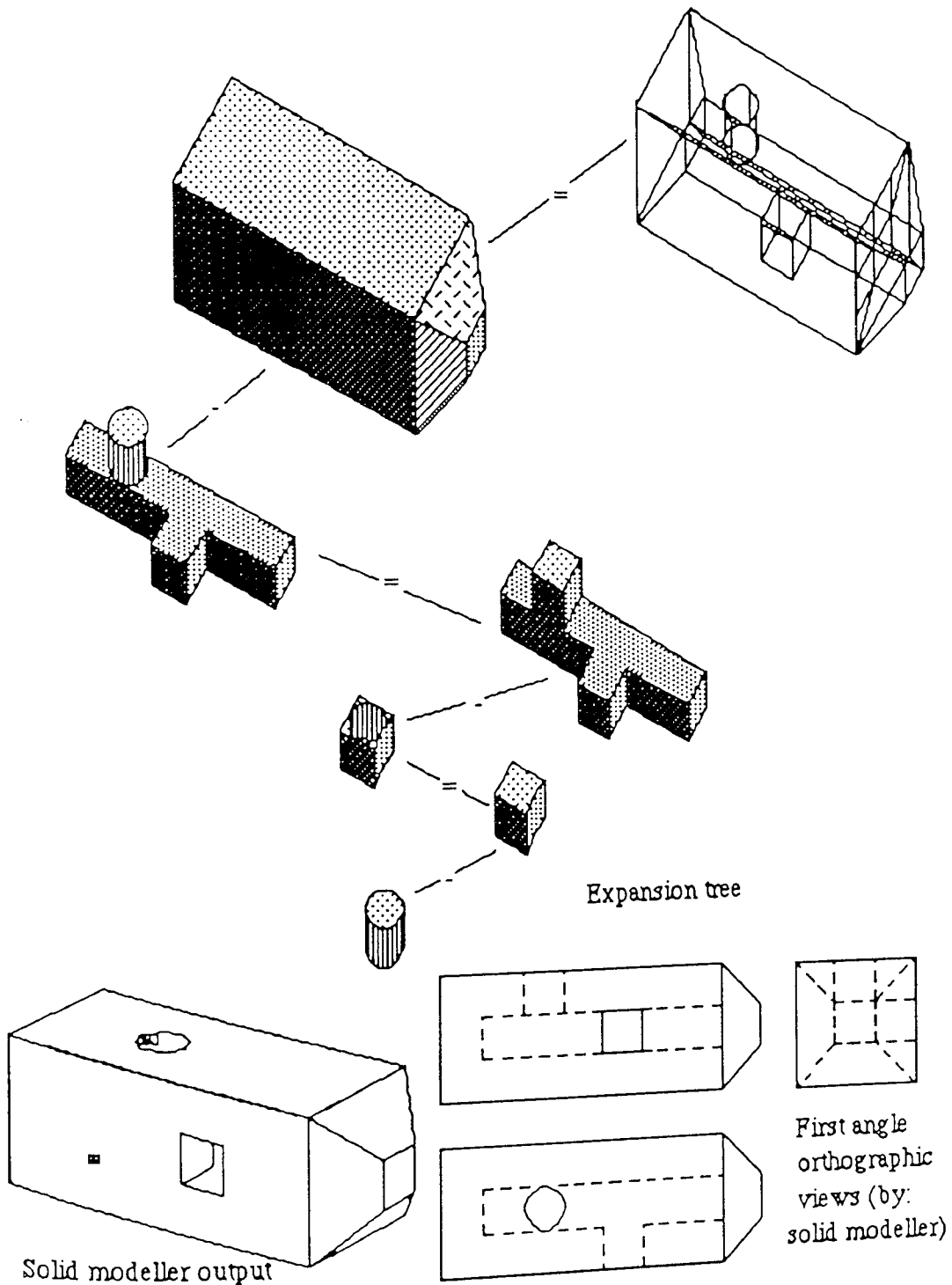
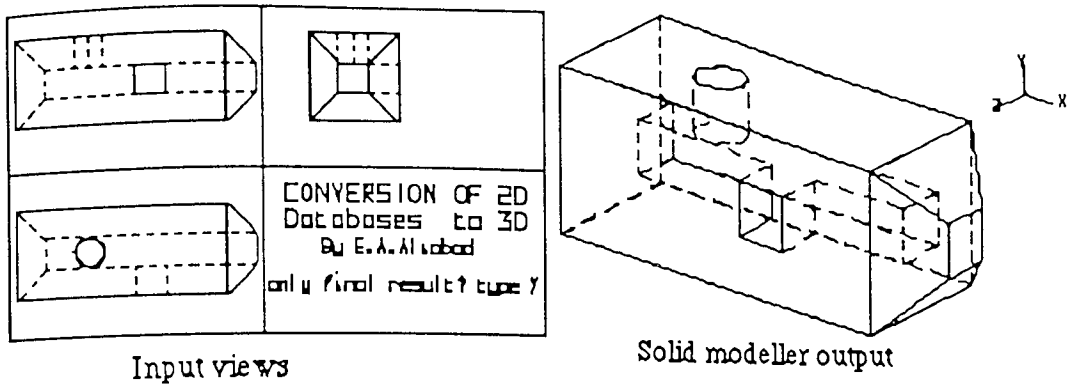


Figure A.3 Example 3

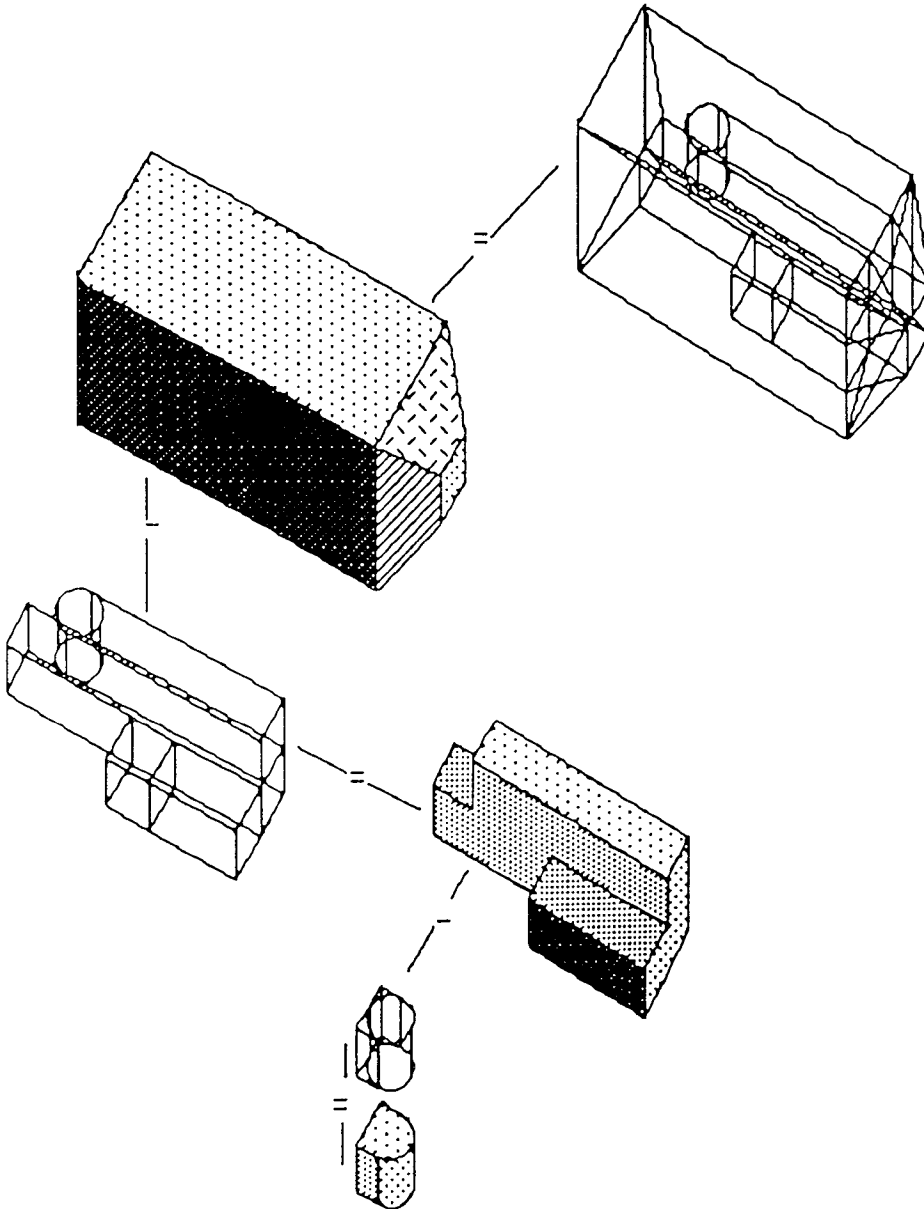
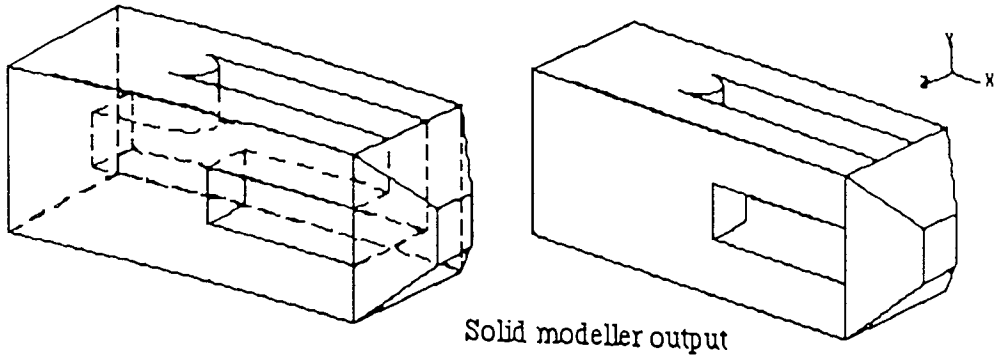
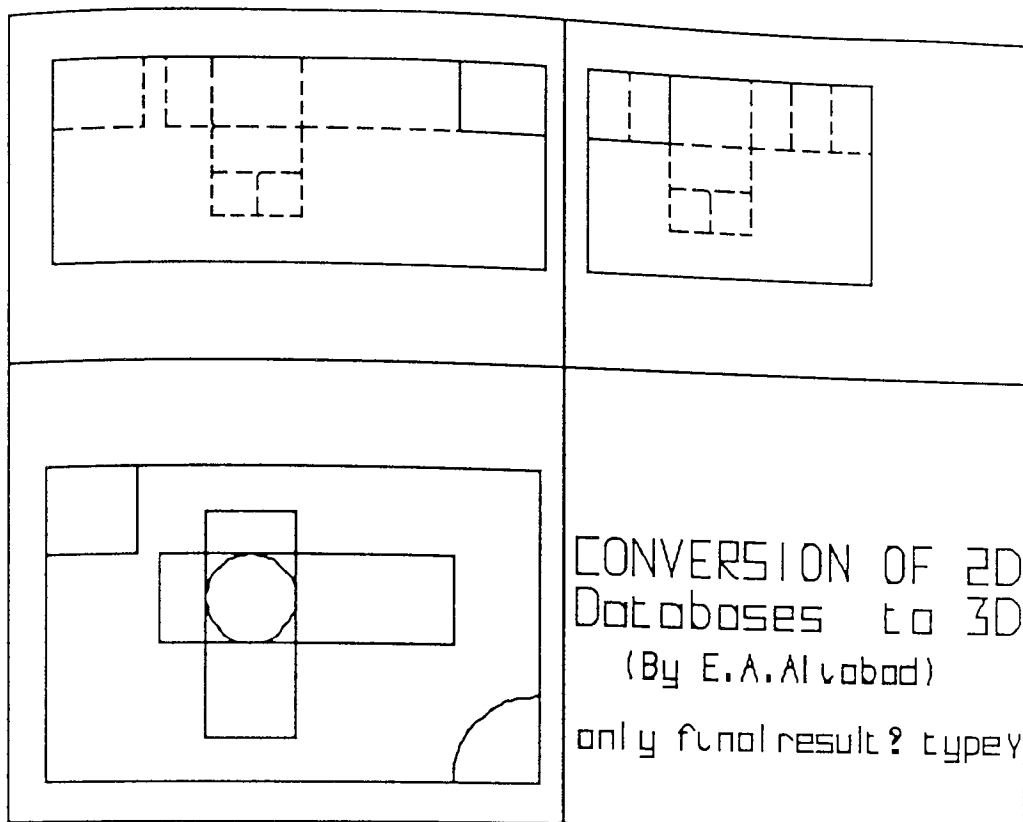
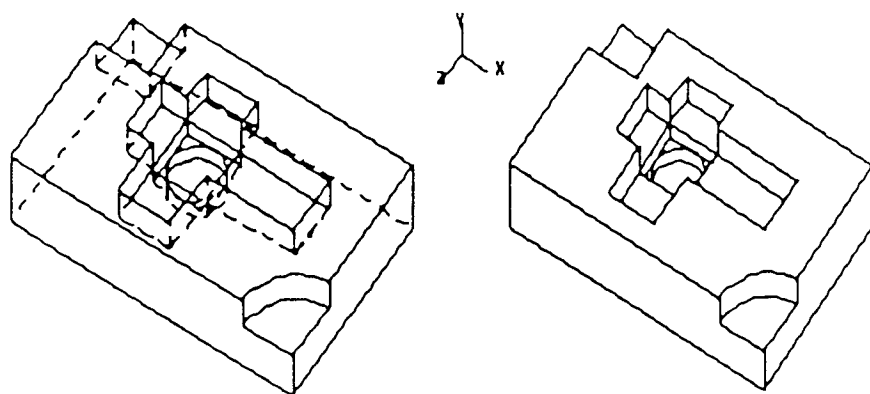


Figure A.4 Example 3 (continued)

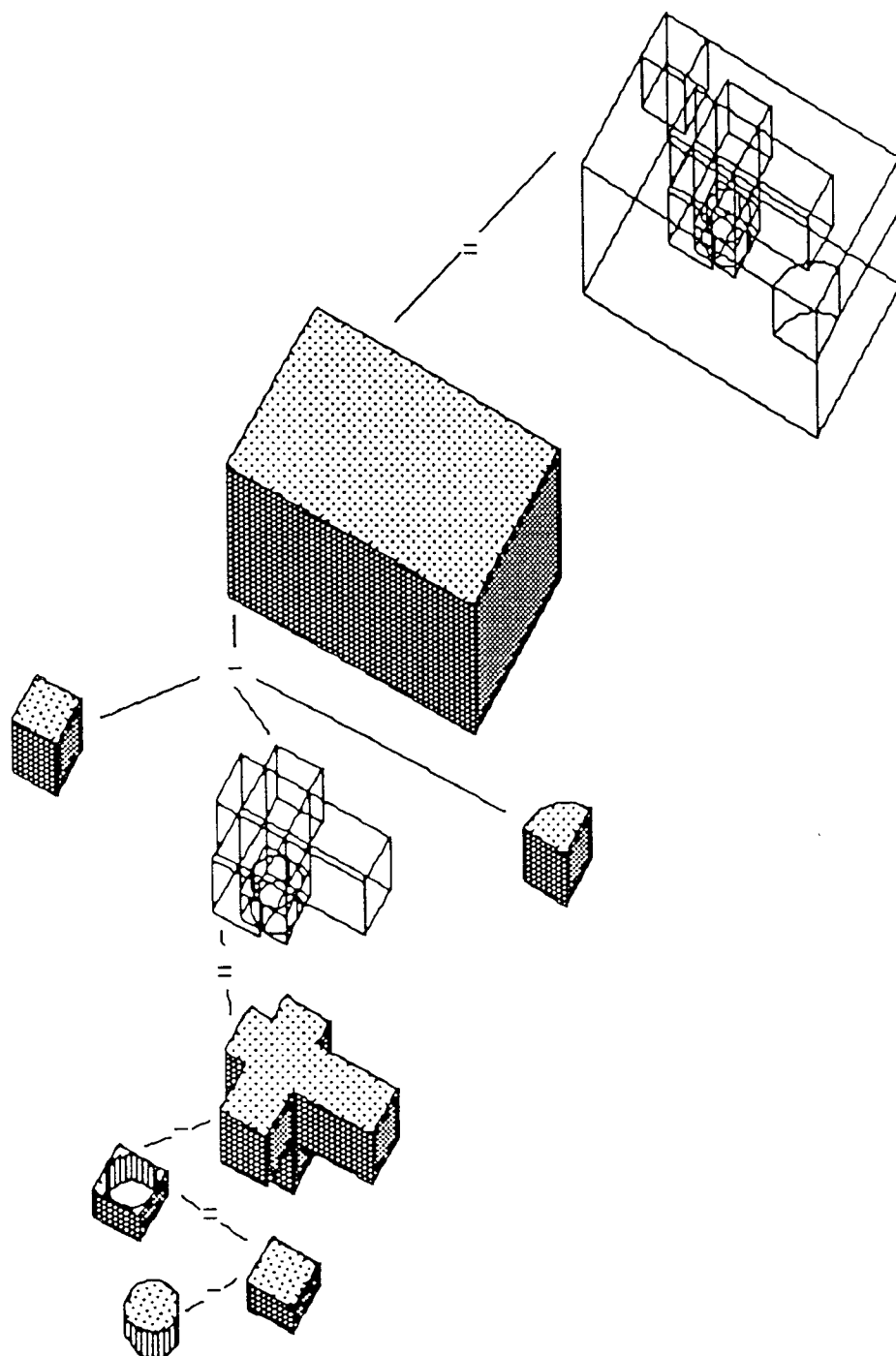


Input views



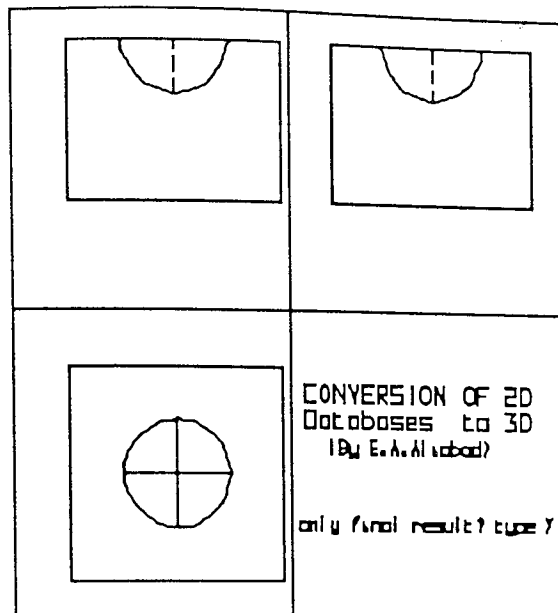
Solid modeller output

Figure A.5 Example 4

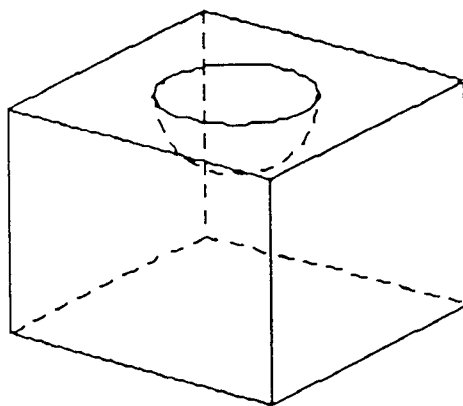


Expansion tree

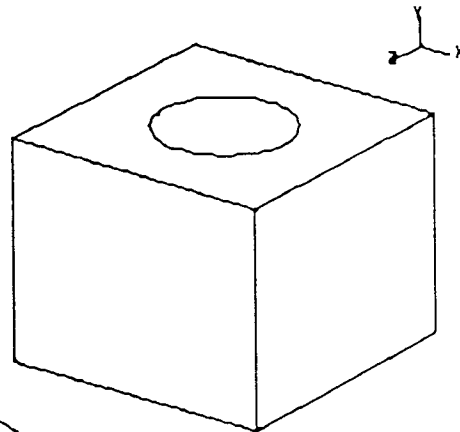
Figure A.6 Example 4 (continued)



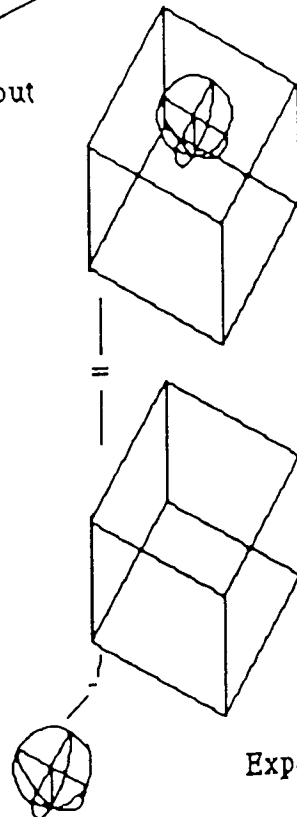
Input views



Solid modeller output

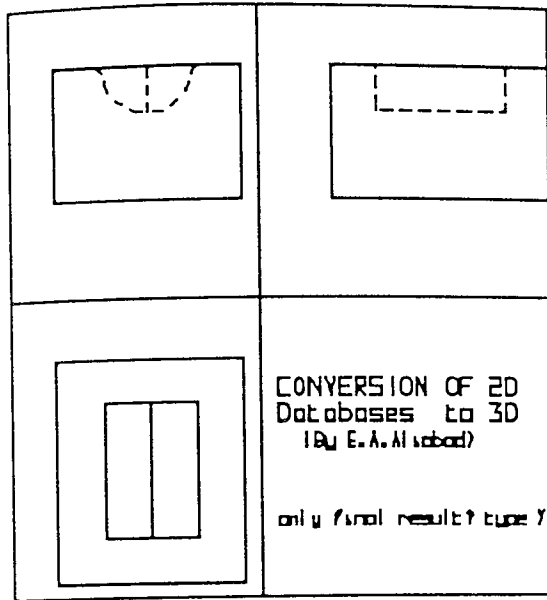


Solid modeller output

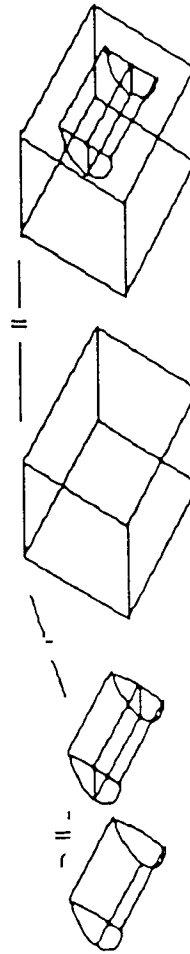


Expansion tree

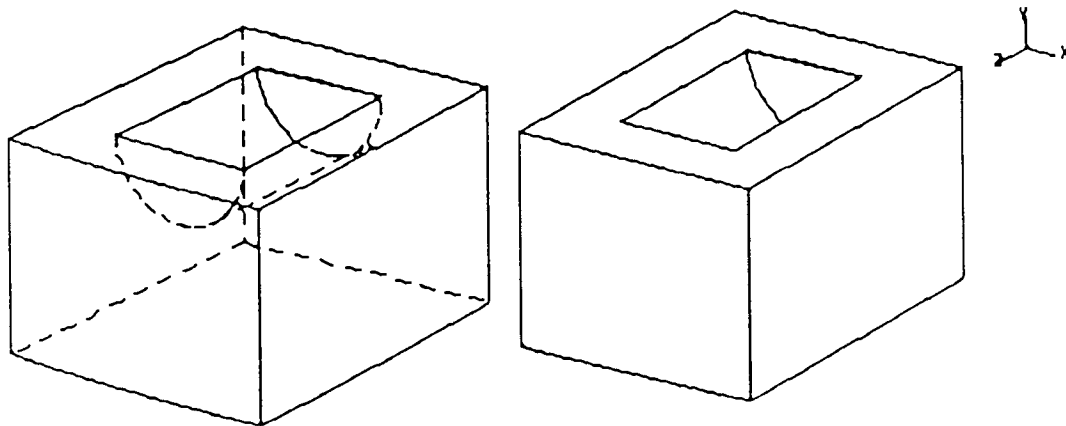
Figure A.7 Example 5



Input views

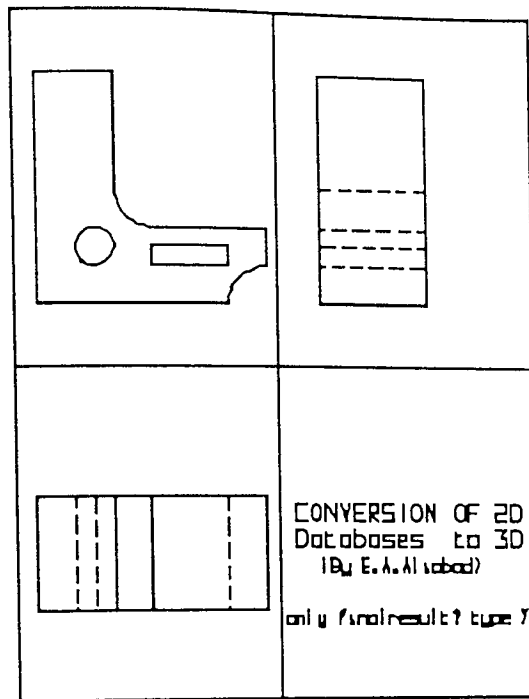


Expansion tree

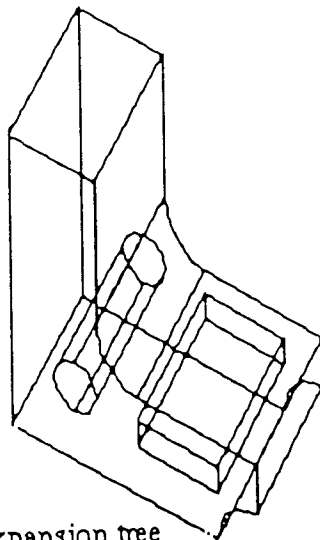


Solid modeller output

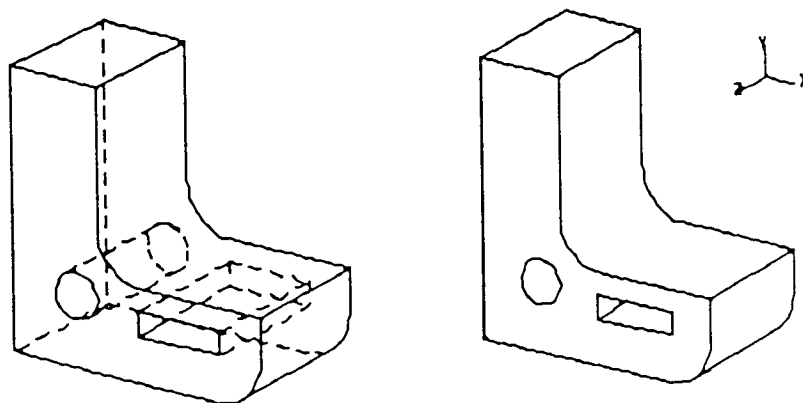
Figure A.8 Example 6



Input views

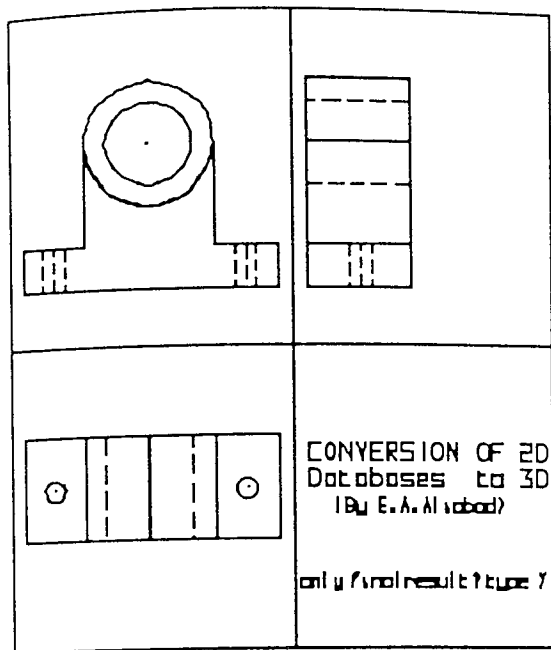


Expansion tree

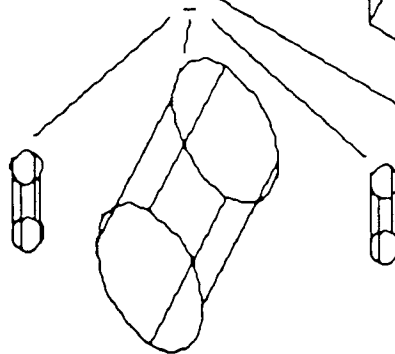
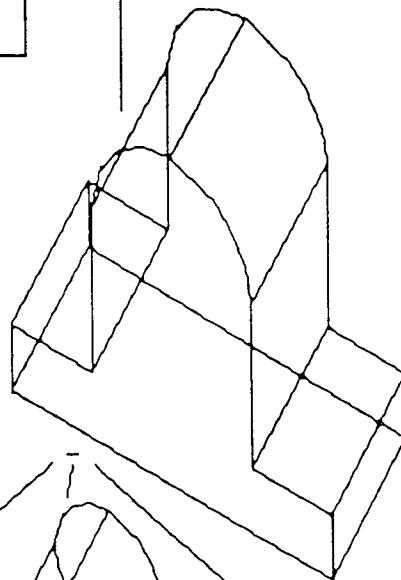
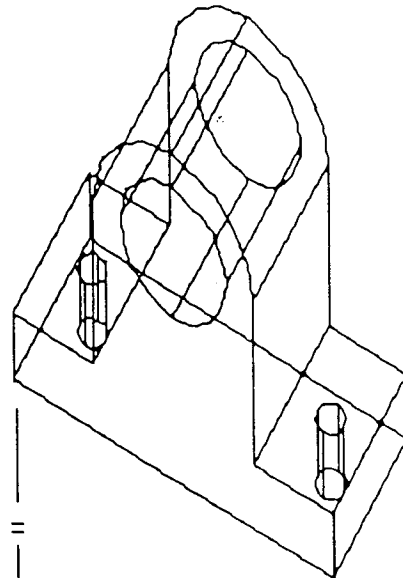


Solid modeller output

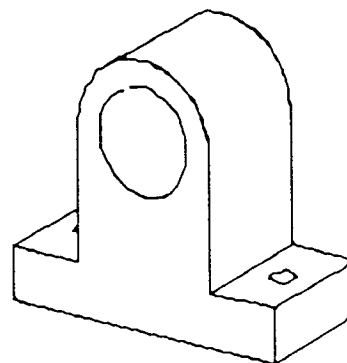
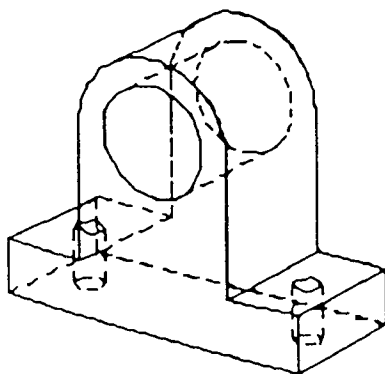
Figure A.9 Example 7



Input views

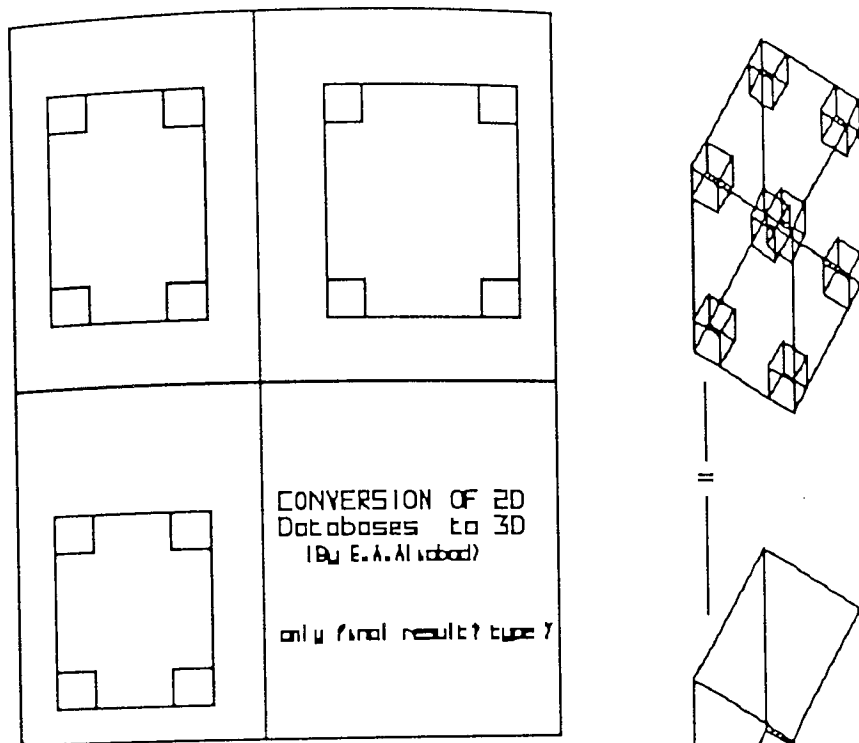


Expansion tree

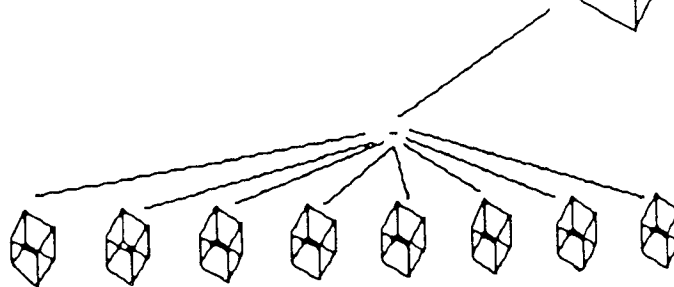


Solid modeller output

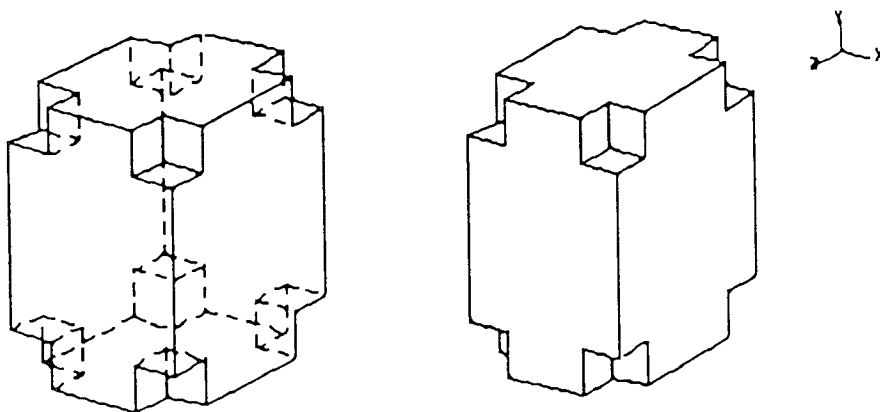
Figure A.10 Example 8



Input views

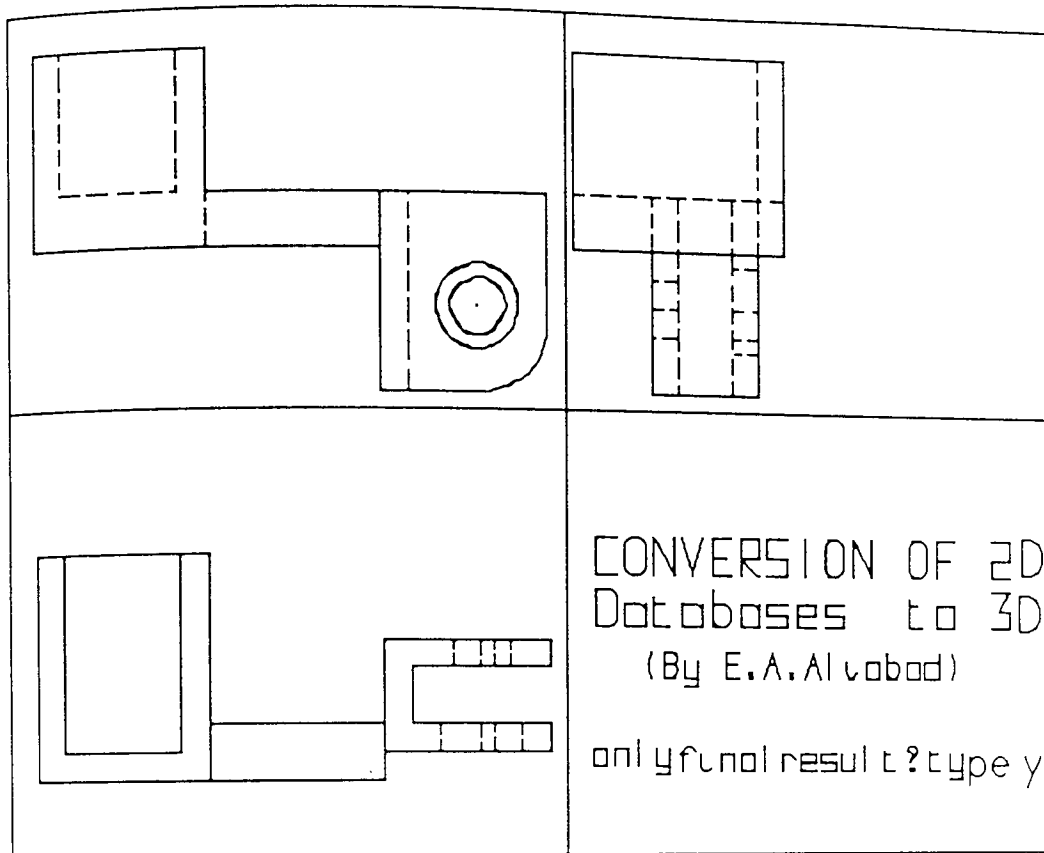


Expansion tree

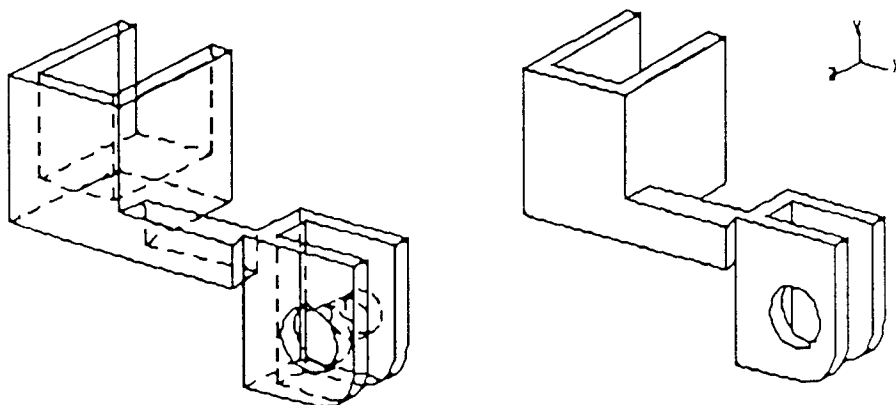


Solid modeller output

Figure A.11 Example 9

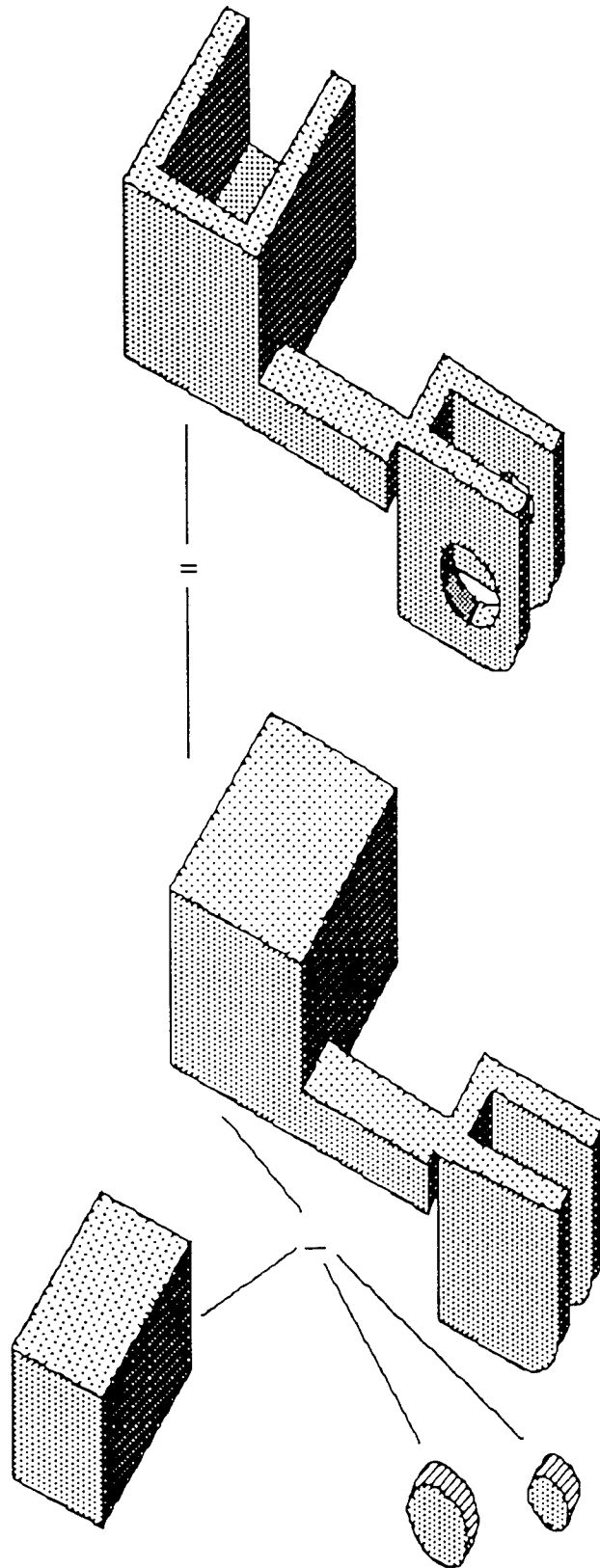


Input views



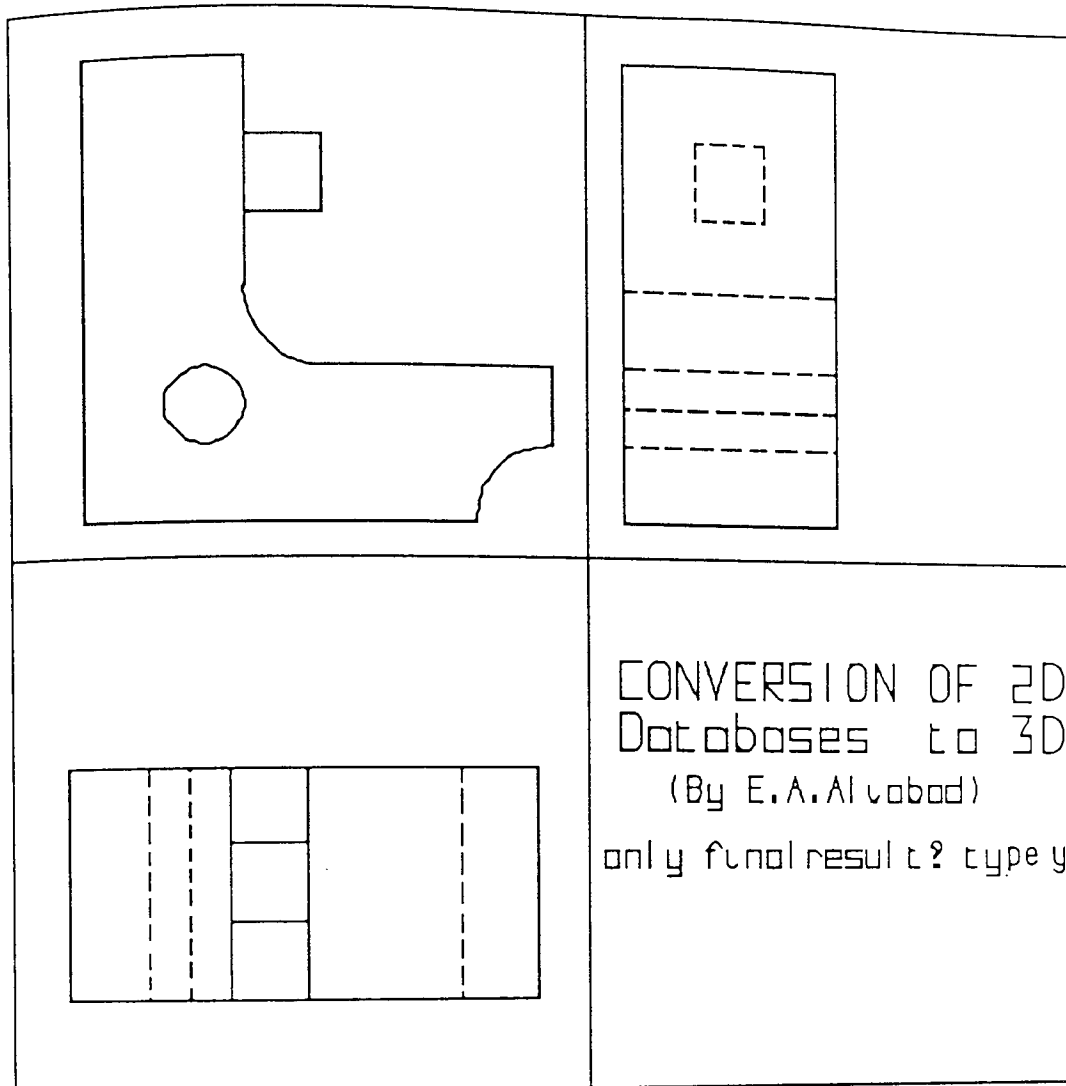
Solid modeller output

Figure A.12 Example 10

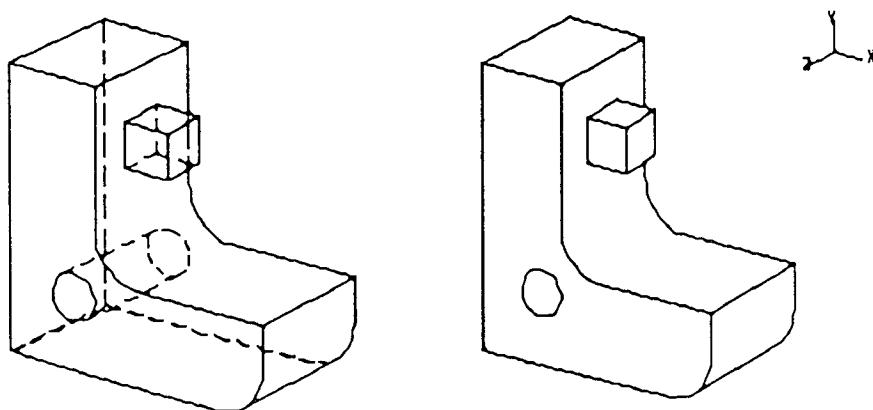


Expansion tree

Figure A.13 Example 10 (continued)

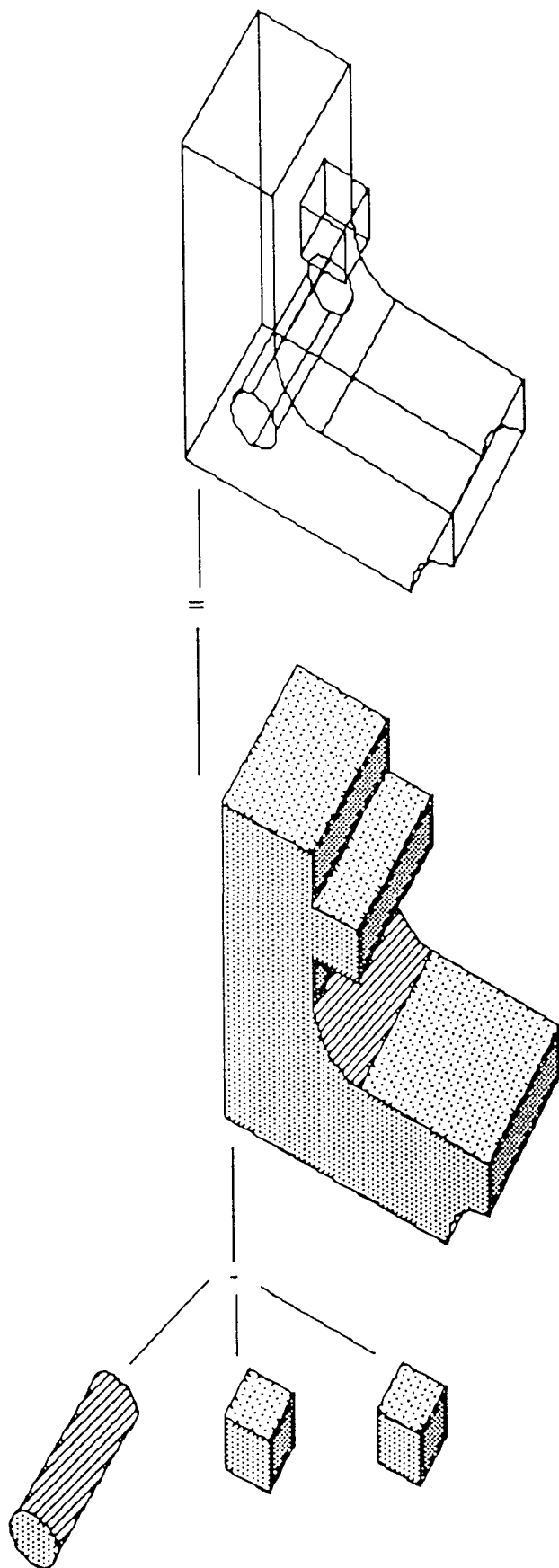


Input views



Solid modeller output

Figure A.14 Example 11



Expansion tree

Figure A.15 Example 11 (continued)

B. CAD/CAM 90 : state of the art

The CAD/CAM 90 exhibition showed the complete spectrum of CAD through to CAM, with every conceivable product, service and application. The latest developments in software techniques combined with advanced technologies in hardware were demonstrated by more than 250 companies in this field, such as: DEC, Prime, Autodesk, Daisy, Hewlett Packard, British Aerospace, Intergraph, Tektronix, Apollo, Olivetti, Sun, Pafec, NEC, Hitachi, Compaq, Roland, Deltacam, Unisys, etc.

Some of the application areas covered by the show were in civil, structural, mechanical, electronic, architecture, manufacturing and production engineering.

An overall look at the demonstrated products makes it obvious how fast this field is growing up both in software and hardware, and this verifies its important role in new industries. Final products were presented for marketing, and the advertiser could only demonstrate the specifications and capabilities of the products from the user point of view. Therefore, much technical points and details of the algorithms used in implementation and development could not be accessed. There were two main categories of presented products. One was hardware devices used for inputting, processing and outputting, i.e. workstations. It is quite evident that new facilities in hardware also affect the techniques of implementing software on them. The other category was CAD software by itself, designed for different applications. We briefly consider both categories separately.

Hardware used in CAD/CAM

Hardware equipment is divided into the following groups:

1- Input devices : This is an important section, especially when it is intended to input drawings into the computer. Some new digitiser tablets and scanners with higher accuracy have been introduced by some companies such as Summagraphics and Houston Instrument.

An interesting device presented by a French company was a 3D digitiser, which consisted of a rotatable platform with the object fixed on it and a laser beam in the middle projected on the object. Also two video cameras had been fixed on two corners. The digitising process begins by starting the rotation of the object, digitised data samples are collected by the computers and then by special software the boundaries of the object are built up from these data.

High resolution scanners are now becoming more popular and are widely used in entering images and drawings. In this regard video cameras are also being considered as another tool for capturing images. Utilisation of these two recent devices gives a reduction in costs, because of the use of relatively cheap charge coupled devices in their construction.

2- Output devices : This section includes devices which have the responsibility of reflecting the processed results (mainly drawings and pictures) for users as follows:

a - Printer: Different types from dot matrix, laser to colour ink jet from different companies.

b - Plotter: Faster colour and multipen plotters with higher resolution, including photo plotters and mask cutter devices from companies like ROLAND, CALCOMP, ZETA, SCHLUMBERGER, HEWLETT PACKARD and so on were presented.

In regard to making plotters independent from the main processor, a device called the

plotSERVER was introduced.

c - Monitor: Most effort has been expended in producing colour displays with higher resolution, since this is the most important section for visualisation, interactive editing, rendering and shading of an object. HITACHI, TAXAN and MITSUBISHI are among those who produce CRT.

3- Main Processor : The heart of a CAD workstation is its processor. It is desired to make them as fast as possible, using different techniques to achieve more processing power in order to be able to handle difficult tasks and time consuming applications. 32-bit processors such as the Intel 80386 and its coprocessor 80387 are now widely being used, and the 80486 is now entering the market. Besides, they are doing their best to make faster and cheaper memories to increase the size of main memory, and to maintain a large working area for large programs with greater size of arrays.

Currently there are many companies around the world making workstations for CAD applications such as DEC, HEWLETT PACKARD, COMPAQ, APOLLO, SUN, etc.

4- Secondary Mass Storage : The more powerful workstations have to have higher capacity hard disks with lower access time and higher transfer rate to maintain large programs, huge databases and libraries, especially for knowledge based systems. Also these must have high capacity floppy disks for transportation of Software. A media like streaming tapes for keeping back-ups is essential too.

CAD Software

Looking to the software products shows that most of the programs are designed for 2D draughting and applications like PCB design, architectural design, pipeline design, electrical schematic drawing and so on. Other areas are 3D solid modelling, rendering and visualisation. For many applications there is a built-in library of symbols which can

be selected from a menu by a tablet or a mouse from CRT. Some use parametric functions or Bezier curves to build curved shapes. There were many systems which were capable of inputting drawings through scanners, editing and saving them for other applications. Only a few systems had the capability of transferring raster images taken by scanners into vector form e.g. RAVEN from PAFEC. Most of the solid modellers used wireframe model as a base and surface modelling or boundary representation. Some used Sweep concepts to generate solids. Only a few systems like BOXER use CSG models to directly make solid objects from primitives. Here we introduce some of these software products:

***SUPERVISIONS 3000 from ATS (Advanced Technology Solution):** This is an advanced, powerful, fully integrated 2D and 3D design and modelling system. While other CAD systems build up entities from a collection of "draughted" lines, arc, etc., SUPERVISIONS models entities in an intelligent way. Design, modelling, shading and draughting are all done within the same package. Also it enables the user to attach additional non-graphical attributes to components and assemblies with information such as part number, description, properties, prices, etc. It can then produce on-line comprehensive bills of materials and schedules. You can take any profile and sweep it around an arc or along a path to produce solids, shells or tubes of your own design or choose from standard elements such as boxes or cylinders. The profile can contain holes and you can decide how many steps you wish to make in the extrusion, and at each step you may magnify or twist the section with respect to the previous one. This system allows you to extract any information from your model, both geometric, with full connectivity, or a list of attributes of the items concerned. Surfaces are created simply like extrusions but each section produced from the original profile can be manipulated independently of its neighbouring sections while the connectivity between them is still maintained. This means you can start off from a simple symmetrical shape and, by using interactive editing commands, precisely design the required shape. These surfaces may be combined with other 3D elements to construct complex objects.

*FESTRESS from ATS: It covers a wide range of stress analysis problems in continuum mechanics with powerful graphic capabilities.

*FESTRUCT from ATS handles structural analysis problems from simple 2D frames to complex 3D surfaces. It shares the same graphics and output capabilities with FESTRESS.

*ATSStructE from ATS is a series of advanced and versatile micro computer based modular software packages developed for analysis and design of structures above and/or below ground, e.g. buildings, bridges, transmission towers, off-shore platforms, basements, tunnels, etc.

*MICRO CADAM from CADAM is a complete pc-based software package for 2D/3D draughting and solid modelling. It utilises much of the technology found in the CADAM computer-aided design system, developed by the Lockheed Corporation.

*AUTOSOLID from AUTODESK Ltd: Autosolid is a hybrid modeller, combining the advantages of B-Rep and CSG methods. It is based on the proven technology of PADL-2, the internationally recognised Part and Assembly Description Language developed at the University of Rochester. Both CSG tree editing and edge/surface data extraction are supported. Mass property data can be obtained at any stage of design, and includes mass, volume, centre of gravity and moments. AutoSolid's two-way translator provides both DXF and IGES input and output, allowing easy exchange of data with AUTOCAD and other specialist software.

*AUTOSHADE from AUTODESK is a rendering program that converts 3D AUTOCAD wireframe drawings into realistic images. It enhances the capabilities of AUTOCAD with perspective, shading, and specular reflection, creating 3D renderings that are lifelike and attractive.

*AUTOCAD AEC from AUTODESK is a powerful design and draughting software package for professionals in the architecture, engineering and construction industries. The program contains a complete library of architectural shapes and symbols. Structural, plumbing, electrical, furniture and appliances, site planning, and titling symbols are all available for automatic insertion anywhere in drawings.

*SCHEMA-PCB from Omaton Inc. : This package, combined with another called SCHEMA-ROUTE, presents the engineer with a powerful, flexible and task-saving tool in the design of printed circuit boards. The combination of two programs delivers the high performance capabilities you expect for fast, efficient automatic routing of connections coupled with the benefits of real time interactive editing.

*CADPIPE, with AUTOCAD as the graphics driver, makes piping design easy and fast, from the production of the first PFD through design, construction, start up, operation, and revamps once the plant is in production.

*AUTOPIPE is a special purpose program for analysis and design of piping systems subjected to operational and environmental loads.

*VERSACAD is a true 3D database which allows full 3D modelling, with hidden lines and light sources shading. Mass properties calculations and automatic take-offs to 2D are standard.

*RAVEN from PAFEC: This is an advanced image processing software package for copying existing drawings into a CAD system. The image of the drawing is captured by an optical scanner, and stored on a high speed computer system. Initially, the image is held in raster form. This raster image can be cleaned up if necessary using simple editing facilities and other techniques such as thresholding. An important feature of RAVEN is that the image can be converted to vector form to be used by another CAD system.

*PAFEC-FEA is a completely self-contained, high powered finite element analysis system which allows the user complete control over the analysis process through a command module. This program can solve:

- 1-Virtually all classes of static problems, both linear and non-linear.
- 2-Linear and nonlinear dynamic problems for evaluation of natural frequencies, harmonic, seismic and transient response.
- 3-Temperature problems both steady state and transient.

*DOGS is PAFEC's well established 2D CAD system, and can drastically reduce the number of steps in the design process and make each step much easier and less tedious to perform. In addition to 2D design, PAFEC offers a complete range of 3D design

software. DOGS3D, SWANS and BOXER are designed to cater for wireframe, surface and solid modelling respectively. Each enables the designer to visualise and gain a better understanding of the design, reducing costly errors otherwise difficult to foresee.

*DOGS3D is used for 3D wireframe or surface modelling and has the following features:

- wireframe or surface or both.
- surfaces can be planar, cylindrical, conical, spherical or toroidal.
- surface area calculations.
- rapid hidden surface removal for on-screen display.
- shading in a variety of colours.
- picture file to save hidden line/surface views for later use.
- up to 10 different light sources for a display.
- flat pattern development available.

*SWANS is a sculptured surface modelling system ideal for generating models of complex design that require surface definition.

*BOXER is a PAFEC's solid modelling package. It allows the solid nature of an object to provide you with a complete system for visualisation, design and mass calculation.

*PROVUE is PAFEC's plant design package. It shows how structural components can be built interactively in 3D space. The system speeds up the design process and allows changes to be implemented very fast. It constructs a complete 3D model of an entire plant and then view the design from any angle.

C. A SELECTION OF COMMERCIALY-AVAILABLE SOLID MODELLERS

REF NO	SUPPLIER	SYSTEM NAME	MODELLER SOURCE	MARKETING APPROACH
1	Applicon	Solid Modelling II	Internal + SYNTHAVISION	Bundled w/Turnkey
2	ATP	CIMPLEX	Internal	Independent
3	Auto-Trol	S7000 Solid Modelling Sys.	Internal + PADL-2	Bundled w/Turnkey
4	CADAM INC.	INTERACTIVE SOLIDS DESIGN	Internal (SYNTHAVIS.)	Bundled w/Turnkey
5	Caetec S/W	PRO-SOLID	Internal	Independent
6	Cambridge Inter- active Systems	MEDUSA	Internal	Bundled w/Turnkey
7	Computervision	Solidesign	Internal	Bundled w/Turnkey
8	Control Data Corporation	ICEM Solid Modeller	Internal + SYNTHAVISION	Bundled w/Turnkey
9	Evans & Suther- land (Shape Data)	ROMULUS	Internal	Independent
10	Gerber	GST-SOLID	Internal + Leeds Univ.	Bundled w/Turnkey
11	Graftek	ROMULUS	Internal + ROMULUS	Bundled w/Turnkey
12	IBM (SDRC)	CAEDS Modeller	SDRC	IBM sales & support
13	IBM (Dassault)	CATIA	Dassault	IBM sales &

REF NO	SUPPLIER	SYSTEM NAME	MODELLER SOURCE	MARKETING APPROACH
14	"ICM, Inc"	GMS	Internal	Independent
15	Intergraph	Solid Modeler	Internal	Bundled
16	MCS	OMNISOLIDS	Internal	Bundled
17	Matra Datavision	EUCLID	Internal	Independent
18	McDonnell Douglas	UNISOLIDS	PADL-2	Bundled
19	PDA	PATRAN II CONCEPTUAL	Internal	Independent
20	Phoenix	INSIGHT	Internal	Independent
21	Prime	MEDUSA	CIS- now independent	Bundled
22	SDRC	GEOMOD	Internal	Independent

----- MODELLER TECHNICAL DATA -----						
REF NO	SINGLE OR DUAL	TYPE - (BR,CSG,..)	REGULAR SURFACE TYPES	SCULPT. SURFACE TYPES	PLANAR APPROX.	ASSEMBLY MODELS
1	dual	BR/CSG	Quad&Torus	B-spline	BR uses facets	Yes
2	single	BR	Quad&Torus	planned	NONE	yes
3	single	CSG	Quadric	planned	NONE	
4	Dual	CSG	Quad&Torus	Yes	NONE	Yes
5	single	CSG	Quad&Torus			
6	single	BR	Quad&Torus	Yes	Yes	
7	single	BR	Quad&Torus	Freeform sched:1986	NONE	Yes
8	single	CSG	Quad&Torus	Yes	NONE	Yes
9	single	BR	Quad&Torus	"Blends, full" Sculpt. in devt.	none	yes
10	dual	BR/CSG	Quad&Torus		yes	
11	single	BR	Quad&Torus	Blends	NONE	yes
12	single	BR	Quad&Torus	NURBS	Yes	yes
13	single	BR	Quad&Torus	yes	yes	

----- MODELLER TECHNICAL DATA -----

REF NO	SINGLE OR DUAL	TYPE - (BR,CSG,..)	REGULAR SUR- FACE TYPES	SCULPT. SUR- FACE TYPES	PLANAR APPROX.	ASSEMBLY MODELS
14	single	BR	Quad&Torus		yes	
15	single	BR	Quad&Torus	B spline	for B- spline surf.	
16	single	CSG & BR	Quadric	yes	yes	
17	single	CSG & BR	Quad&Torus	Bezier	Yes	Yes
18	single	CSG	Quadric		NONE	yes
19	single	BR	Quad&Torus	cubic - parametric	NONE	
20	single	Octree				
21	single	BR	Quad&Torus	Yes	Yes	
22	single	BR	Quad&Torus	NURBS	Yes	yes

----- MODELLING TOOLS -----				
REF NO	PRIMITIVES	BOOLEANS	SWEEPING	LOCAL OPERATORS
-----	-----	-----	-----	-----
1	Yes	Yes	Yes	No
2	yes	yes	yes	
3	yes	yes	Yes	No
4	yes	yes	yes	No
5	yes	yes		
6	No	Yes	Yes	Yes
7	yes	yes	yes	
8	yes	yes	yes	
9	yes	yes	yes	yes
10	yes	yes	yes	
11	yes	yes	yes	yes
12	yes	yes	yes	yes
13	yes	yes	yes	

----- MODELLING TOOLS -----				
REF NO	PRIMITIVES	BOOLEANS	SWEEPING	LOCAL OPERATORS
----	-----	-----	-----	-----
14	yes	yes	yes	
15	yes	yes	yes	
16	Analytic solids	yes	yes	
17	yes	yes	yes	
18	yes	yes	yes	
19	yes	yes	yes	yes
20	Yes	Yes		
21	no	yes	yes	yes
22	yes	yes	yes	yes

REF MASS		SHADED IMAGES	APPLICATIONS			
NO PROPERTIES			FEM GENERATION	NC GENERATION	MODEL DRAFTING	DATA EXTRACT
1	Yes	Yes - multi light source	yes	Yes	Yes	BF Xfer IGES 2.5
2	yes	yes	yes	yes	yes	CADAM I/F IGES
3	Yes	Yes	Yes	Yes	Yes	BF Xfer IGES 2.0
4	Yes	Yes - multi light source	Yes	Yes	Yes	CADAM I/F IGES
5	Yes	Yes	Yes	PRO-DRAFT		for FEM & drafting
6	Yes	Yes - multi light source	Yes	Yes	Yes	IGES
7	Yes	Yes	Yes	Yes	Yes	IGES
8	Yes	Yes		Yes		IGES
9	Yes	Yes	Patran I/F	GNC & APT I/F avail	Yes	IGES
10		yes	via I/F	via I/F	via I/F	via I/F
11	Yes	"Yes, multi-" light planned	yes	yes	yes	IGES
12	yes	Yes - multi light source	"yes, and" Automatic		no - 2D xfer to CADAM	IGES
13	yes	yes		yes	"yes, and" CADAM I/F	IGES

		APPLICATIONS				
REF	MASS NO PROPERTIES	SHADED IMAGES	FEM GENERATION	NC GENERATION	MODEL DRAFTING	DATA EXTRACT
14	Yes	Yes	ANSYS and PATRAN I/F			
15	yes	yes	Yes	Yes	Yes	SIF and IGES
16	yes	yes	yes	yes	yes	IGES
17	Yes	Yes - multi light source	I/F to FEM analysis	yes	yes	IGES
18	Yes	Yes	Yes	Yes	Yes	IGES
19		Yes	Yes			Gateway module
20	Yes	Yes				
21	Yes	Yes	yes	yes	Yes	DARS
22	Yes	Yes - multi light source	"yes, and" Automatic		yes - with GEODRAW	Universal file & IGES