1987

# Assessment of classical database models for representing solids

Julia L. Deal

Rochester Institute of Technology

School of Computer Science and Technology

# Assessment of Classical Database Models for Representing Solids

by

Julia L. Deal

A thesis, submitted to
the Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree
of
Master of Science in Computer Science

Approved by:

1/28/87
Professor Guy Johnson, committee chairman

1/28/87
Professor Jeffrey A. Lasky

1/28/87
Professor Chris Comte

I prefer to be contacted each time a request for reproduction is made. I can be reached at the following address.

1/28/87

Julia L. Deal Date

17 Authors Avenue
Henrietta, NY 14467

ABSTRACT:

Solid modeling is being explored as a method of representing three dimensional parts for mechanical design and manufacturing.

This work analyzes the data storage requirements of the Boundary Representation and Constructive Solid Geometry methods of representing solid models. The ability of the database models (Hierarchical, Network, and Relational) to support solid modeling needs is evaluated. The goal is to determine the database model(s) best suited to store and manage the graphical data for solid model representations.

Background information about mechanical engineering, graphics, and database models is presented.

Entity-Relationship diagrams are used to define data requirements.

ACKNOWLEDGEMENT:

This thesis could not have been completed without the assistence, encouragement, and understanding given by so many, especially the following: Richard Chesley, Jim Scudder, Glenn Martin, and Barbara Smith, Jeffrey Yu, Richard Bartl and Stephanie Remillard, my thesis commitee: Guy Johnson, Jeff Lasky, and Chris Comte; and most importantly, Stephen Deal.

I am also very greatful to the dishwasher and microwave for providing uninterrupted service during this endeavor.

DEDICATION:

To Devo, WanDou, and Hairy, who remained faithful friends through it all.

Table of Contents

Figures

# 1. Description of the Research

## 1.1 Introduction

The engineering field has historically used pencil and paper drawings to represent objects. It was one of the first fields to take advantage of the benefits of computer graphics. The productivity, and in some cases the creativity, of an engineer is vastly improved with access to "Computer Aided" tools.

Mechanical engineering (including both design and manufacturing) is concerned with physical systems such as buildings, automobiles, ships, aircraft, and both large and small-scale machines. These items are three dimensional, solid objects. Traditionally, they were represented on paper by human-drawn diagrams; i.e., two dimensional models. The 2D model has since been transferred to the computer, and subsequently has grown into 3D wireframe representation. However, for many applications the wireframe model is not sufficient.

The mechanical engineering process has changed very little over the past century [REQU77]. In recent years, however, changes have taken place in the form of computer automation of many parts of the mechanical engineering process. The major goal in automating mechanical engineering process is to enable the systems involved (Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), Computer Aided

Engineering (CAE) to communicate effectively. This requires an integrated database to represent and supply the complete and accurate information needed by all the design and manufacturing processes. This is known as Computer Integrated Manufacturing (CIM).

Solid modeling is an area of increasing promise for providing complete and accurate information about three dimensional solid objects. It is a method of representing three dimensional objects that can be stored and manipulated by computers. The representation can also be used to answer questions about the real object.

Mechanical engineering deals with three dimensional objects that are made up of assemblies, sub-assemblies, and parts. This thesis is concerned with the storage of individual parts that are contained in a product database, and subsequently combined and assembled into the final product.

There is a distinction between drawing or drafting (end in itself), and design and analysis or post-processing (means to an end). The use of computers in the mechanical design process is changing from a drawing aid to a critical tool used every phase of mechanical engineering. For this reason graphical data is only one of a growing number of types of data that must be stored during the mechanical

engineering design process. Graphical data is the focus of this thesis because it is the basis of mechanical engineering, and because it is created at the beginning of the cycle in the design phase. The non-graphical data is also important, and references to it and its repercussions will be made whenever appropriate. Throughout this thesis the word "graphics" refers to the mechanical engineering graphics associated with design.

A large amount of work has been done on the use of computer graphics in the process of electrical and circuit design, especially VLSI chip design. The graphics used for electrical applications is mainly two dimensional graphics. Some of this work is applicable to solid modeling graphics and will be referenced in this thesis.

The goal of this thesis is to determine the database requirements of solid modeling graphical representation, and analyze the ability of the three classical database models to support these requirements. The choice of data structure is critical to the success of any computerized application. I believe that this is true of solid modeling for mechanical engineering.

## 1.2 Contents of this Thesis

Chapter two contains a discussion of engineering design

methodology, solid modeling, databases, and Entity Relationship diagrams. Chapter three briefly outlines the history of computer aided design (with emphasis on solid modeling) and relevant database and computer science developments, as well as a summary of commercial solid modeling packages available today. Chapters four and five, the main body of the thesis, we examine the data required by two different representations used to define solid models: Boundary Representations and Constructive Solid Geometry, and then analyze how the data may be stored in a database, including examples and a comparison of the various database models. Chapter six contains a summary of the results and conclusions drawn. It is concluded that, for both Boundary Representation and Constructive Solid Geometry solid model graphical representations, the relational database model is the most suitable.

The computer science field is plagued by the use of jargon and acronyms, as is the engineering field. Where possible terms are explained in the body of this thesis. A glossary is included in appendix A for reference.

## 2. Background Information

Solid modeling is useful in both mechanical design and manufacturing. This thesis addresses the needs of design, and less fully the needs of manufacturing. Engineering design methodology, solid modeling, database models, are discussed to provide insight into the data needed for solid modeling. Entity-Relationship diagrams are explained prior to their use in chapter 4. References are given and should be consulted if the reader requires more information.

### 2.1 Engineering Design Methodology

#### 2.1.1 The Mechanical Engineering Process

It will be helpful to understand the mechanical design process and the adaptations required by automation. The following is an outline of the steps in the design process. It is similar to most approaches to problem solving:

1. Recognition of Need
2. Definition of the Problem
3. Synthesis of a Solution
4. Analysis and Optimization
5. Evaluation
6. Presentation

Steps 2 through 5 form a cycle, which is repeated until the results are satisfactory. The basic concept behind design is gradual refinement and enrichment of the model as various options are exercised.

Design data is generally both large and complex. It is often desirable to view the data at different levels of abstraction, i.e., view different levels in the hierarchy. This enables the engineer to see what he needs, without unnecessary clutter. To complicate matters further, there are often alternate designs under consideration. When working on large projects there may be many designers involved at one time. In order for the parts to fit together, the designers need to see and reference (e.g. obtain measurement data) each others' designs.

There are two phases in the engineering process: design and manufacturing. The people involved in these two phases have different points of view and requirements. The designer is concerned with creation and refinement of parts and assemblies. The manufacturer, or product engineer is concerned with one part at a time, the description of which has been determined and fixed by the designer[JAKO83]. However, there is much data used by both the designer and manufacturing engineer. At the present time the link passing data between the two automated halves of the process seems weak at best.

### 2.1.2 Automation of the Engineering Process

The following is a list (in alphabetical order) of applications that could make use of design and other related data, if it was stored in a database in a complete and accessible form.

- Analysis of test data generated by prototypes
- Automatic assembly
- Bill of Materials
- Design
- Dimensional Inspection and Verification
- Documentation
- Drawing creation and plotting
- F.E.M.
- Interference checking
- Kinematic Simulation
- Mass properties
- Manufacturing engineering
- Part programming
- Process Planning
- Quality Control (Q.C.)
- Real time monitoring of sensors during prototype testing
- Simulation
- Structural analysis
- Tolerance Stacking
- Tool path control (N.C.)

Currently there is little coordination, and much data redundancy and duplication of effort.

#### Design and Manufacturing Components

Figure 2.1 is based on [RYAN85]. It breaks the engineering process into six processes and indicates applications used during each. As stated in the

Introduction to chapter 2, engineering design is the focus of this thesis. The ideal system would encompass the application shown here in a flexible and powerful system.

Design and Manufacturing Process

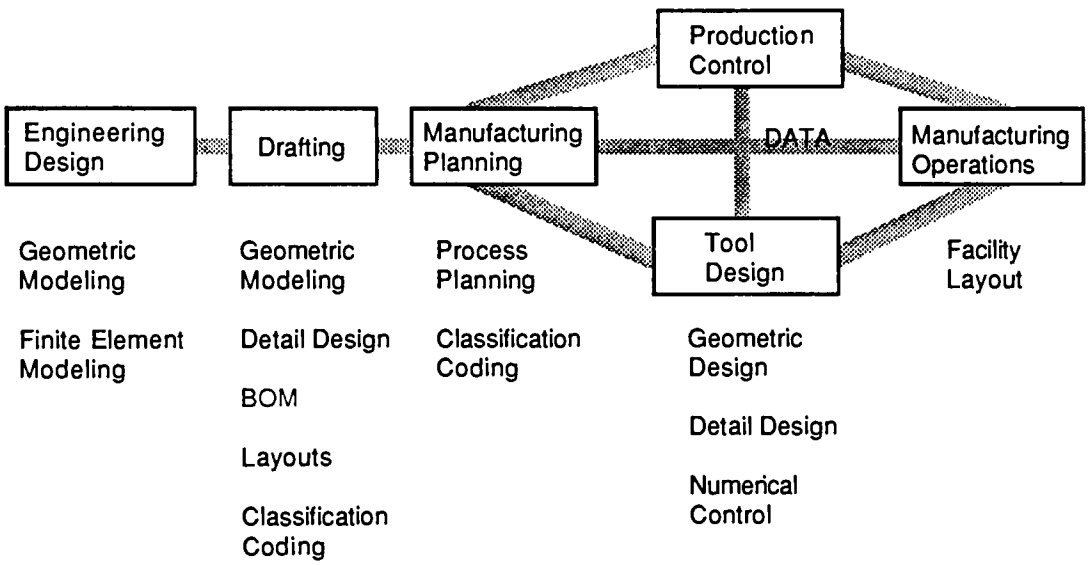



Figure 2.1

## 2.2 Solid Modeling Concepts

Solid modeling is generally an interactive graphics application which models solid objects. Interactive computer graphics enables the "user to dynamically control the pictures' content, format, size, and color on a display surface by means of interaction devices such as a keyboard, lever, or joystick" [FOLE83]. Pictures are a more natural form of communication for humans than characters and printed words, especially in engineering applications. This fact has provided much of the motivation for the development of user friendly interactive computer graphics including solid models.

This section covers the representation of solids in the computer, and the operations possible on such representations.

### 2.2.1 Methods of Representing Solid Objects

The computer is used to manage models of three dimensional objects. A useful model will provide answers to questions about the real object. In the case of a three dimensional or solid model, volume, mass, and moments of inertia may be determined. The following definition is given by [MORT85]:

> "A solid geometric model is the unambiguous and informationally complete mathematical representation of the shape of a physical object in a form that a computer can process."

The benefit of using a mathematically complete model is that it can answer many questions about the solid, such as volume, mass profile, and others, assuming that the interior is homogeneous. A robust solid model will enable the design and manufacturing communities to share the same database.

There are many possible representations, listed here in alphabetical order:

- Analytic Solid Modeling (ASM)
- Boundary Representation (BR)
  - Polygon meshes
  - Parametric bicubic patches
- Cell decomposition (including spatial occupancy enumeration)
- Contructive Solid Geometry (CSG)
- Instances and parameterized shapes (including template instancing)
- Octree Encoding
- Sweep representation
- Voxels

This list of representations was compiled from [BRAI75b], [BROW81], [JOHN84], and [REQU82]. One reason that there are so many, is that they support different functionality with varying ease. For example, some models are easy to manipulate and others are better suited for display. No one representation provides the functionality required by all applications. It is common for solid modeling systems to use more than one representation simultaneously to compensate for weaknesses in other representations. Table 2.4 provides a comparison of the functionality provided by the two most commonly used representations: Boundary

Representations (BR) and Contructive Solid Geometry (CSG). The use of multiple representations requires conversion from one representation to another each time a change is made to either version of the solid. For the system to be of value, the contents of both structures must be consistent, (i.e., represent the same solid) [REQU82].

There are four major concerns when choosing a solid modeling representation: mathematical representation, effective algorithms, suitable data structures, and the man-machine interface. Neither BR or CSG satisfy all the requirements but as table 2.4 shows, they compensate for each other's weaknesses in many areas.

To be practical in an engineering environment, a solid modeling system must be able to represent the majority of the conventional unsculptured parts. According to [REQU82], approximately 95 percent of these unsculptured parts can be represented by some combination of: solids definable via "natural" quadratic halfspaces, (ie. planar, cylindrical, conical, and spherical halfspaces) plus toroidal halfspaces; and attributes (such as discretionary rounds and fillets, and process defined geometry eg. threads, knurls, etc...). In addition, many industries such as the automobile, air/space, and ship building, require sculptured parts. However, systems which support sculptured solids in an efficient and reliable manner are

beyond the current state of the art [REQU82].

The BR and CSG representations are described below. Since they are the representations most commonly used in commercial solid modeling packages they will be the basis for the analysis done in chapters 4 and 5 of this thesis.

Boundary Representation (BR)

A solid object can be described by its boundaries, which are surfaces, faces, edges and vertices. Surfaces may be nonplanar (sculptured surfaces) or planar. There are two methods of defining a surface: polygon meshes and parametric bicubic patches.

Polygon meshes consist of a series of polygonally bounded planar surfaces. This method is relatively simple but is an approximation. Accuracy can be improved by the use of a greater number of more complex polygons. This requires additional storage and calculation resources. A polygon mesh can also be used to store flat, or 2D faces. The polygon mesh technique is analyzed in this thesis.

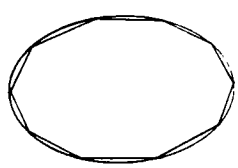


Figure 2.2

The second method of surface representation is the Parametric bicubic patch, which

> "defines the coordinates of points on a curved surface by using three equations, one for each x, y, and z. Each equation has two variables (parameters) and terms for all powers of the parameter up to their cube (hence the terms *bi* and *cubic*). The boundaries of the patch are parametric cubic curves" [FOLE83].

Compared to polygon meshes, fewer parametric bicubic patches are needed for accuracy, however the algorithms are more complex. Both methods represent a three dimensional object as an enclosed surface. Neither representation ensures that the model is a complete solid, thus both require additional overhead of validity checking either by software or by the user.

A boundary representation is normally broken into levels in a hierarchical manner. Solids contain surfaces made up of faces, which are bounded by edges, which have a vertex at each end.

## Constructive Solid Geometry (CSG)

A CSG model consists of a set of commands needed to reproduce the picture from shape primitives such as a cube, wedge, tetrahedron, cylinder, sector, fillet, cone, sphere. PADL-2, which was developed at the University of

Rochester by H. Voelcker and others [BROW82], is an example of a system based on CSG. The solid object can be represented using a binary tree of boolean operations or a list structure [JOHN84]. The leaves of the tree are the parametric shape primitives, sized and positioned in space. The nodes represent the results of the set operators: union, difference, and intersection on the leaves or nodes below [MORT85].

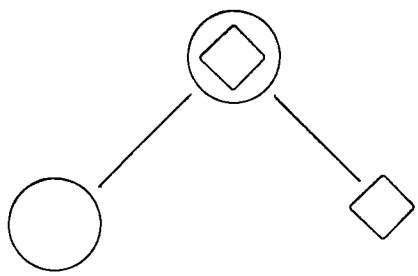


Figure 2.3

The PADL-2 project uses collections of graph data structures where the data elements are referred to as nodes [MARI86]. Indexing is provided to enable both graphical (pointing) and lexical (naming) access to the data. PADL-2 uses halfspaces to construct the block, cylinder, and other "primitives" [HART85d]. The various types of halfspaces include: planar, cylindrical, conical, and toroidal. A comparison of BR and CSG, their strengths and weaknesses, is given in Figure 2.4. The data necessary to store BR and CSG representations is analyzed in chapter 4 and 5.

## Comparison of BR and CSG Representations

| Function | Boundary Representation | Constructive Solid Geometry |
|---|---|---|
| Accuracy | Irregular surfaces approximated with B-splines or patches | More Accurate |
| Attribute Attaching | Unknown | Difficult if surfaces don't exist in the primitives |
| Ambiguity | Unambiguous if faces are Unambiguous | Unambiguous |
| Automatic Generation | Difficult & Expensive | No Algorithms known |
| Conversion | To CSG: Difficult<br>to others: Not known | To BR: Algorithms are known<br>to others: Fairly easy |
| Creation by humans | Difficult | Easy |
| Domain Determined By | Type of faces permitted | Primitives, Positioning Operators<br>Composition Operators |
| Efficiency | Unknown | Poor |
| Graphics Display | Efficient | Not Efficient and Slow |
| Mechanical Applications: | | |
| - Volume | Possible if homogenous | Divide and Conquer |
| - Mass | Possible if model is complete | Divide and Conquer |
| - NC | If dimensions & tolerance information are available | Not Efficient |
| - FEM | Possible from Polygon mesh | Difficult: primitives not disjoint |
| Nature of Dominant Errors | Geometric approximation and Numerical | Numerical |
| Storage | Bulky | Efficient & Compact |
| Transmission | Difficult | Easy |
| Tweaking | Possible | Not Possible |
| Uniqueness | Unique if faces are unique | Not unique => makes equality assessment difficult |
| Validity | Established thru properties of topological polyhedra in Euclidean space: Combinatorial & Metric conditions => can allow invalid parts if desired | Guaranteed validity if leaves are valid => all phases of design must (will) be valid parts |

Figure 2.4

### 2.2.2 Operations on Solids Objects

There are three phases of operation in interactive graphics. The first is input, during which data may be stored interactively or in batch mode. The second operation is manipulation of the picture on the display screen, and may include changing the shape of the object, or rotating it to expose a different view. The results of the manipulation may, or may not be stored in the data structure, depending on the particular operation and the results desired. The third, output, involves retrieval and display of data on a screen or plotter, or output to an application program. The algorithms used to achieve the above operations depend on the data structure used.

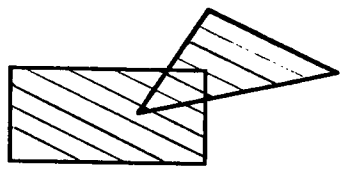
The operations discussed in this section may be used in both input and manipulation of solid models. The operators may be divided into two groups: Boolean and Unary.

Boolean Operators combine two objects creating one resulting object. [REQU85] lists three contexts were boolean operations are useful in solid modeling: (1) defining or creating solid models, especially operations on solid primitives; (2) modeling or simulating manufacturing processes, (e.g. milling and drilling); and (3) interference and collision checking. This thesis is primarily concerned with the creation and manipulation

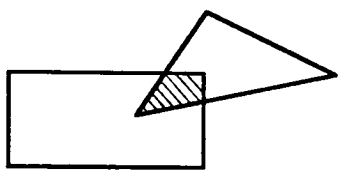
processes needed for mechanical design.

There are three boolean operators used in solid modeling. Each is described below and accompanied by a two dimensional example.

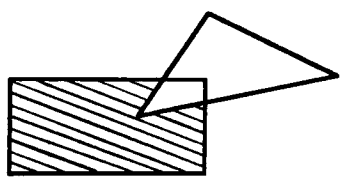
- Union: A U B. The resulting object contains both input entities A and B.



- Intersection: A & B. The result contains only what was common to both input entities.

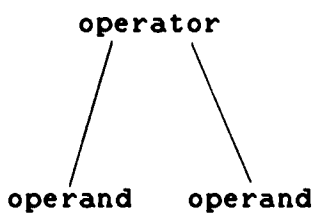


- Difference: A - B. The result contains the part of A that was not common to B.



When used in CSG systems a boolean operation does not affect the operands; a single entity is created as the output. In a BR system the two input entities are merged

to produce a single resulting entity. In either system an operand may be a primitive or the output of previous boolean operations. This can be depicted by a tree or hierarchical structure:




It is important that the representation and operators used in solid modeling systems maintain closure under boolean operations. The results must always be "homogeneously three dimensional"; i.e., there must be no dangling faces, edges or isolated points. For more detailed information about closure and regular set operations see [REQU80] or [BROW81].

Closure ensures that the output or result of a boolean operation will be valid input to subsequent boolean operations, as well as any other operation or calculation that may be performed on the solid model. In the remainder of this thesis all boolean operations discussed will be assumed to be regularized boolean operations.

Unary operations modify a single object, creating a

single resulting object. In many cases the output entity is the input entity, in some way modified.

Unary operations on solid models may operate on either the numeric data that controls the size and position of the solid, or on the topology of the solid or both. The operations covered in this section manipulate only the numeric data.

- Copy: the output is a copy of the input. This is the only unary operation which creates an output entity instead of modifying the input entity.
- Mirror: The result is the modified input entity, in a new location determined by the location and orientation of the mirror.
- Scale: The input entity is reduced or enlarged depending on the scale factor used and positioned depending on the focal point of the scaling operation.
- Rotation: The input entity is positioned according to the center of rotation and the angle of rotation. The convention is that positive rotation is counter clockwise.
- Translation: The output is in a new location, but at the same orientation as the input entity.

## 2.3 Database Concepts Related to Graphics

### 2.3.1 Database Requirements of Graphics Applications

The goals of an engineering database used to hold solid model representations include the goals of a general or business DBMS:

- Access Flexibility
- Administration and Control
- Concurrency
- Data Independence
- Ease of Use by End Users
- Ease of Use for Software Development
- Integrity
- Performance and Efficiency
- Recovery
- Redundancy of Data
- Security

Many in the engineering community feel that meeting these goals in an engineering and manufacturing environment is more difficult due to the additional complexity imposed by that environment[MELK84]. It is necessary to store many types of data in an engineering database: graphical/geometric data, relationships information, attribute data, semantic information etc. This increases the complexity when compared to storing business data consisting of numeric and text data.

Some special situations that occur in engineering databases include:

- Multiple representations of the same object, sometimes parallel models accessed by different design or manufacturing processes.
- Distributed computer components including "standalone" applications which must fit into the system.
- Performance of interactive, concurrent users is critical, and there are often large number of the users.
- Complexity, including recursive models is greater then generally encountered in business applications.
- Complex relationships requiring abstract data types, and access to these data types as units not collections of records.
- The languages historically used for engineering computing are FORTRAN and assembler, and more recently ALGOL, C, Pascal, PL/I, and SIMULA. Many earlier DBMSs supported only COBOL.
- It is necessary to minimize the storage space required due to the large amount of data stored.
- Multiple applications must have simultaneous access to the data, without compromising the integrity of the data.
- Large applications which often access data in a random manner [SIDL80] are different from large

applications which generally reqiure batch, sequential processing.

- The frequency of updates is not high, but each update affects a large amount of data and often takes a relatively long time to complete [SIDL80].
- Concurrent usage, but infrequent conflict characterizes access to engineering databases.

It is clear from the eleven points listed above that the requirements of a database management system used for engineering are quite different from those of a business environment.

### 2.3.2 Database Models

The three classical database models are outlined briefly below. For more detailed information consult [CARD85] or other database text books.

2.3.2.1 Hierarchical or Tree structures have been used since the late sixties. Hierarchical structures may be described in the following way: A parent node (or record) is decomposed into child nodes. This decomposition process may be repeated several times. The beginning or topmost node is called the root, and nodes with no children are called leaves. The connection between each child and its single parent is a branch, and the number of levels in the tree is the height of the tree [CARD 85].

There are no rings or closed circuits in this structure. Each leaf or node may have only one parent node. The following structure represents the faces and edges of a tetrahedron.

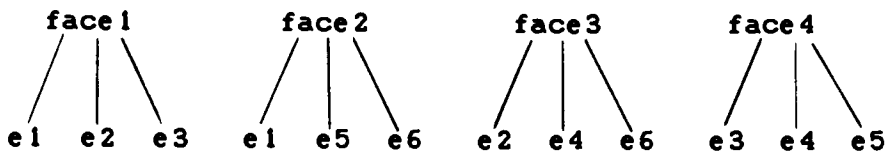



Figure 2.5

The above figure demonstrates that each edge appears twice, once for each face of which it is a boundary.

2.3.2.2 The Network structure is based on work by Bachman. It is more general (and complex) than the hierarchical model because a node or record may have more than one parent or owner. This allows representation of many to many relationships. These relationships must be maintained via a (possibly complex) set of pointers. Rings or circular paths through the database may exist in this model.

Figure 2.6 below shows the network of faces and edges required to represent a tetrahedron. Note that each edge is connected to two faces. Compare this figure with the hierarchical model in figure 2.5, where each edge is

repeated.

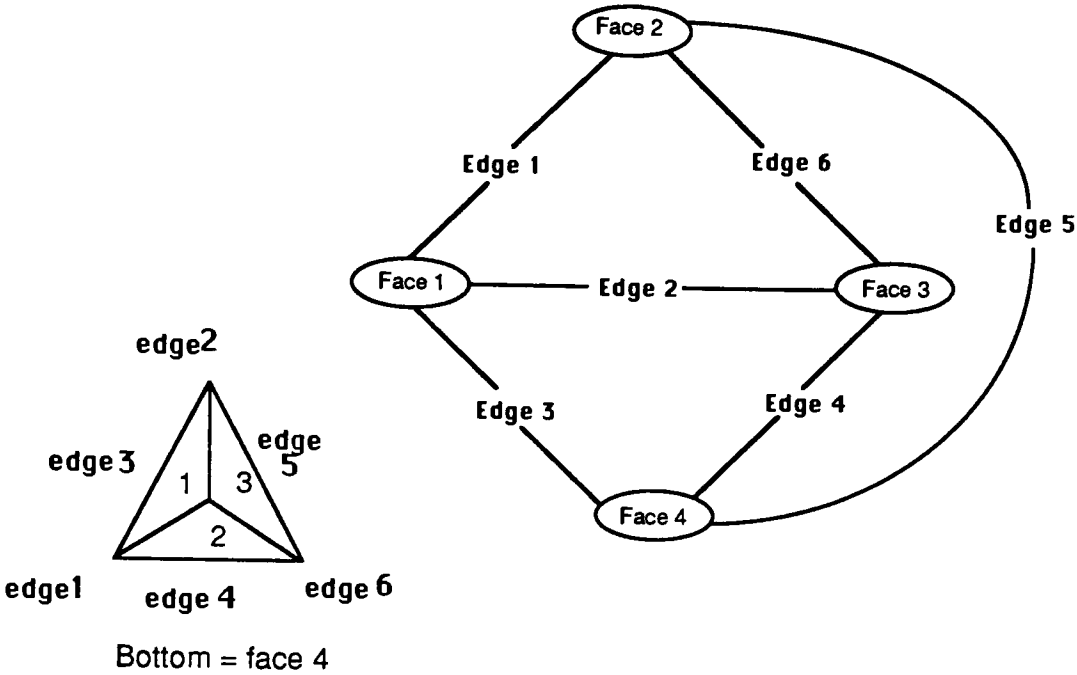



Figure 2.6

2.3.2.3 The Relational model was conceived by E. F. Codd in 1970, and has been further refined since that time. The logical database is viewed as a collection of two-dimensional tables called relations. Tables have columns (domains) and rows (tuples) of data. Each tuple must have a unique key; duplicate rows are not permitted. The three basic operations on a relational database are 1) Project: which returns a subset of the columns in a table, 2) Select: which returns a subset of the rows in a table, and 3) Join: which combines data from multiple tables.

Figure 2.7 below depicts the same tetrahedron used in the previous two examples. The Edge-id column is the key of the Edge-Face table. The list of edges bounding face 2 can be obtained via the following retrieve statement:

```
retrieve (edge-face.edge-id)
   where edge-face.face-id1 = 2
      or edge-face.face-id2 = 2
```

Edge-Face Table

| Edge-id | Face-id1 | Face-id2 |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 1 | 3 |
| 3 | 1 | 4 |
| 4 | 3 | 4 |
| 5 | 2 | 4 |
| 6 | 2 | 3 |

Figure 2.7

### 2.3.3 Modifications to the Standard Database Models

According to [LILL81], there is not total agreement about the usefulness of the traditional database models in engineering applications. Various authors have expressed the concern that the three standard database models are not ideal for an engineering database because:

- The schema for available DBMSs are static. Changes

occur during the engineering process, so the database must be flexible and adapt as necessary [KIMU81], [BARO81].

- The manual drawing approach (by a human with a pencil on paper) produces drawings used for communication. Understanding the meaning of the information depends on human knowledge and recognition ability. In addition, drawing content and meaning has not been formally documented. Further, the drawing is just the beginning of the process which continues through analysis and manufacture [KIMU81]. Using the computer to implement such an application is difficult.

- Access speeds, especially with multiple concurrent users, are not adequate [BARO81], [GRAB82].

- It has been widely accepted that the three "conventional" data models (hierarchic, network and relational "flat-file" structures) are not powerful enough to express all semantic structures and constraints in a reasonable way. Nevertheless, their usefulness as intermediate languages in a layered architecture is without doubt [EBER81].

The general concensus is that use of a DBMS is better than

re-inventing the wheel, especially if non-engineering data is to be stored in addition to the graphical representation. A popular research topic is how DBMSs might be modified to better suit the needs of mechanical engineering applications. Although the structure of graphical data is somewhat hierarchical, the most popular database model for graphics and engineering database research is currently the relational model, as evidenced by the summary of recent research in Table 2.4. Most of the implementations make some modification to the standard database model used. Other implementations of engineering applications with commercial DBMSs include [SIDL80], [KORE75], [MASA74] and [BRIS79].

[FOLE83] believes that "general-purpose database management systems are increasingly being used for the data structure function in graphics applications." The author believes that this is due to the increasing power and flexibility of the DBMSs currently on the market, especially relational DBMSs.

## Summary of Recent Database Research for Engineering Applications

| Reference | System Name | Data Model | Modification |
|---|---|---|---|
| [FISC79] | PHIDAS | Network | Temp DB keys (TDBKs) for concurrent updates |
| [BARO81] | CORAS | Associative storage | Time-critical access, Data & Reference Attributed, Graphics Relational & Network Primitives |
| [LORI81] | | Relational | Complex Objects, conversational Transactions, Archiving data on secondary DB, Non-formatted data elements, orderings, repetitive access to data |
| [GRAB82] | RASGOT | Network | Ring-Associativity structure to store Geometrical, Organizational & Technological data |
| [PATN82] | INTERGRAD | Relational | |
| [CHU 83] | VDD | Relational | Un-normalized Tables, redundant storage: tables & b-tree storage |
| [PATN 83] | ARDBID | Relational | Design Aiding Procedures<br>Actually three databases:<br>- 2D pictorial<br>- 3D pictorial<br>- non-pictorial |
| [STON83] | INGRES | Relational | Abstract data types |
| [HARD84] | | Relational | Complex Objects |
| [KITA84] | Formgraphics | Relational | Nested Tables |
| [SPOO84] | | Relational | Complex Objects |
| [WANG84] | | Network | Relational access to the data |
| [WASL84] | | Hierarchical | Linked Lists |
| [HART85d] | PADL-2 | Post-fix files | Abstract data types |

Figure 2.8

## 2.4 Entity-Relationship Diagrams

The following description of the DACOM Data Modeling technique is based on [DACO86].

An Entity-Relationship (E-R) diagram shows all the entities in a data model, and the relationships between them. This diagramming technique is often used in planning a database, and is a clear, concise, easy to understand communication tool. It is also a means of documenting what will be (or is currently) contained in the data structure. It generally represents the Conceptual Schema, as opposed to the Internal or External schemas.

The parts of an E-R diagram include:

1. Entities: are a collection of objects such as people, places, or things, about which data is collected. An entity is portrayed by a box with its name above the box.

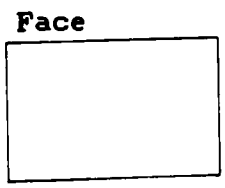



Figure 2.9

2. Attributes: are characteristics or properties of an entity. Attributes appear inside the entities' box, with the keys listed first and a horizontal line separating them from the remainder of the attributes. Generally, the basic E-R diagram used in the DACOM modeling technique does not contain attributes, however, in this thesis the "fully-attributed" version will be used.

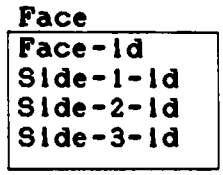



Figure 2.10

3. Relationships: are the connection of an entity to another. This is shown on the E-R diagram by a line connecting the two entity boxes involved, and a dot and a letter symbolizing the cardinality of the relationship. This may be zero, one, or many, as indicated by the following letters:

| | |
|---|---|
| P | positive (one or more) |
| M | zero, one, or many |
| Z | zero or one |
| n | where n is a required value |
| a-b | a range of values |

The dot indicates which side of the relationship is zero, one, or many. The side without the dot is always a single entity in the relationship. Each relationship line on the diagram is labeled with the name of the relationship. Many to many relationships are avoided and are divided into several simpler relationships.

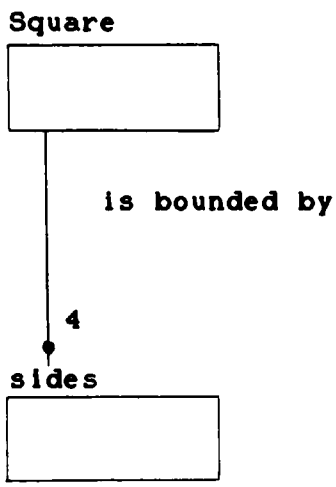



Figure 2.11

Category relationships relate a generic enntity to one or more specific categories. Each instance of the generic entity may belong to only one specific category. The generic entity contains an attribute called a discriminator that indicates to which category each generic entity belongs. A single horizontal line indicates that the categories are not fully enumerated, a double line indicates that they are fully enumerated.

Figure 2.12 shows a partially enumerated category.

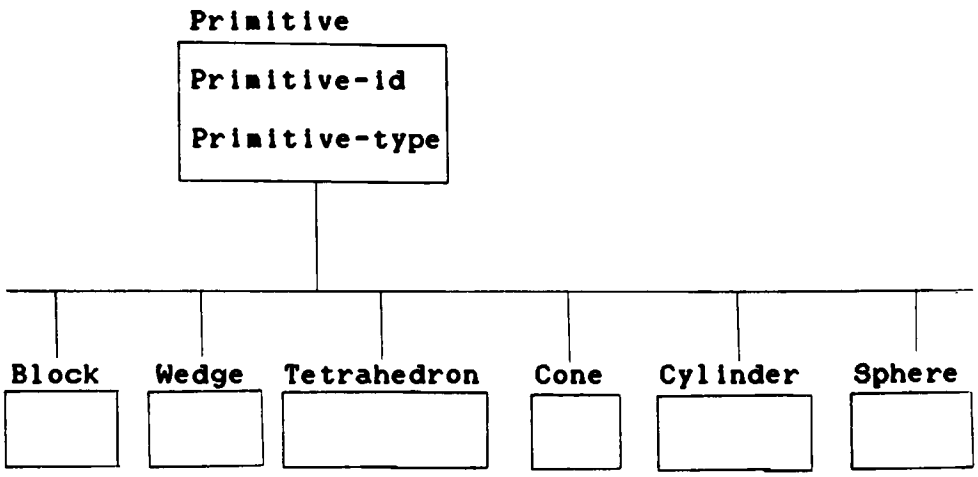



Figure 2.12

The data modeling technique may also include the creation of:

- A glossary: which defines entities and attributes.
- Business statements: which are detailed written descriptions of the relationships between entities.
- Entity-Relationship Matrix: a table showing the relationship of every entity to every other entity.

These basic data modeling techniques will be used in chapter 4 of this thesis. The advantage of using this technique to produce a fully-attributed model is that the result is a relational data model, normalized to the third normal form.

## 3. Past and Present Commercial Systems and Relevant Computer Science Developments

The use of computers for graphics applications, especially solid modeling, is not as well developed as that of business applications. This historical outline traces the development of the mechanical engineering application of graphics, and its development into solids modeling.

### 3.1 1950 - 1965: First Generation

Automatically Programmed Tool (APT), is a general geometric language used to control manufacturing equipment developed and used during this time period. CAD batch language was used to produce loft lines for automobiles [BIRD69].

Military use of graphics began in the mid fifties for applications such as SAGE air defense system [MACH69].

Ivan Sutherland's seminal Ph.D. work on the Sketchpad drawing system [FOLE83 p 18] done at Massachusetts Institute of Technology allowed the user to sketch on a display scope (CRT screen) with a light pen. He introduced ring data structures for storing symbol hierarchies that are built up via easy replication of standard components. It was used to simulate a bridge design where calculations were made on the basis of the drawing, and then corrected/adjusted as necessary

[MACH69]. It was also used to draw and simulate the movement of a linkage. Sketchpad III allowed the user to store solid objects in a 3D representation (as opposed to an adaptation of a 2D representation) also using the ring data structure [SUTH63]. The system had many sophisticated features and numerous applications, but its creators were most impressed with its ability to do tedious, repetitious tasks.

A Class Oriented Ring Association Language (CORAL) was developed in 1963 also at Massachusetts Institute of Technology by L. Roberts. The goal was to provide a user interface for graphics applications. It was based on the ring structure used by Sketchpad. It was basically a list structure language, graphical and problem oriented in nature. The hardware suggested was typical of the time period: a display scope, light pen, keyboard (function keys), and a typewriter (regular keyboard) [ROBE63]. Roberts was also involved in solid modeling and scene analysis problems during this period [REQU82].

In a discussion of the state of the art of computer graphics it was estimated that 100 graphics display devices were in use in the United States during this period [MACH69] [DUCE83]. An interactive graphics installation, including a dedicated host computer, cost in the neighborhood of $400,000 [DUCE83]. Color display

devices appeared in 1962 [DUCE83].

The concept of time sharing was implemented in the early sixties, enabling the transition from batch processing to interactive applications. Other advances in computer science which benefited the development of interactive graphics applications include: on-line real time environments, multi-programming, multi-processing, job control, and remote job entry [FOLE83].

## 3.2 1965 - 1975: Second Generation

The first commercially available interactive graphics CAD systems appeared, (eg. "ADAM" developed by Hanratty) [BYLE85]. Computer, automobile and aerospace industries were the first to show interest in (and have the economies of scale and resources required for) using interactive graphics for design and manufacturing. Lockheed-Georgia, McDonnell-Douglas, and Boeing used numerical control part programming of continuous path machines, where they had previously used APT [BYLE85].

General Motors' digigraphic design system, written for General Motors in PL/1 by IBM to run on the IBM 2250 console became available. Design Augmented by Computer (DAC) was developed in 1964 and used by General Motors. It was 8 to 10 times faster than batch processing of the same tasks. The concepts used in DAC were similar to those used in Sketchpad. DAC was also used by Ford Motor Company [MACH69]. Itek Laboratories developed an online graphics system, called Digraphics, for lens design, based on an X-Y plotter, during the same period [PRIN71].

Research in the sixties was initiated in the ship building design, building design and blueprint generation fields [PRIN71].

Several factors prevented the wide spread use of

interactive graphics when first developed. There were:

- Very expensive and specialized hardware, including the display equipment restricted use of graphics. Input was via light pens, track ball, joystick, touch wires, or tablets [EVAN69].
- Considerable computing resources were necessary to support extensive data structures, interactive graphics manipulation and the numerous post processing application programs. This further increased costs.
- Existing programmers were unfamiliar with interactive, timesharing programming and graphics.
- Standards were not in place, so each piece of software was custom developed and non-portable. This added to the cost [FOLE83] [PRIN71].

The Marconi Display System, developed by S. Bird in the late sixties, provided 23-bit accuracy in the data stored in the data structure even though the available displays are only capable of 10-bit resolution. Windowing capabilities enable expansion up u@ 8192 times. "It is equivalent to having a drawing board 2 miles wide" [BIRD69]. Additional non-graphical daup could be stored in the database, and, although not visible on the display, it was available to application programs. Bird's data structure was based on the Ring structure.

There were 500-1000 graphics display units in place by 1969

[MACH69]. The popular "graphics console" was the IBM 2250, often running on an IBM 360 series computer [PRIN71]. Despite the problems, increasing usage of graphics was indicated by the list of vendors in the back of the book documenting the Brunel Symposium in 1968. They included: CalComp, Datagraphic & Digital Systems, Cossar, Digital, Ferranti, Graphic Displays Ltd., IBM, International Commputers Ltd.(ICL), Marconi Company Limited, 3M Microfilm Products, Tektronics U.K. Ltd., and Sperry and Rand Univac.

According to [BRAI75a] and [REQU82], solid modeling research and development initiated during this time period included:

- Compac system at the T. U. of Berlin in 1969
- Proren system at U. of Ruhr shortly after 1969
- Build-1 system by I. C. Braid at Cambridge in 1973
- TIPS-1 at Hokkaido U. in 1973
- Geomap at Tokyo U. about the same time as TIPS-1
- PADL-1 at U. of Rochester in 1972-73
- Shapes by Laning in 1973
- EUKLID by Engeli in 1974
- GEOMOD by Baumgart in 1974

### 3.3 1975 - 1980: Third Generation

This time period was characterized by increased integration of interactive CAD and CAM graphics, including: design, analysis, attribute listing, drawing annotation, and manufacturing process setup. Wire frame and surface modeling were introduced. Larger and more complex parts were handled [BYLE85].

Solid modeling gained tremendous popularity after [BRAI75b] published an article titled "The synthesis of Solids Bounded by Many Faces" [HANR84]. The system he described allowed creation of complex solid objects via addition and subtraction (addition of negative objects) of six shape primitives: cube, wedge, tetrahedron, cylinder, sector and fillet. Computational geometry was stored for the complex object, and could be used to calculate the volume, centroid, moments of inertia, principal axes of objects, and other properties of solid models.

Solid modeling projects conducted during this time period included:

- CAM-I GMP started in 1976 [CAMI86]
- Work on the PADL-2 (CSG) project at the University of Rochester and Part and Assembly Description Lanuage
- Synthavision worked on by MAGI, GM and Boeing

- Shapes by Draper Labs, supported by Fiat
- GM Solid at General Motors in 1977
- Romulus at Shape Data in 1978
- Romulus, marketed by Evans & Sutherland in 1980
- Glide-1 at Carnegie-Mellon
- Geometric Modeling Project at Leeds in the U.K. in 1979

Processing power was gradually moved from the CPU to the display device by the use of minicomputers, which could handle "servicing the display", user interaction devices (such as a tablet, joystick, or mouse), and application programs locally. Increasing functionality migrated from the graphics software on the main computer to the display processor and associated software.

Television technology contributed raster graphics. This technology requires a large refresh buffer to hold the bitmap, and a processor, as did the hardware used in the early sixties (but by the late seventies the cost of this storage was more reasonable). A drawback of the raster displays is some loss of resolution. Extra memory is required to store the necessary number of pixels to improve the resolution [FOLE83]. However, as the cost of solid-state memory decreases the resolution may increase.

As device independent software became more common, standards were developed:

- Graphics Standards Planning Committee (GSPC) was formed in 1974 in the United States and eventually produced Core.
- Seillac I graphics workshop was held in France in May 1976. The purpose and content of a graphics standard were debated [DUCE83].
- The Core Graphics System was published by the ACM SIGGRAPH sponsored group GSPC in 1977. The 1978 issue of ACM Computing Survey was devoted to its description. Another version published in 1979 included raster graphics, in addition to vector graphics. It is a full 3D graphics system.
- The formalization of the GKS 2D standard was sponsored by ISO [FOLE83].

Despite the above standards, a neutral format was needed to translate between dissimilar systems. IGES - The Initial Graphics Exchange Standard - project was started in 1979, and administered by the National Bureau of Standards. IGES is a neutral data format specification. Approximately thirty vendors provide software to translate their data files to and from IGES format [LIEW85].

### 3.4 1980 - 1986: Fourth Generation

The current generation is experiencing increased capabilities in solids modeling in addition to those of the previous generation. Large files are supported with greater mathematical and geometric precision. Despite these advances, todays software is still slow, and difficult to maintain [BYLE85]. The first major commercial solid modeling systems, appearing in 1981, include those by Applicon and Computervision [WAGN84].

Hanratty proposes putting the software on a chip to speed up response time. Several vendors market hardware aids for the problems encountered in implementing solid modeling. [WAGN84] lists the following vendors:

- Weitek: solid modeling engine
- Lexidata: high speed processor
- Intergraph: processor
- Silicon Graphics': IRIS workstation
- Megatek Corp: Merlin 9200

Stonebraker proposes extending the relational database model with the use of abstract data types (ADT) [STON83] and speculates that ADTs may be supported by Relational Technology's INGRES by 1987 [STON86]. Many others are studying the use of databases to store graphical data. Some results are summarized in Table 2.2 in section 2.3.3 of this thesis.

The GKS standard was discussed and revised repeatedly until version 7.2 was submitted as a Draft International Standard in June 1982 [DUCE83].

MAP - Manufacturing Automation Protocol - was initiated by General Motors in 1982, and is being developed by the National Bureau of Standards (NBS). It is based on the IEEE network standards, and the seven layer OSI model defined by ISO. It is a factory device communications standard. The current version, 2.1, implements levels 1 through 4 of the OSI model [C/CA86b]. Industrial Technology Institute (ITI) will test and certify conformance of vendors to the MAP standards.

IGES Version 2.0 was released in February 1983. It supports transfer only of graphics and text for mechanical and electrical applications. Future versions may include support for solid modeling, numerical control, and robotics [STON85]. The advanced Geometry Subcommittee has been working since 1981 to include the ability to translate solid models with IGES [WAGN84].

XBF-2 (Experimental Boundary File-2) is a format developed by CAM-I (Computer Aided Manufacturing-International) for transfer of information between solid modeling systems [CAMI86]. XBF-2 is based

on IGES and supports BR and CSG translation.

An alternate standard also being developed by CAM-I, is called Applications Interface Specification (AIS), which can access the model data and perhaps build the XBF-2 file. It can be used by multiple systems to produce a consistent interface to application programs [CAMI86].

EDIF (Electronic Design Interchange Format) was proposed in 1983 to combine the good features of TDF, CIDF, GAIL, and TIDAL [LABU85]. Its development is supported by numerous corporations who do and or support electrical or circuit design. Version 0.8 was released in May 1985. EDIF is an extension of LISP.

STEP (STandard for the Exchange of Product data) has recently been proposed as a neutral format CAD/CAM database. It is intended as a worldwide format to replace American IGES, West German VDA-FS, and French SET formats [JONE86].

PDES (Product Design Exchange System) is due out in 1988 at the earliest. PDES is a format by which all product data can be passed among dissimilar systems [C/CA86b]. The PDES format will be replacing IGES, and equivalent to version 3.0 of IGES.

PHIGS (Programmer's Hierarchical Interactive Graphics System) is a third proposed graphics standard, more flexible and sophisticated than CORE and GKS. An implementation called Figaro was recently released by Template (Software division of Megatek Corp) [HECK86].

There is an increased awareness of the need to integrate the various computer aided parts of the engineering - manufacturing cycle, as evidenced by the proliferation of standard formats, and, conferences such as: "CAD/CAM Databases '85: Who's in Control?", and "CAD/CAM Databases '86: Control for the Decade Ahead" [C/CA86a]. Compared to many of today's CAD/CAM systems Sketchpad had superior functionality. This functionality is now being added back into the current products, once again making them more than just drafting aids.

## 3.5 Summary of Current Solid Modeling Packages

| Company | System Name | Packaged In | Represen | Oper | Hardware | Applic | Yr |
|---|---|---|---|---|---|---|---|
| Applicon | Solids Mod II | BRAVO CAD/CAM | BR,CSG | | D | A,D,F,K,M,N | 83 |
| Auto-trol | BMOD | 500d 2D or | BR | | D | in develop- | 83 |
| | CMOD | 7000 3D | CSG | | D | ment | 83 |
| Catronix | Catsolid | SA | CSG,BR | UI | A,D,HP,PC, S,T | D,F,I,M,T | 84 |
| CV | SOLIDESIGN | CDS4000 | BR | UID | CDS4000 | M,N | 83 |
| CDC | ICEM (SV) | ICEM | CSG,RC | | C,CDC | D,E,F,G,M,N | |
| E&S Shape Design | ROMULUS | Picture System | BR /SA | UID | A, AM, C, D, | A,F,M,N HP,I,P | 80 |
| GRAFTEK | ROMULUS | GMS | BR | UID | G | D,F,I,M,N | 84 |
| IBM | GDP | | PL | | | | |
| IBM | CATIA | CADAM | BR | UID | I | D,V | |
| ICM Computer | ICM-Geometrical Modeling System | SA | CSG,BR | UID | A,D | I,M,V | 81 |
| Intergraph | Solid Model Sys | Mech Des Sys | CSG,BR | UID | D | D,F,M,N | 69 |
| MAGI | SYNTHAVISION | SA | CSG,RC | | D,I,CDC, PE,SEL | D,I,M,T | 81 |
| Matra Datavision | EUCLID | SA | PL,CSG | UID | D,I | A,B,D,E,F, M,V | 70 |
| MDC | UNISOLIDS | SA | CSG,BR | UID | D,DG | A,I,M | 81 |
| Phoenix Data Systems | INSIGHT | SA | V,CSG,BR | | D Octree in memory | I,M | 83 |
| Prime | MEDUSA | 8 modules | PL,CSG | UID | P | D,M | |
| SDRC | GEOMOD | I-DEAS/SA | BR, CSG | UID | A,D,I | A,F,G,K,M | |
| Sperry | UNIS*CAD | UNIS*CAD | CSG,BR | UD | SP | D,F,G,N,P | 84 |

Table 3.1

## Key to Table 3.1

Most of the solid modeling packages available today are listed in Table 3.1 above. [REQU85] estimates that there were more than 20 solid modeling systems on the market in January of 1985, and the number is growing. The key below explains the abbreviations used. The contents of Table 3.1 was compiled from [JOHN84], [WAGN84], [REQU82], and [REQU84].

### Company:

The entries are listed in alphabetical order by the name of the company selling the solid modeling system. The abbreviations used for the vendor of the solids modeling package are:

| | |
|---|---|
| CV | = Computer Vision |
| CDC | = Control Data Coprporation |
| E&S | = Evans & Sutherland |
| GRAFTEK | = Graphics Technology Coproration |
| ICM | = Interactive Computer Modeling |
| MAGI | = Mathematics Applications Group Inc. |
| MDC | = McDonnell Douglas Corporation |
| SDRC | = Structured Dynamics Research Corporation |

### System Name:

The name of the solids modeling system.

| | |
|---|---|
| ICEM | = Integrated Computer Aided Engineering and Manufacturing |
| Mech Des Sys | = Mechanical Design System |
| SV | = Synthavision |

### Packaged In:

The name of the CAD/CAM system of which the solids system is a part. SA indicates Stand Alone. If the package name

is the same as the solids name then the solids system is part of (and included in) the system.

Representation:

The representation method(s) are listed in order of usage within the modeling system.

BR = Boundary Representation
CSG = Constructive Solid Geometry
O = Octree
PL = Polynomial Boundary Representation
RC = Ray Casting
V = Voxel

Op:

The Boolean Operators available in the solids modeling system:

D = Difference
I = Intersection
U = Union

Hardware:

The abbreviations for the hardware on which the solids modeling system can be run:

A = Apollo
AM = AMDAHL
C = Cyber
CDC = Control Data Corp
D = DEC/VAX
DG = Data General
G = Gould Computer Systems
HP = Hewlett Packard - HP9000
I = IBM
P = Prime
PC = IBM PC
PE = Perkin-Elmer
S = Sun Micro
SP = Sperry
T = Terak

Appl:

The applications available (to be used on the output of the solid modeling system) are listed in alphabetical order.

A = Assemblies
B = Bill of Materials
E = Drafting of Engineering Drawings
D = Database for "Drawing" datafiles
F = Finite Element Analysis
G = IGES translation
I = Interference Testing
K = Kinematics
M = Mass Properties
N = Numerical Control
P = Process Planning
V = Volume

Yr:

The year in which the solids system became commercially available.

Some conclusions about the current state of the art in Solid Modeling can be drawn from this information.

- All but two of the solid modeling systems were introduced in the 1980's. The systems implemented in 1969 and 1970 have changed a great deal since that time. Solid modeling is a young computer application.

- Almost all major CAD/CAM vendors supply some support for solid modeling, either as an addition to their existing

systems, or as a stand alone product.

• A total of twelve different applications were listed. An average of four applications were provided per solid modeling system. All but Auto-trol provided at least two applications. Clearly full support of applications has yet to be implemented.

## 4. Data Requirements Analysis

In this chapter the data required by the BR and CSG solid model representations will be examined, and diagrammed using an Entity-Relationship (E-R) diagram. Data requirements are determined by the functionality required of the solid modeling system. This thesis will consider the following functionality:

1. Creation of the solid model by the user

2. Display of the model, both
   - wireframe display and
   - hidden lines removed wireframe display.

3. Manipulation of the model including
   - Boolean Operations: Union, Difference, Intersection,
   - Unary Operations: Copy, Mirror, Rotation, Scale, Translation.

4. Storage and retrieval of the model
   - Geometric interrogation.

The functions listed below are useful in the mechanical engineering field. However, their detailed discussion is beyond the scope of this thesis.

- Volume calculations
- Mass properties calculations
- Tolerancing information/attributes attached to named features of the solid [BROW 81]
- Shaded model display is extremely helpful for the viewer.

Pseudo code is provided in this chapter to show the details of database access. Database access statements in the pseudo code are based on the Quel retrieval language used in Ingres [RELA86]. The remaining statements are in C like pseudo code.

## 4.1 Boundary Representation

A Boundary Representation (BR) consists of surfaces and faces, edges and vertices, which together define a complete solid object. As described in section 2.2.1, either polygon meshes or parametric bicubic patches may be used to represent complex or sculptured surfaces. In this chapter polygon meshes will be used to model boundary representations of solids for both complex surfaces and planar surfaces. Planar, or simple, faces may be represented by the edges and vertices on the boundary of each face, they do not require the further division into a mesh.

### 4.1.1 Data Stored to Create the BR Model

The user may specify vertices to indicate each corner of a face. Each face must have at least 3 vertices, and may have more. As the location of each vertex is specified, it is entered into the database. Each vertex in a solid must have a unique location. Therefore, multiple faces sharing a common vertex will refer to the same vertex instance in the database. A usage-count attribute in the vertex entity is necessary to keep track of how many faces contain the vertex. A vertex cannot be deleted if the usage-count is greater than zero. As each pair of connected vertices is created, an edge entity can be created in the edge table in the database. Similarly,

once multiple edges have been stored, a face may be created. The information about a face describes a planar polygon. One or more faces make up a surface.

When a solid model is created in an interactive mode it is important that the software be as friendly to the user as possible. The above method of input assumes that the user knows what the final solid or picture will look like, and knows the coordinates of all the vertices. Often, this is not the case. A more convenient method of input may be to define simple pieces or components of the model and combine them with boolean operators. This is similar to the method used when creating CSG solid models. Boolean operations on BR solid models are covered in section 4.1.3.

The pseudo code below shows subroutines used to create the various entities stored in the BR database. Emphasis in the pseudo code is on the database access statements. Refer to the Entity Relationship diagram in Figure 4.1 for record structures.

```
Create-vertex (X, Y, Z)
[
/* look for existing vertex */
retrieve (vertex-id = vertex.vertex-id,
          usage-count = vertex.usage-count
      where (vertex.X = X and vertex.Y = Y
         and vertex.Z = Z);
if (found = true) then
/*use existing vertex */
      replace vertex(usage-count = usage-count  + 1)
            where (vertex.vertex-id = vertex-id);
      return (vertex-id);
else
/* create new vertex */
      vertex-id = call generate-id;
      append vertex (vertex-id = vertex-id,
            X = X,
            Y = Y,
            Z = Z,
            usage-count = 1);
      return (vertex-id);
endif;
]


Create-edge (X1, Y1, Z1, X2, Y2, Z2)

[
/* find or create vertices of edge */
vertex-id1 = call create-vertex (X1, Y1, Z1);
vertex-id2 = call create-vertex (X2, Y2, Z2);

/* order vertex-id numbers */
if (vertex-id1 > vertex-id2) then
      temp = vertex-id1;
      vertex-id1 = vertex-id2;
      vertex-id2 = temp;
endif;

/* look for existing edge */
retrieve (edge-id = edge.edge-id,
          usage-count = edge.usage-count)
    where (edge.vertex-id1 = vertex-id1
        and edge.vertex-id2 = vertex-id2);

if (found = true) then
/* use existing edge */
    replace edge(usage-count = usage-count + 1)
        where (edge.edge-id = edge-id);
    return (edge-id);
else
/* create a new edge */
    edge-id = call generate-id;
```

```
    append edge (edge-id = edge-id,
        vertex-id1 = vertex-id1,
        vertex-id2 = vertex-id2,
        usage-count = 1);
    return (edge-id);
endif;
]
```

Create-face (surface-id, vertex-count, array of coordinates)

```
[
face-id = call genererate-id;
append face (face-id = face-id,
    edge-count = 0,
    surface-id = surface-id);

foreach edge /*pair of adjacent vertices*/
    [
/* find or create edge and store it */
    edge-id = call create-edge (X1,Y1,Z1,X2,Y2,Z2);
    replace face(edge-count = edge-count + 1)
        where (face.face-id = face-id);
    append face-edge (face-id = face-id,
        edge-id = edge-id);
    ]
endfor;
]
```

Create-surface (solid-id, surface-type, parameters)

```
[
surface-id = call generate-id;
if surface-type = planar then
/* one face for a planar surface */
    face-id = call create-face (array-of-coordinates);
    append surface (surface-id = surface-id,
        face-count = 1)
else
    append surface (solid-id = solid-id,
        surface-id = surface-id,
        face-count = 0);
    define mesh of faces
    foreach face
        face-id = call create-face
           (array-of-coordinates);
    end foreach;
endif;
]
```

```
Create-solid (parameters)
[
solid-id = call generate-id;
append solid (solid-id = solid-id,
    surface-count = 0);
foreach surface
      surface-id = call create-surface (solid-id,
          surface-type, parameters);
      replace solid(surface-count =
          solid.surface-count + 1);
endfor;
]
```

```
Generate-id (id-type)

/* Generate unique ids */
/* for each type of entity used so that each entity is
unique, even across solids withing the same database */
[
if (id-type = "solid") then
    begin transaction
        retrieve (id = unqiue-ids.solid-id);
        replace unique-ids (solid-id = id + 1)
            where (unique-ids.solid-id = id);
    end transaction;
    return (id);

else if (id-type = "surface") then
    begin transaction
        retrieve (id = unqiue-ids.surface-id);
        replace unique-ids (surface-id = id + 1)
            where (unique-ids.surface-id = id);
    end transaction;
    return (id);


else if (id-type = "face") then
    begin transaction
        retrieve (id = unqiue-ids.face-id);
        replace unique-ids (face-id = id + 1)
            where (unique-ids.face-id = id);
    end transaction;
    return (id);


else if (id-type = "edge") then
    begin transaction
        retrieve (id = unqiue-ids.edge-id);
        replace unique-ids (edge-id = id + 1)
            where (unique-ids.edge-id = id);
```

```
    end transaction;
    return (id);


else if (id-type = "vertex") then
    begin transaction
        retrieve (id = unqiue-ids.vertex-id);
        replace unique-ids (vertex-id = id + 1)
            where (unique-ids.vertex-id = id);
    end transaction;
    return (id);

else
    return (error-number);

endif;
]
```

The entities and their attributes stored at the creation of a Boundary Representation model are shown in Figure 4.1.

Certain constraints are necessary to ensure that a valid model is the result of the creation process. These constraints can be built into the subroutines which create the solid. For example, the routine which creates a vertex always checks first if the database contains a vertex with the same coordinates, and if so, does not create a duplicate.

Because a face contains multiple edges, and an edge is on the boundary of two faces there is a many to many relationship between faces and edges. This relationhip is represented by a relationship entity such as the Face-Edge entity in Figure 4.1.

### 4.1.1 Entity Relationship Diagram: BR

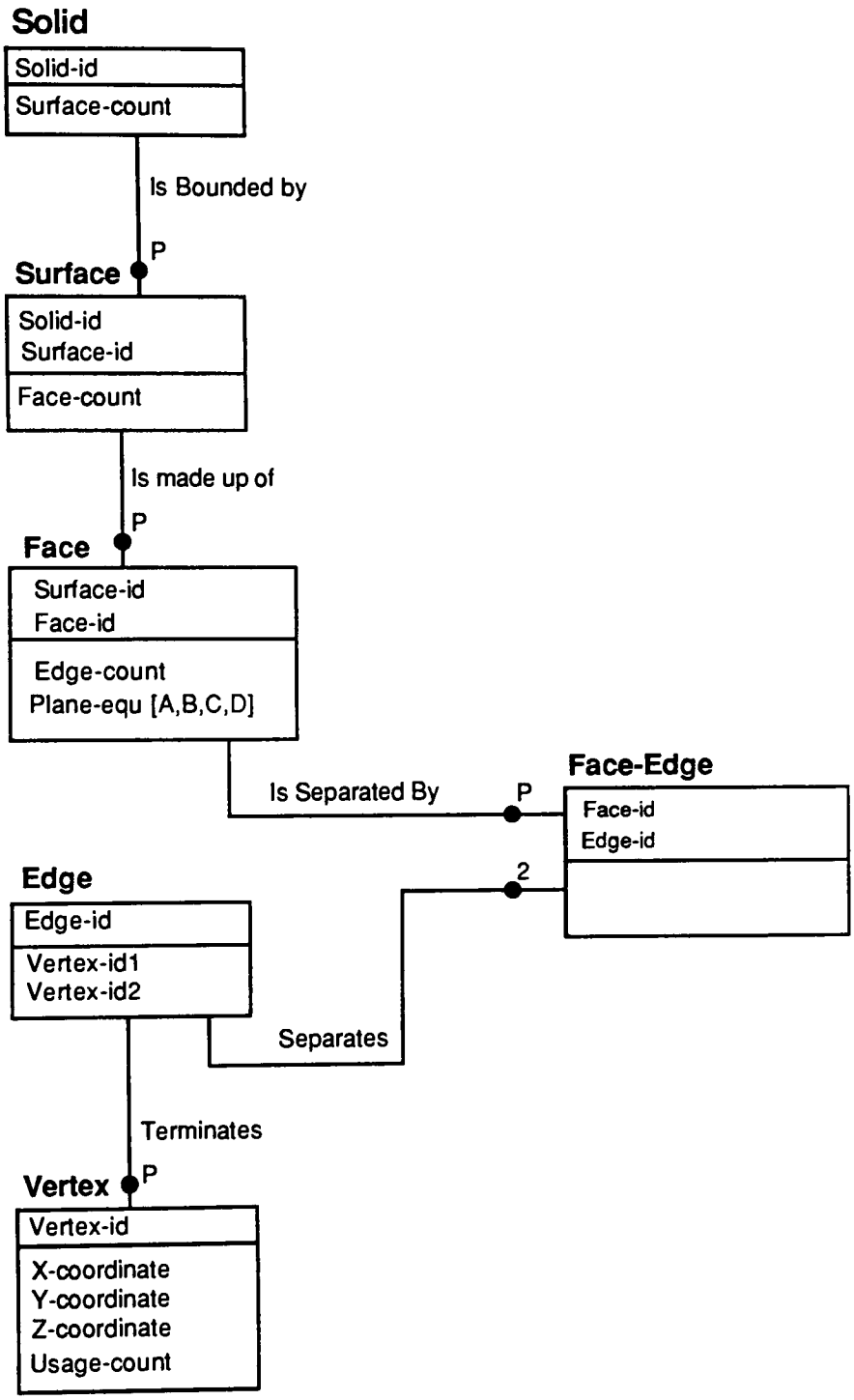



Figure 4.1

### 4.1.2 Display of the BR Model

#### 4.1.2.1 Wireframe Display

In a wireframe display the only parts of the model which are visible are edges. This information can easily be obtained from the Edge entity in the BR database. BR is very efficient for wireframe display of solids.

If the edges were NOT stored in the database, and each face stored only vertex ids, then the edges may be drawn or displayed by the following algorithm:

Display-Vertex-Pairs

```
[
Foreach (face-id) do begin
    foreach (pair of connected vertices) do begin
        display edge (x2,y2,z2)-(x1,y1,z1);
    end;
end;
]
```

However, we have stored information about each edge, in the Edge entity. This would eliminate the need to calculate edge information each time the solid is displayed. This makes the display process more efficient. The E-R diagram including Edge entities is shown in Figure 4.1.

In addition, it would be advantageous to prevent the drawing of edges that are co-incident in space. For example, in a cube there are 12 edges in all. Each edge belongs to two faces. The algorithm shown above will draw

each edge of the cube twice. This can be prevented by removing the loop for each face, and processing only the edge list:

Display-Edges

```
[
/* define short hand "range variables"*/

range of v1 is vertex;
range of v2 is vertex;

Foreach edge-id do begin
/* find both vertices */

    retrieve (x1 = v1.x, y1 = v1.y, z1 = v1.z,
        x2 = v2.x, y2 = v2.y, z2 = v2.z)
            where (edge.vertex-id1 = v1.vetex-id
                and edge.vertex-id2 = v2.vertex-id);

    draw the edge (x2,y2,z2)-(x1,y1,z1);
end;
]
```

4.1.2.2 Wireframe Display with Hidden Lines Removed

Further processing is necessary to indentify and remove hidden edges from the wireframe display. We must determine which edges are behind the solid, and thus invisible. Also, the part of the solid which is visible varies with the location of the viewpoint, ie. the location of the viewer.

The following is an object-space algorithm for hidden line removal. Calculations are performed with as much precision as possible, enabling the enlargement of the picture without loss of accuracy.

Plane Equation / Back-face Elimination:

This method works well on single convex objects. Using the equation for the plane of a face, and the location of the viewpoint for (x,y,z), each face can be tested with the equation ax+by+cz+d=0. If the result is negative then the viewpoint is behind the face, and the face is on the far side of (or inside) the solid with respect to the viewpoint. All faces identified in this way will not be visible and can be eliminated from further display calculations. The edges of the remaining faces are the only edges visible. This is also known as Back-face elimination.

The equation for a plane is made up of four numbers [a,b,c,d] so that ax+by+cz+d=0 when the point (x,y,z) lies exactly on the plane. If the point is outside the solid, the equation will yield a positive result. Conversely, if the point is inside the solid, (behind the plane) the result will be negative. The normal to the plane is given by: [abc] [NEWN79]. The values for a, b, c and d can be calculated from the x, y, and z coordinates of any three vertices on the face, and stored in the face entity when it is created. The following algorithm uses three points labled J, K, and L, the coordinates for J for example, being XJ, YJ, ZJ [BOWY83].

```
Calculate-Plane (face-id)
[
range of fe is face-edge;
range of e is edge;
range of v is vertex;

ctr = 0;

retrieve (x = v.x, y = v.y, z = v.z)
    where (fe.face-id = face-id
        and fe.edge-id = e.edge-id
        and e.vertex-id1 = v.vertex-id)
            sort by vertex-id;
/* sort eliminates duplicates */

      [
/* retrieve loop performed once for each row retrieved */

      ctr = ctr + 1;
      coords (ctr, 1) = x;
      coords (ctr, 2) = y;
      coords (ctr, 3) = z;
      ]

/* use first three rows of data for J, K, and L */
/* calculate A, B, C, and D */

XKJ = coords (2, 1) - (1, 1);
YKJ = coords (2, 2) - (2, 1);
ZKJ = coords (2, 3) - (3, 1);
XLJ = coords (3, 1) - (1, 1);
YLJ = coords (3, 2) - (1, 2);
ZLJ = coords (3, 3) - (1, 3);

A = YKJ * ZLJ - ZKJ * YLJ;
B = ZKJ * XLJ - XKJ * ZLJ;
C = XKJ * YLJ - YKJ * XLJ;
D = -(XK * A + YK * B + ZK * C);

replace face (A = A, B = B, C = C, D = D)
      where (face.face-id = face-id);
]
```

The following is an example of an image-space algorithm.

Overlap Test

The minmax test, also called the bounding box test, compares the minimum x coordinate of a polygon with the maximum x coordinate of another. The same ust is applied to y coordinates, and z coordinates if depth overlap must be determined. If the minmax test rules out overlap, uxen no further tests must be done on those faces. The remaining faces may or may not overlap, and additional processing is required.

The information about the bounding box can be stored in the database so that it is not necessary to recalculate each time it is needed. The min x, max x, min y, max y, min z, max z may be stored in the Face entity or the Edge entity, or some way of designating the vertices themselves could be devised. A consequence of this addition to the data model is that coordinate information would be stored in entities other than the vertex entity. This would complicate the unary operation algorithms as discussed in section 4.1.3, as well as increase redundancy in the data model. In this thesis the min-max data will not be stored, but calculated when necessary.

Geometric Sorting

Geometric sorting may be used to speed up the Overlap test

by arranging the edges to be compared for overlap next to each other in the sequence. The process is similar to, but more complicated than, alphabetical sorting [NEWM79]. This may provide a variable amount of benefit depending on the database model used to hold the data, because the data may already be sorted. In this thesis the data in the database will not be sorted, so this algorithm will provide improvement in the time required to execute the comparison algorithms on large solids.

### 4.1.3 Manipulation of the BR Model

Manipulation of a BR solid model falls into three categories: addition of new features to the solid, boolean operations, and unary opertions. Each is treated in a section below.

Boolean Operations:

Boolean operators operate on two solids combining them to create a single solid as output. In order to complete a boolean operation, the identity of both operands must be known. Operands for a boolean operation are BR solids.

The three boolean operators are:

- Addition of solids (Union: U)
- Subtraction of solids (Difference: -)
- Intersection of solids (Intersection: &)

As stated in section 2.2.2, all reference to boolean operations in this thesis assumes regularized operations.

Unlike the CSG representation, where another level in the tree structure is added and the operands remain unchanged, when boolean operations are performed on a BR, the two solids are merged into one. This is known as boundary merging. There is very little published about the details of boundary merging algorithms. The reader may consult several papers published by those working on PADL-2 at the

University of Rochester, including [REQU80], [REQU85], and [HART85d].

A boundary merging algorithm calculates the boundary of the resulting solid based on the boundaries of the two operands and the boolean operator. This must be done for entities at each level in the boundary: surface, face, edge, and vertex.

There are four possible scenarios for each entity: (surface, face, edge, and vertex) and each is described below:

1. A face, edge or vertex from an operand is included, unchanged in the new solid.

2. A face or edge from an operand is included in a modified form in the resulting solid.

> Modifications to a Face include: edges added, modified, deleted or divided into several parts.
>
> Modifications to an Edge include: Vertex or vetices changed, or the edge may be broken into several parts, and new vertices added.

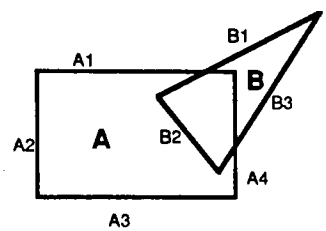
3. A face, edge or vertex from an operand is not included

in the new solid.

4. A new face edge or vertex is created in the new solid that did not exist in either operand.

These possible scenarios are illustrated with 2D examples for clarity:

Union Example

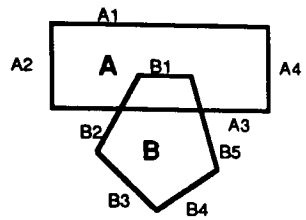



A = A1, A2, A3, A4 edges

B = B1, B2, B3 edges

A U B = unchanged: A2, A3
modified: B1, B3, A1, A4
split:
unused: B2
created:

## Difference Example

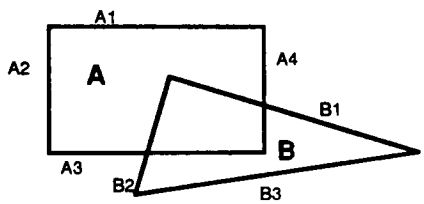



A = A1, A2, A3, A4 edges

B = B1, B2, B3, B4, B5 edges

A U B = unchanged: A1, A2, A4, B1
modified: B2, B5
split: A3
unused: B3, B4
created:

## Intersection Example




A = A1, A2, A3, A4 edges

B = B1, B2, B3 edges

A U B = unchanged: A1, A2
modified: A3, A4, B3
split:
unused: A1, A2, B3
created:

Unary Operations:

When performing unary operations, it is necessary to recalculate the position of all vertices affected and replace the new values in the database. Other entities do not require changes because they do not contain coordinate data and the topology of a solid does not change during unary operations. The user's view of unary operations is that one entity is input, and operated on, and the same entity is the output in its new location or orientation. This is true in all cases except the Copy operation.

A four by four matrix representation of transformations may be calculated from parameters the user supplies, and then used to perform matrix multiplication on the coordinates of the vertices in the solid. In all cases the pseudo code is the same, and the matrix effects the desired transformation of the solid.

Transformation Calculation (Matrix)

```
[
range of v is vertex;
range of fe is face-edge;
retrieve (V.X, V.Y, V.Z, vertex-id = v.vertex-id)
    where (solid.solid-id =  surface.solid-id)
        and (surface.surface-id = face.surface-id)
        and (face.face-id = fe.face-id)
        and (fe.edge-id = edge.edge-id)
        and (edge.vertex-id1 = v.vertex-id
        or edge.vertex-id2 = v.vertex-id);

/* matrix multiplication */

replace vertex (X = X, Y = Y, Z = Z)
    where vertex.vertex-id = vertex-id;
]
```

• Copy:

This is a special type of unary operator since the entity input is not actually changed as it is in the remaining unary operations. All entities in the solid are duplicated with a new set of unique ids. After the copy has been made other unary routines may be called by the user to position the copy.

```
Copy-Solid (Solid-id)
[
/* retrieve each part of solid */

/* call the create routine for each instance of each
entity */

]
```

• Mirror:

This requires the equation of the mirror plane through which the solid is mirrored. As discussed in section 4.1.2.2 three non-colinear points are sufficient to uniquely identify a plane. The user may use a face, an edge and a third point, or three points to define the plane desired. Mirror translation may be broken down into three steps:

1. Translate and/or rotate the solid's x-y axis to the location of the mirror plane
2. Invert the z coordinate
3. Un-move the axis to its original location

• Rotation:

An angle of rotation may be measured and input by the user in degrees or radians. A vector to indicate the axis of rotation requires (x,y,z) values. Rotation can be calculated for each of the axes (as shown below) and then the total rotation matrix calculated by multiplying them together.

X axis matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Y axix matrix

$$\begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Z axis matrix

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

• Scale:

The location (x,y,z) of the point about which the scaling is to be done and the scale factor in the form of a percentage will be specified by the user and used to calculate the matrix:

$$\begin{bmatrix} SX & 0 & 0 & 0 \\ 0 & SY & 0 & 0 \\ 0 & 0 & SZ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Translation:

The x, y, and z delta, or an indication of the new location coordinates of an origin point on the solid are necessary to position a solid. The delta values are determined, then used to create the matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ DX & DY & DZ & 1 \end{bmatrix}$$

### 4.1.4 Storage and Retrieval of the BR Model

The data representing the solid model will be stored in the database at all times. When a value is required a retrieval is executed to obtain a copy of the data in local memory. This is true in either interactive or batch mode. If the value is modified, it must be written back to the database upon completion of the calculation. This is done with the retrieve and replace commands:

```
Retrieve-Replace (vertex-id)
[
retrieve (tempx = v.x-coord,
          tempy = v.y-coord,
          tempz = v.z-coord)
      where (v.vertex-id = vertex-id)

/* new values calcualated for x, y, and z */

replace (v.x-coord = tempx,
         v.y-coord = tempy,
         v.z-coord = tempz)
    where (v.vertex-id = vertex-id)
]
```

At each level in the hierarchy (solid, surface, faces, edges, vertices) the identity of the sub-components must be determinable and data about them locatable. Conversely, given a vertex, it must be possible to identify and locate the edges, faces, surfaces, and the solid containing it.

The following code will list the vertices in a solid:

```
List-Vertices (Solid-id)
[
retrieve (v.vertex-id)
     where ((edge.vertex-id1 = v.vertex-id)
         or (edge.vertex-i2 = v.vertex-id))
         and (edge.edge-id = fe.edge-id)
         and (fe.face-id = face.face-id)
         and (face.surface-id = surface.surface-id)
         and (surface.solid-id = solid.solid-id)
        sort by vertex-id     /* eliminate duplicates */
]
```

Interactive selection of Entities by the User

When a user selects an entity on the screen, the coordinates of the cursor or crosshairs are obtained from the hardware and passed to the software. They are used as

an index into the database to locate the entity. The only entity which contains coordinate information is the vertex entity. Therefore, the search for information about a BR solid selected on the screen must begin at the vertex level. After locating the vertex (or vertices) that are at, or near, the indicated coordinates, the software must be able to determine which edges, faces and surfaces contain or surround it (them). This is done with the following retrieve routines:

```
Locate-Edge (x, y, z)
[
retrieve (edge.edge-id)

    where ((edge.vertex-id1 = vertex.vertex-id)
          or (edge.vertex-id2 = vertex.vertex-id))
          and (vertex.x = x
          and vertex.y = y
          and vertex.z = z);
]
```

```
Locate-Face (x, y, z)
[
retrieve (face.face-id)
    where (face.face-id = fe.face-id)
          and (fe.edge-id = edge.edge-id)
          and (edge.vertex-id1 = vertex.vertex-id
               or edge.vertex-id2 = vertex-id)
          and (vertex.x = x
          and vertex.y = y
          and vertex.z = z);
]
```

```
Show-Dimensions (face-id)
[
retrieve (x = vertex.x, y = vertex.y, z = vertex.z,
edge.edge-id)
    where (          )
        sort by edge-id
/* Calculate length of each edge */
]
```

The above routines may find zero, one, or many entities, depending on the location of the crosshairs on the screen relative to the entities in the solid. It is important that the entities be presented to the user in a friendly manner. If there are no exact matches, then entities should be retrieved if they are within a certain tolerance of the x, y, z specified, and presented to the user in order from closest to furthest away.

Questions a user would want to ask about a BR solid include:

- How many surfaces does solid A contain?
- What is the length of edge B?
- How many edges are boundaries of face C?

## 4.2 Constructive Solid Geometry

Solids represented by Constructive Solid Geometry consist of primitives that, when instanced, positioned and combined by boolean operators, define a complete solid object. CSG was chosen for analysis because, as described in sections 2.2.1 and 3.5, it is currently one of the most widely used methods of representing solids.

### 4.2.1 Data Stored to Create the CSG Model

To create a CSG model the user selects, sizes and positions one or more primitives. Instances of primitives are then combined using boolean operators. The boolean operators used to combine solids and the unary operators used to position primitives and/or solids are the same ones used to manipulate the model later in its life cycle.

For each instance of a primitive it is necessary to know the following information:

- Primitve Id (Solid-id in Solid entity)
- Required parameters for the type of primitive
- Primitive Type
- Location and Orientation of the primitive

Information about a primitive is stored in the database in three entities: (1) the Solid entity contains the type of

solid, in this case a primitive; (2) the primitive entity holds the primitive-id, primitive-type, and motion matrix (a four by four matrix); (3) a specialized group of primitive entities hold the primitive-id, and the parameters necessary for each type of primitive. These entities are depicted in figure 4.2.

The following code shows the creation of a block entity:

```
Create-Block (type, motion-matrix, width,
                    height, length)
[
prim-id = call generate-id;
append solid (solid-id = prim-id,
      solid-type = "P");
append prim (prim-id = prim-id,
      prim-type = type,
      motion-matrix = motion-matrix);
append block (prim-id = prim-id,
      prim-width = width,
      prim-height = height,
      prim-length = length);
return;
]
```

When a primitive is created it can be given default orientation at the origin by using the identity matrix for the motion matrix. The user may however specify a transformation which is translated into a 4 by 4 matrix.

For each entity that is a boolean combination of two other entities the following information is required:

- A flag to indicate that it is a compound solid
- Unique solid-id

- Operator
- Operand1-id
- Operand2-id

The operands may be primitives or previously created compound solids. This information is stored in uue Compound-solid entity and the Solid entity. The pseudo code to create and store an entity appears in section 4.2.3 on boolean operations. Uue E-R diagram for a CSG model stored in a database is shown in Figure 4.2.

## Entity Relationship Diagram: CSG

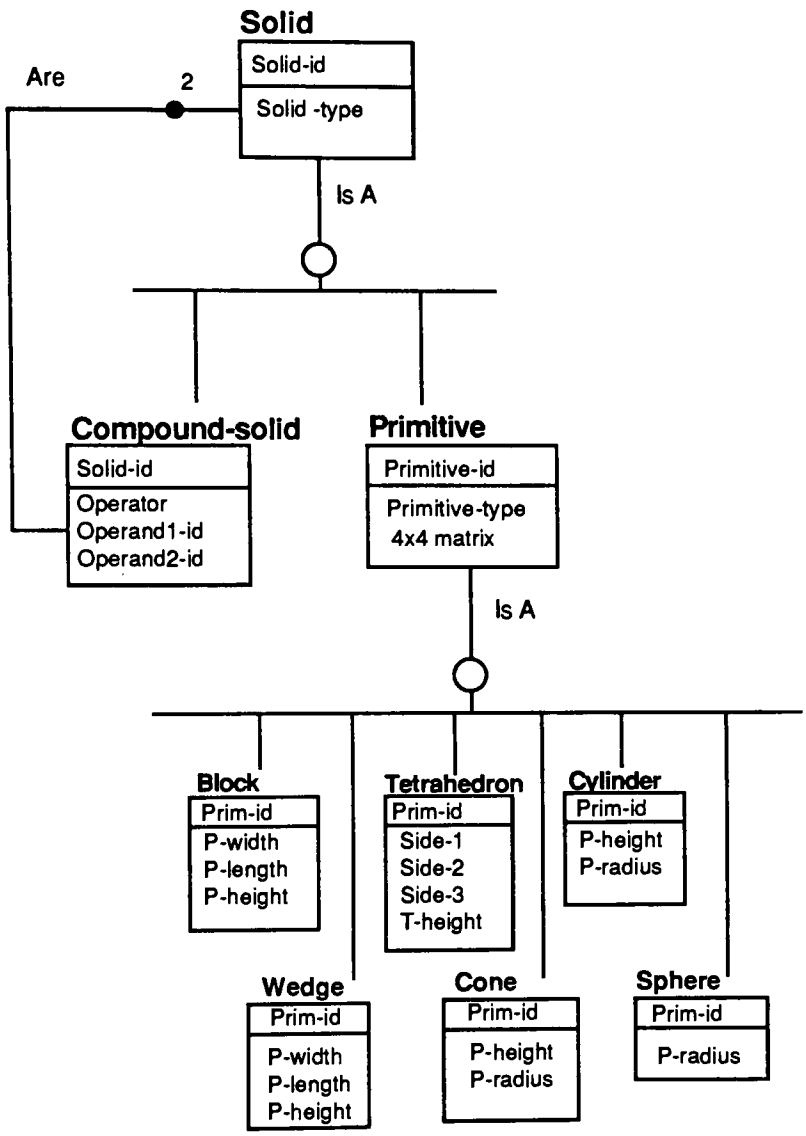



Figure 4.2

### 4.2.2 Display of the CSG Model

#### 4.2.2.1 Display of a Wireframe

There are currently no efficient algorithms for generating a wireframe display from a CSG model. Most CSG systems convert the CSG model to a BR at this point. Even the overhead of conversion is more efficient than available CSG display methods. For example, the PADL2 solid modeling system primarily uses CSG for internal representation, and converts the solid to BR for display of the solid [HART85a]. There are two conversion approaches, each with advantages and drawbacks:

Non-Incremental Conversion

Each time a display is required the entire picture is converted from CSG to BR. This approach is useful when a given solid is translated only once. In the dynamic environment of creation and modification of solid models, this approach is not efficient. This method was used by PADL-1.

Incremental Conversion

This method involves storing the BR for the sub-parts and using that information when a solid containing the parts is converted at a later time. This requires that the results (the BR representation) of each conversion be retained for use in subsequent conversion process(es).

The same structure as that used to hold the BR described in section 4.1 can be used and the boolean operations outlined in section 4.1.3 can be used to combine them. This method is used by PADL-2 [REQU85]. This approach works well in the dynamic and repetative environment of model creation and modification, and will therefore be used in this thesis.

It is necessary to convert each primitive only once. Some method of tagging those already translated will prevent repreat translations. An alternative is translation done at the time the entity is created. This requires no additional storage and will be the technique used in this thesis.

It is necessary to convert only primitives directly from CSG to BR. The primitives in compound solids are translated, and combined via BR boundary merge techniques proposed in section 4.1.3.

When translating a CSG primitive to a BR solid, it is necessary to determine the BR entities contained in the primitive. [HART85d] explains the incremental conversion process used in PADL-2, including several ways of optimizing the process. For example, to translate a block primitive, we must create six surfaces, six faces must be

identified, the faces can be intersected to locate the 12 edges which bound the faces. Similarly, the edges are intersected to determine the vertices' coordinate values. Once converted, the BR methods are used to create the display of the solid model as outlined in section 4.1.2.

#### 4.2.2.2 Display of a Wireframe with Hidden Lines Removed

The same methods as above are used to convert from CSG to BR. Then the methods of hidden line removal outlined in section 4.1.2.2 are used to display the solid.

### 4.2.3 Manipulation of the CSG Model

Manipulation, or editing the CSG solid model involves boolean operations used to combine two solids or primitives, and unary operations to transform existing primitives. These two types of editing a CSG solid model are quite different, and are discussed separately below.

**Boolean Operations:**

Boolean operations are used extensively in CSG systems to combine primitives and compound solids to create increasingly complex solids. The three boolean operators are:

- Addition (Union:U)
- Subtraction (Difference:-)
- Intersection (Intersection: &)

To complete a boolean operation, the identity of both operand entities must be known. Operands may be primitives or compound solids created by previous boolean operations. As pictured in the data model in figure 4.2, a boolean operation requires the creation of a new solid entity and a new compound-solid entity.

```
Boolean-Op (operand-id1, operand-id2, operator)
[
solid-type = "C"
solid-id = call generate-id
append compsolid (solid-id = solid-id,
      operator = operator,
      operand-id1 = operand-id1,
      operand-id2 = operand-id2)
append solid (solid-id = solid-id,
      solid-type = solid-type)
return;
]
```

There is no limit to the number of boolean operations which may be used to create a complex solid.

## Unary Operators

An unary operator alters the rigid motion matrix of one operand. This matrix is stored at the time a primitive is instanced, as shown in section 4.2.1 and may be modified

subsequently in the editing process by unary operators. An Unary operation will take the input provided by the user, calculate the 4 by 4 matrix representing the transformation, and multiply it by the values in the motion matrix to create the new motion matrix. As with the BR model in section 4.1.3, rigid motion is represented by a four by four matrix. Both primitive and compound entities may be operands for unary operations, however each primitive in a compound entity is operated on individually.

• Copy: This is handled in much the same way as BR handles the copy operation. A complete copy is made of the operand, and then it is repositioned as desired by calling upon the other unary operators.

```
Copy (operand-id)
[
retrieve (operand-type = solid.solid-type)
    where solid.solid-id = operand-id;

if (operand-type = "P") then
    /* copy the primitive */
    retrieve (prim-type = prim.prim-type,
        motion-matrix = prim.motion-matrix)
        where prim.prim-id = operand-id;
    if (prim-type = "BL") then
        retrieve (width = block.p-width,
            length = block.p-length,
            height = block.p-height)
            where block.prim-id = operand-id;
        call createblock (type, motion-matrix, width,
          length, height);
    else
```

```
        /* have an else branch for each primitive
           type supported */
    endif;
    return;

else if (operand-type = "C") then
    /* copy each part */
    retrieve (operand1 = comp-solid.operand1-id,
        operand2 = comp-solid.operand2-id)
        where comp-solid.solid-id = operand-id;
    call copy (operand1);
    call copy (operand2);
    return;
endif;
return;
]
```

The matrices for the following unary operators are calculated in the same manner as BR matrices. See section 4.1.3 for example matrices.

• Mirror: The user will specify the location of the mirror plane. The mirror process is then broken into three steps:

1. Translate and/or rotate the solid's x-y axis to the location of the mirror plane
2. Invert the Z coordinate
3. Un-move the axix to its original location.

However, if the mirror plane lines on the X=0, the Y=0, or the Z=0 plane, the appropriate matrix (shown below) may be multiplied against the entity's motion matrix [MORT85].

x=0 plane

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

y=0 plane

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

z=0 plane

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

- Rotation: As with the BR representation, the rotation matrix for each axis is created, then multipled together to produce the composite rotation matrix. This is then multiplied by the matrix stored in the primitive, which is then replaced by the result.

- Scale:

The scale operation is handled by changing the sizing information in the primitive entity. The user will input the coordinates of the point about which the sizing will be done, and a scaling factor.

```
Scale-Block (prim-id)
[
retrieve (width = block.p-width,
    length = block.p-length
    height = block.p-height)
    where block.prim-id = prim-id;
```

```
/* Scale calculations */

replace block (p-width = new-width,
    p-length = new-length
    p-height = new-height)
    where block.prim-id = prim-id;
return;
]
```

- Translation: The X, Y, and Z displacement are used to create the translation matrix.

### 4.2.4 Storage and Retrieval of the CSG Model

The CSG data is stored in the database. Values are retrieved and updated when necessary, in the same manner as the BR model described in section 4.1.4. Many examples of appending, retrieving, and replacing data are shown throughout section 4.2. Users must have the ability to ask questions about the model.

- What are the extents of the solid (outer boundaries)?
- What are the dimensions of primitive block A?
- List all the primitives in solid B.
- What is the angle of rotation of primitive C?

The following routines show how these questions might be answered by the solid modeling system.

```
List-Primitives (Solid-id)
[
retrieve (prim-id = prim.prim-id,
  prim-type = prim.prim.type)
      where (prim.prim-id = solid-id)
      sort by prim-type, prim-id;
/* List all primitives in order by type and id */
return;
]


Show-Motion (prim-id)
[
retrieve (matrix = Primitive.matrix)
    where (prim.prim-id = prim-id);
/* Display matrix OR convert to a user-friendly
   format */
return;
]


Show-Dimensions (prim-id)
[
retrieve (prim-type = prim.prim-type)
    where prim.prim-id = prim-id;
if (prim-type = "BL") then
    retrieve (width = block.prim-width,
        height = block.prim-height,
        length = block.prim-length)
        where (block.prim-id = prim-id)
        sort by prim-width;
    /* Show the dimensions of the primitive */
else
    /* create an else if branch for each type of
       primitive supported */
endif;
return;
]
```

## 4.3 Solid Modeling Example

This section shows examples of the data necessary to represent a solid. The solid is shown in Figure 4.3. All entities and the data they contain are presented. Data for both BR and a CSG representations is generated from this solid. To limit the amount of data and still have a useful example, a relatively simple solid was chosen.

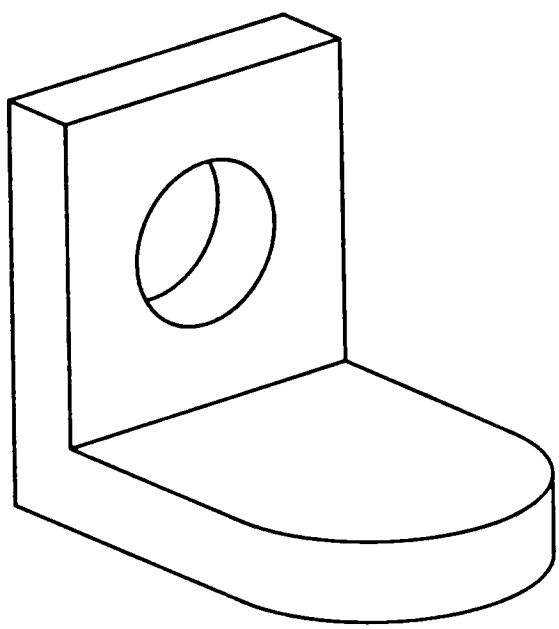


Figure 4.3

### 4.3.1 BR Example

Refer to the Entity Relationship diagram in Figure 4.1 for the list of entities used to store BR solid models.

This example shows how cylindrical surfaces are filled

with a mesh of planar faces in the BR representation. The following formulae are used to determine the size and number of faces in a mesh.

```
circumference = 2 * Pi * Radius
number-parts =  SquareRoot (circumference * 18)
number-degrees in 1 part = 360/ number-parts
```

The entities and data necessary to store a BR model of the example solid are listed on the following pages. The title(s) of key field(s) (those fields which uniquely identify a tuple of data) appear in italics.

Entity Name: Solid

| *ID* | Surface-Count |
|---|---|
| 1 | 7 |

Entity Name: Surface

| Solid-Id | *Surface-Id* | Face-Count |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 2 | 1 |
| 1 | 3 | 1 |
| 1 | 4 | 1 |
| 1 | 5 | 1 |
| 1 | 6 | 1 |
| 1 | 7 | 1 |
| 1 | 8 | 11 |
| 1 | 9 | 18 |

Entity Name: Face

Several exemplary faces were chosen to show calculations of the columns A, B, C, and D in the Face entity. The calculations used are shown below.

Face: 1
J (vertex 1) = 0, 0, 4
K (vertex 2) = 0, 5, 4
L (vertex 3) = 0, 5, 0

XKJ = 0 - 0 = 0
YKJ = 5 - 0 = 5
ZKJ = 4 - 4 = 0
XLJ = 0 - 0 = 0
YLJ = 5 - 0 = 5
ZLJ = 0 - 4 = -4

A = 5 * -4 - 0 * 5 = -20
B = 0 * 0 - 0 * -4 = 0
C = 0 * 5 - 5 * 0 = 0
D = -(0 * -20 + 5 * 0 + 4 * 0) = 0

Face: 4
J (vertex 17) = 5.73, 1, 3
K (vertex 21) = 5.97, 1, 1.65
L (vertex 9) = 4, 1, 0

XKJ = 5.97 - 5.73 = 0.24
YKJ = 1 - 1 = 0
ZKJ = 1.65 - 3 = -1.35
XLJ = 4 - 5.73 = -1.73
YLJ = 1 - 1 = 0
ZLJ = 0 - 3 = -3

A = 0 * -3 - -1.73 * 0 = 0
B = -1.35 * -1.73 - .24 * -3 = 3.0555
C = 0.24 * 0 - 0 * -1.73 = 0
D = -(5.97 * 0 + 1 * 3.0555 + 1.65 * 0) = -3.0555

Face: 8
J (vertex 11) = 4, 0, 4
K (vertex 12) = 4, 1, 4
L (vertex 13) = 4.46, 1, 3.88

XKJ = 4 - 4 = 0
YKJ = 1 - 0 = 1
ZKJ = 4 - 4 = 0
XLJ = 4.68 - 4 = 0.68
YLJ = 1 - 0 = 1
ZLJ = 3.88 - 4 = 0.12

A = 1 * 0.12 - 0 * 1 = 0.12
B = 0 * 0.68 - 0 * 0.12 = 0
C = 0 * 1 - 1 * 0.68 = 0.68
D = -(4 * 0.12 + 1 * 0 + 4 * 0.68) = -3.2

| Surface-Id | Face-Id | Edge-Count | Plane-Equation | | | |
|---|---|---|---|---|---|---|
| | | | A | B | C | D |
| 1 | 1 | 22 | -20 | 0 | 0 | 0 |
| 2 | 2 | 22 | | | | |
| 3 | 3 | 4 | | | | |
| 4 | 4 | 12 | 0 | 3.0555 | 0 | -3.0555 |
| 5 | 5 | 12 | | | | |
| 6 | 6 | 6 | | | | |
| 6 | 7 | 6 | | | | |
| 6 | 8 | 4 | 0.12 | 0 | 0.68 | -3.2 |
| 6 | 9 | 4 | | | | |
| 6 | 10 | 4 | | | | |
| 6 | 11 | 4 | | | | |
| 6 | 12 | 4 | | | | |
| 6 | 13 | 4 | | | | |
| 6 | 14 | 4 | | | | |
| 6 | 15 | 4 | | | | |
| 6 | 16 | 4 | | | | |
| 7 | 17 | 4 | | | | |
| 7 | 18 | 4 | | | | |
| 7 | 19 | 4 | | | | |
| 7 | 20 | 4 | | | | |
| 7 | 21 | 4 | | | | |
| 7 | 22 | 4 | | | | |
| 7 | 23 | 4 | | | | |
| 7 | 24 | 4 | | | | |
| 7 | 25 | 4 | | | | |
| 7 | 26 | 4 | | | | |
| 7 | 27 | 4 | | | | |
| 7 | 28 | 4 | | | | |
| 7 | 29 | 4 | | | | |
| 7 | 30 | 4 | | | | |
| 7 | 31 | 4 | | | | |
| 7 | 32 | 4 | | | | |
| 7 | 33 | 4 | | | | |
| 7 | 34 | 4 | | | | |

Entity Name: Face-Edge

| *Face-Id* | *Edge-Id* |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 1 | 4 |
| 1 | 61 |
| 1 | 62 |
| 1 | 63 |
| 1 | 64 |
| 1 | 65 |
| 1 | 66 |
| 1 | 67 |
| 1 | 68 |
| 1 | 69 |
| 1 | 70 |
| 1 | 71 |
| 1 | 72 |
| 1 | 73 |
| 1 | 74 |
| 1 | 75 |
| 1 | 76 |
| 1 | 77 |
| 1 | 78 |
| 2 | 5 |
| 2 | 6 |
| 2 | 7 |
| 2 | 8 |
| 2 | 43 |
| 2 | 44 |
| 2 | 45 |
| 2 | 46 |
| 2 | 47 |
| 2 | 48 |
| 2 | 49 |
| 2 | 50 |
| 2 | 51 |
| 2 | 52 |
| 2 | 53 |
| 2 | 54 |
| 2 | 55 |
| 2 | 56 |
| 2 | 57 |
| 2 | 58 |
| 2 | 59 |
| 2 | 60 |
| 3 | 2 |
| 3 | 6 |
| 3 | 9 |
| 3 | 10 |

Entity Name: Face-Edge (Cont.)

| *Face-Id* | *Edge-Id* |
|---|---|
| 4 | 8 |
| 4 | 11 |
| 4 | 14 |
| 4 | 17 |
| 4 | 20 |
| 4 | 23 |
| 4 | 26 |
| 4 | 29 |
| 4 | 32 |
| 4 | 35 |
| 4 | 38 |
| 4 | 41 |
| 5 | 4 |
| 5 | 12 |
| 5 | 13 |
| 5 | 18 |
| 5 | 21 |
| 5 | 24 |
| 5 | 27 |
| 5 | 30 |
| 5 | 33 |
| 5 | 36 |
| 5 | 39 |
| 5 | 42 |
| 6 | 1 |
| 6 | 5 |
| 6 | 9 |
| 6 | 13 |
| 6 | 14 |
| 6 | 15 |
| 7 | 3 |
| 7 | 7 |
| 7 | 10 |
| 7 | 11 |
| 7 | 12 |
| 7 | 16 |
| 8 | 15 |
| 8 | 17 |
| 8 | 18 |
| 8 | 19 |
| 9 | 19 |
| 9 | 20 |
| 9 | 21 |
| 9 | 22 |
| 10 | 22 |
| 10 | 23 |
| 10 | 24 |
| 10 | 25 |

Entity Name: Face-Edge (Cont.)

| *Face-Id* | *Edge-Id* |
|---|---|
| 11 | 25 |
| 11 | 26 |
| 11 | 27 |
| 11 | 28 |
| 12 | 28 |
| 12 | 20 |
| 12 | 30 |
| 12 | 31 |
| 13 | 31 |
| 13 | 32 |
| 13 | 33 |
| 13 | 34 |
| 14 | 34 |
| 14 | 35 |
| 14 | 36 |
| 14 | 37 |
| 15 | 37 |
| 15 | 38 |
| 15 | 39 |
| 15 | 40 |
| 16 | 16 |
| 16 | 40 |
| 16 | 41 |
| 16 | 42 |
| 17 | 43 |
| 17 | 61 |
| 17 | 79 |
| 17 | 80 |
| 18 | 44 |
| 18 | 62 |
| 18 | 80 |
| 18 | 81 |
| 19 | 45 |
| 19 | 63 |
| 19 | 81 |
| 19 | 82 |
| 20 | 46 |
| 20 | 64 |
| 20 | 82 |
| 20 | 83 |
| 21 | 47 |
| 21 | 65 |
| 21 | 83 |
| 21 | 84 |
| 22 | 48 |
| 22 | 66 |
| 22 | 84 |
| 22 | 85 |

Entity Name: Face-Edge (Cont.)

| *Face-Id* | *Edge-Id* |
|---|---|
| 23 | 49 |
| 23 | 67 |
| 23 | 85 |
| 23 | 86 |
| 24 | 50 |
| 24 | 68 |
| 24 | 86 |
| 24 | 87 |
| 25 | 51 |
| 25 | 69 |
| 25 | 87 |
| 25 | 88 |
| 26 | 52 |
| 26 | 70 |
| 26 | 88 |
| 26 | 89 |
| 27 | 53 |
| 27 | 71 |
| 27 | 89 |
| 27 | 90 |
| 28 | 54 |
| 28 | 72 |
| 28 | 90 |
| 28 | 91 |
| 29 | 55 |
| 29 | 73 |
| 29 | 91 |
| 29 | 92 |
| 30 | 56 |
| 30 | 74 |
| 30 | 92 |
| 30 | 93 |
| 31 | 57 |
| 31 | 75 |
| 31 | 93 |
| 31 | 94 |
| 32 | 58 |
| 32 | 76 |
| 32 | 94 |
| 32 | 95 |
| 33 | 59 |
| 33 | 77 |
| 33 | 95 |
| 33 | 96 |
| 34 | 60 |
| 34 | 78 |
| 34 | 96 |
| 34 | 97 |

Entity Name: Edge

| *Edge-Id* | Vertex-Id1 | Vertex-Id2 |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 2 | 3 |
| 3 | 3 | 4 |
| 4 | 4 | 1 |
| 5 | 5 | 6 |
| 6 | 6 | 7 |
| 7 | 7 | 8 |
| 8 | 8 | 5 |
| 9 | 2 | 6 |
| 10 | 3 | 7 |
| 11 | 8 | 9 |
| 12 | 4 | 10 |
| 13 | 1 | 11 |
| 14 | 5 | 12 |
| 15 | 11 | 12 |
| 16 | 9 | 10 |
| 17 | 12 | 13 |
| 18 | 11 | 14 |
| 19 | 13 | 14 |
| 20 | 13 | 15 |
| 21 | 14 | 16 |
| 22 | 15 | 16 |
| 23 | 15 | 17 |
| 24 | 16 | 18 |
| 25 | 17 | 18 |
| 26 | 17 | 19 |
| 27 | 18 | 20 |
| 28 | 19 | 20 |
| 29 | 19 | 21 |
| 30 | 20 | 22 |
| 31 | 21 | 22 |
| 32 | 21 | 23 |
| 33 | 22 | 24 |
| 34 | 23 | 24 |
| 35 | 23 | 25 |
| 36 | 24 | 26 |
| 37 | 25 | 26 |
| 38 | 25 | 27 |
| 39 | 26 | 28 |
| 40 | 27 | 28 |
| 41 | 9 | 27 |
| 42 | 10 | 28 |
| 43 | 29 | 30 |
| 44 | 30 | 31 |
| 45 | 31 | 32 |
| 46 | 32 | 33 |
| 47 | 33 | 34 |

Entity Name: Edge (Cont.)

| Edge-Id | Vertex-Id1 | Vertex-Id2 |
|---|---|---|
| 48 | 34 | 35 |
| 49 | 35 | 36 |
| 50 | 36 | 37 |
| 51 | 37 | 38 |
| 52 | 38 | 39 |
| 53 | 39 | 40 |
| 54 | 40 | 41 |
| 55 | 41 | 42 |
| 56 | 42 | 43 |
| 57 | 43 | 44 |
| 58 | 44 | 45 |
| 59 | 45 | 46 |
| 60 | 29 | 46 |
| 61 | 47 | 48 |
| 62 | 48 | 49 |
| 63 | 49 | 50 |
| 64 | 50 | 51 |
| 65 | 51 | 52 |
| 66 | 52 | 53 |
| 67 | 53 | 54 |
| 68 | 54 | 55 |
| 69 | 55 | 56 |
| 70 | 56 | 57 |
| 71 | 57 | 58 |
| 72 | 58 | 59 |
| 73 | 59 | 60 |
| 74 | 60 | 61 |
| 75 | 61 | 62 |
| 76 | 62 | 63 |
| 77 | 63 | 64 |
| 78 | 47 | 64 |
| 79 | 29 | 47 |
| 80 | 30 | 48 |
| 81 | 31 | 49 |
| 82 | 32 | 50 |
| 83 | 33 | 51 |
| 84 | 34 | 52 |
| 85 | 35 | 53 |
| 86 | 36 | 54 |
| 87 | 37 | 55 |
| 88 | 38 | 56 |
| 89 | 39 | 57 |
| 90 | 40 | 58 |
| 91 | 41 | 59 |
| 92 | 42 | 60 |
| 93 | 43 | 61 |
| 94 | 44 | 62 |
| 95 | 45 | 63 |
| 96 | 46 | 64 |

**Entity Name:** Vertex

| ID | X | Y | Z | Usage-count |
|---|---|---|---|---|
| 1 | 0 | 0 | 4 | 3 |
| 2 | 0 | 5 | 4 | 3 |
| 3 | 0 | 5 | 0 | 3 |
| 4 | 0 | 0 | 0 | 3 |
| 5 | 1 | 1 | 4 | 3 |
| 6 | 1 | 5 | 4 | 3 |
| 7 | 1 | 5 | 0 | 3 |
| 8 | 1 | 1 | 0 | 3 |
| 9 | 4 | 1 | 0 | 3 |
| 10 | 4 | 0 | 0 | 3 |
| 11 | 4 | 0 | 4 | 3 |
| 12 | 4 | 1 | 4 | 3 |
| 13 | 4.68 | 1 | 3.88 | 3 |
| 14 | 4.68 | 0 | 3.88 | 3 |
| 15 | 5.28 | 1 | 3.53 | 3 |
| 16 | 5.28 | 0 | 3.53 | 3 |
| 17 | 5.73 | 1 | 3 | 3 |
| 18 | 5.73 | 0 | 3 | 3 |
| 19 | 5.97 | 1 | 2.35 | 3 |
| 20 | 5.97 | 0 | 2.35 | 3 |
| 21 | 5.97 | 1 | 1.65 | 3 |
| 22 | 5.97 | 0 | 1.65 | 3 |
| 23 | 5.73 | 1 | 1 | 3 |
| 24 | 5.73 | 0 | 1 | 3 |
| 25 | 5.28 | 1 | 0.47 | 3 |
| 26 | 5.28 | 0 | 0.47 | 3 |
| 27 | 4.68 | 1 | 0.12 | 3 |
| 28 | 4.68 | 0 | 0.12 | 3 |
| 29 | 1 | 2.22 | 2.6 | 3 |
| 30 | 1 | 2.05 | 2.35 | 3 |
| 31 | 1 | 2 | 2 | 3 |
| 32 | 1 | 2.05 | 1.65 | 3 |
| 33 | 1 | 2.22 | 1.35 | 3 |
| 34 | 1 | 2.5 | 1.1 | 3 |
| 35 | 1 | 2.82 | 1 | 3 |
| 36 | 1 | 3.18 | 1 | 3 |
| 37 | 1 | 3.5 | 1.1 | 3 |
| 38 | 1 | 3.78 | 1.35 | 3 |
| 39 | 1 | 3.95 | 1.65 | 3 |
| 40 | 1 | 4 | 2 | 3 |
| 41 | 1 | 3.95 | 2.35 | 3 |
| 42 | 1 | 3.78 | 2.6 | 3 |
| 43 | 1 | 3.5 | 2.85 | 3 |
| 44 | 1 | 3.18 | 3 | 3 |

Entity Name: Vertex (Cont.)

| ID | X | Y | Z | Usage-count |
|---|---|---|---|---|
| 45 | 1 | 2.82 | 3 | 3 |
| 46 | 1 | 2.5 | 2.85 | 3 |
| 47 | 0 | 2.22 | 2.6 | 3 |
| 48 | 0 | 2.05 | 2.35 | 3 |
| 49 | 0 | 2 | 2 | 3 |
| 50 | 0 | 2.05 | 1.65 | 3 |
| 51 | 0 | 2.22 | 1.35 | 3 |
| 52 | 0 | 2.5 | 1.1 | 3 |
| 53 | 0 | 2.82 | 1 | 3 |
| 54 | 0 | 3.18 | 1 | 3 |
| 55 | 0 | 3.5 | 1.1 | 3 |
| 56 | 0 | 3.78 | 1.35 | 3 |
| 57 | 0 | 3.95 | 1.65 | 3 |
| 58 | 0 | 4 | 2 | 3 |
| 59 | 0 | 3.95 | 2.35 | 3 |
| 60 | 0 | 3.78 | 2.6 | 3 |
| 61 | 0 | 3.5 | 2.85 | 3 |
| 62 | 0 | 3.18 | 3 | 3 |
| 63 | 0 | 2.82 | 3 | 3 |
| 64 | 0 | 2.5 | 2.85 | 3 |

### 4.3.2 CSG Example

Refer to the Entity Relationship diagram in Figure 4.2 for the entities used to store CGS solid models. The example used is the same as that used for the BR example. This example uses the block and cylinder primitives.

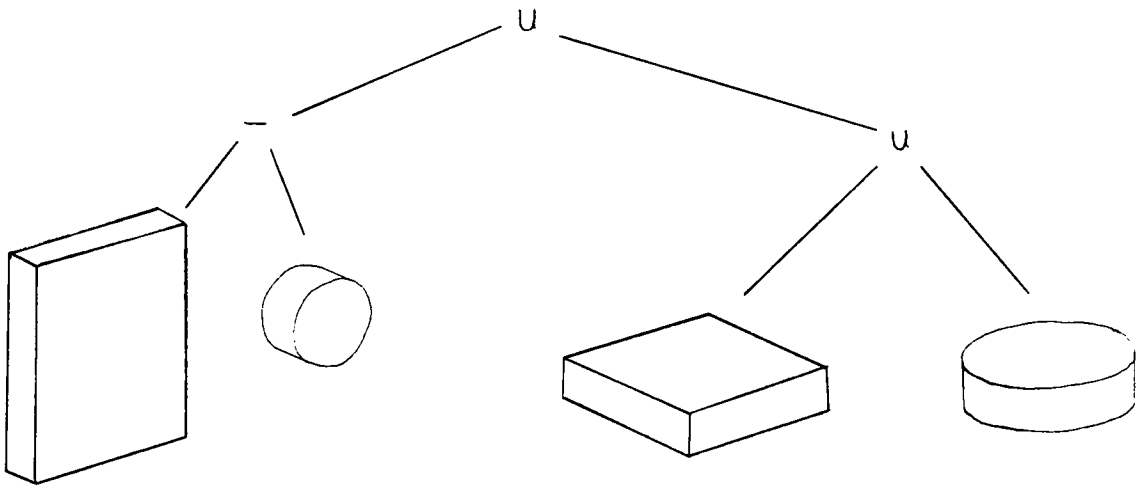



Figure 4.4

Entity Name: Solid

| *Solid-Id* | Solid-Type |
|---|---|
| 1 | P |
| 2 | P |
| 3 | P |
| 4 | P |
| 5 | C |
| 6 | C |
| 7 | C |

P = Primitive, C = Compound

Entity Name: Compound-Solid

| *Solid-Id* | Operator | Operand-1 | Operand-2 |
|---|---|---|---|
| 5 | D | 1 | 2 |
| 6 | D | 3 | 4 |
| 7 | U | 5 | 6 |

I = Intersection, D = Difference, U = Union

Entity Name: Primitive

| *Primitive-Id* | Primitive-Type | 4x4 Matrix |
|---|---|---|
| 1 | BL | |
| 2 | CY | |
| 3 | BL | |
| 4 | CY | |

BL = Block, CY = Cylinder

Entity Name: Block

| *Prim-Id* | Width | Length | Height |
|---|---|---|---|
| 1 | 6 | 1 | 4 |
| 3 | 1 | 5 | 4 |

Entity Name: Cylinder

| *Prim-Id* | Height | Radius |
|---|---|---|
| 2 | 1 | 2 |
| 4 | 1 | 5 |

## 5. Database Implementation Analysis

In this chapter the goals of an engineering database (as discussed in section 2.3.1) are used to measure and evaluate each database model's ability to represent BR and CSG solid models. The goals are listed in alphabetical order. In each section the ability of each database model to support BR and CSG solid modeling representations is discussed and compared with the other database models. A general comparison of the three database models is found in section 2.3.2.

The Relational model has an advantage over the Hierarchical and Network models in this thesis because the E-R diagram technique used in Chapter 4 produces a normalized list of entities and specification of the relationships between the entities. This collection of entities can be implemented without modification in a relational database.

### 5.1 Access Flexibility

Access Flexibility is the ability of a DBMS to allow users to see the data from their own point of view. This includes access via non-key attributes; user friendly, English like access languages for interactive ad hoc querying; interfaces to conventional programming languages; and control of access to the database by the

DBA. This last point will be covered in the section on Security.

It is necessary in solid modeling application to permit different logical views of the data, because of the number of different applications using the solid modeling data and the various subsets of data they utilize. The Relational model provides this ability.

Most relationships are multidirectional. A connection can be made in either direction with the relational join command. The following retrieval shows the retrieval of a list of all edges and vertices (and their coordinates) bounding each edge. They are listed in order the by edges.

```
List-Edges (Solid-Id)
[
retrieve (edge.edge-id, vertex.vertex-id, vertex.x,
vertex.y, vertex.z)
      where (edge.vertex-id1 = vertex.vertex-id
         or edge.vertex-id2 = vertex.vertex-id)
        and (edge.edge-id = fe.edge-id
        and fe.face-id = face.face-id
        and face.surface-id = surface.surface-id
        and surface.solid-id = solid-id)
      sort by edge-id, vertex-id;
]
```

This query can be modified slightly to retrieve either the data for a known edge or a known vertex.

It is not necessary to include a list of all the edges

terminated by each vertex in the database, because the information can be determined with the above join by changing the sort clause to:

```
sort by vertex-id, edge-id
```

In addition complicated multi-tables joins can be specified as 'views' and accessed by end users and programmers as if they were simple tables. This is a distinct advantage of the Relational model.

In the Hierarchical model the users must navigate through the database using the pointers specified when the database was created. Or, programmers will have to write custom routines to navigate through the database for the end users. This is inflexible because it restricts access to the predefined relationships in the hierarchy.

The Network model is more flexible then the Hierarchical model because the database creator can specify multiple paths through the database. The data access process must still specify and follow these pre-determined paths. This is considerable less flexible than the Relational model.

In all DBMS models the data must be arranged to conform to the structure supported by the database. This is generally possible, although some sacrifices must often be made, such as additional redundancy. See section 5.10 for

further discussion of redundancy.

Engineering data is often more meaningful when groups of related data items or records can be accessed as a unit. For example, in the BR database, it would be convenient and useful to manipulate an edge or a face as a unit. Abstract data types could provide this ability. Currently this feature is not available in commercial DBMSs.

## 5.2 Administration and Control

The responsibility for this function is generally given to a single person, known as the Data Base Administrator (DBA). It includes: design of the database; data element names and descriptions (data-dictionary); and performance tunning for overall efficiency; and database security.

In the Hierarchical model these tasks are generally done by the DBA when the database is created. It is necessary that these tasks be well coordinated because changes or errors will affect existing software. Security is discussed in section 5.11.

Administration and control in the Network model are generally the same as the Hiearchical model. The DBA in both cases often must keep extensive documentation about access methods.

In the Relational model the concept of tables is easily understood, and end users can create their own simple databases and perform the function of a DBA with relative ease. Many of the newer Relational DBMSs on the market in the 80's have menu driven 'screen-type' interfaces to perform these tasks. One danger is that multiple people creating tables in the database could create overlapping or duplicate tables. For this reason some central coordination is advised. Only the DBA should be allowed to destroy or modify tables, as many users may use them.

## 5.3 Concurrency

Concurrency is also known as data sharability. It is the ability of multiple users of the database to access data without being aware of other's activity in the database. The multiple users may be running the same or different applications.

Relational DBMSs provide the ability to specify multiple, alternate indexes on a table, allowing the various applications to utilize the index most appropriate for them. This provides quicker access, and thereby reduces the likelihood of contention for resources.

In a Relational DBMS if low level locking (such as record

level locking) is provided, concurrent solid modeling activities should be able to run with little contention. There is however increased overhead involved in managing low level locks. Page and table level locking require less overhead, but are less satisfactory for highly concurrent usage, escecially if the pages or tables stay locked for any length of time.

In Hierarchical and Network DBMSs locking is generally implemented at a higher level. This high level locking is not suitable for a solid modeling database due to the amount of data accessed, and the length of time it takes to complete a transaction. A high level locking scheme could quickly degrade performance to unacceptable levels.

## 5.4 Data Independence

Data Independence is a crucial quality in databases. There are two parts to Data Independence: the applications are insulated from changes to the logical or physical organization of the data; and the application programs cannot affect the data structures.

In both the Hierarchical and Network models, a change to the database structure could cause the pointers connecting the entities to change. This would make any programs using these pointers inoperable.

The Relational model provides greater data independence than the other database models. New tables can be created at any time, allowing phased development, adding enhancements or "fixes" with no disruption of existing software. In a Hierarchical or Network model this is not possible because the pointers to other entities are embedded in each entity.

In a Relational DBMS reclaiming unused space is a relatively straight forward operation. This is because the physical location of an entity is not used as a pointer by other entities, as is done in Hierarchical and Network DBMSs.

### 5.5 Ease of Use for End Users

The majority of those using a solid modeling system will use the applications developed by programmers. However, as business applications have found, there is often a need for ad hoc user quiries. One specific example is troubleshooting corrupted solid models. It is often necessary to know the database structure and correct corrupt data values.

User friendly tools are available for many Relational DBMSs. This is possible since navigation with pointers is

not necessary, as it is in Hierarchical and Network DBMSs. Most Hierarchical and Network DBMSs to not support end-user interfaces for ad hoc queries.

Many of the points discussed in the section on Access Flexibility directly affect the user friendliness of a DBMS.

In many DBMSs, regardless of the underlying model, there is no facility for making a copy of an existing entity. Several circumstances where this would be usful include: 1) making a copy to make a revision, 2) moving an entity between public and private libraries, and 3) creating a new entry slightly different from an existing entity (the copy command in BR) [SIDL80]. In many DBMSs making a copy can only be accomplished by down loading a "copy" to an external file, and then up loading the file to a new location in the database.

## 5.6 Ease of Use for Software Development

The main user community for a solid modeling database is the programming staff who are developing the solid modeling system and its related applications. The ease of use by software developers is affected by the power and flexibility of the tools provided to access the database, and the complexity of the tasks which must be handled by

the programmer.

The navigation with, and maintenance of pointers in Hierarchical and Network DBMSs is error prone and generally unfriendly. This complicates the development process and makes it more difficult.

The software development tools available on some commercial Relational DBMSs make the productivity of software development much greater than that possible on Hierarchical and Network systems.

The amount of code necessary to accomplish a task is affected by the language used and the level of detail which must be controlled by the software developer. Some Relational DBMSs provide "Fourth Generation" languages which are relatively simple to learn, and very productive to use. They make it possible for some end users to develop their own simple applications.

The languages supported by many Relational DBMSs are more numerous than those supported by Hierarchical and Network DBMSs: often only COBOL is supported. The data manipulation languages currently used do not suit the engineering and solid modeling environment, or provide the type of access necessary to complete the tasks required in a solid modeling system.

Full support of true floating point (REAL), fixed point (INTEGER), and bit (LOGICAL) data types must be supported for engineering applications such as:solid modeling. This support should include all standard operations for these data types, as well as efficient storage of the data.

Most Relational DBMSs support these standard data:types. The Hierarchical and Network models were created with business usage in mind, and sometimes do not provide these critical data types.

There are several different query languages used in Relational DBMSs. The selection of a standard query language would be advantageous to the entire database user community. Currently SQL is being considered for acceptance as an ANSI standard. Unfortunately SQL does not support engineering functionality adequately.

Abstract data types, as mentioned in section 5.1 on Access Flexibilty, enable the software developer to manipulate a collection of related entities is if they were a single entity, greatly improving productivity and reliability. This would be an extremly powerful tool in a solid modeling database.

## 5.7 Integrity

Integrity ensures coordination of access by multiple concurrent users, propagation of updates, and validity checks on input data.

A self-verifying representation is more reliable than a user validated model. It is desirable to have the DBMS do as much validation as possible, so that the user and the software don't have to do it.

Both the BR and CSG data models are complex. In a Hierarchical or Network database some constaints are imposed by the nature of the relationships between entities. For example, child node cannot be added unless their parent exists. This improves the integrity of such systems.

Referential intergrity is a concept used in Relational databases to accomplish the restraints built into Network and Hierarchical DBMSs. It stipulates that if table B depends on table A, then a tuple must exist in A corresponding to the tuple to be added to table B, before the tuple can be added to table B. Referential integrity must be explicitly specified in a relational database. Referential integrity is necessary for both the BR and CSG databases because, for example, a vertex cannot be used in an edge before the vertex is created.

Here the Relational DBMS is at a disadvantage when compared to a Hierarchical or Network database, because even if referential integrity is available it involves a retrieve to an additional table to determine whether or not the tuple referred to exists.

## 5.8 Performance

The database performance measure that a user oriented system must be concerned with is the length of time it takes the database to execute a command and respond to the user. The users of a solid modeling system will generally expect a maximum of four second response time on simple calculations or manipulations, especially those that are done frequently. For operations that are done infrequently, a user may be willing to wait a little longer for a task to complete.

Generally, the performance of a Relational DBMS is somewhat less than the other database models. This is an important consideration because interactive users expect prompt response from a computer. However, as advances are made in hardware (for both storage and processing) the difference in performance between Relation and the other models will lessen. The Relational DBMS is a relatively young technology, and many vendors announce a 40 to 50 percent performance

increase every six to twelve months. This is due to improved implementation.

The performance of a DBMS can be affected by various attributes of a DBMS and tuning done to optimize them.

Since access to the data may use non-key attributes for the sake of user friendliness, it is often necessas to provie secondary indexes (or directories) to the data to maintain adequate performance. This capability is frequently provided on a global level by a DBMS. However, since the data (in the case of both BR and CSG) shows hierarchial tendencies, it would yield better performance if the indexing scheme could take the values of the data at each level into consideration. For example, it would scan only the vertices in the edges bounding the faces in the solid in question, instead of all the vertices in the database. This can be simulated with concatenated keys from each level in the hierarchy, or by using other structures such as linked lists. These methods require more time and storage than a localized directory scheme [SIDL80].

The performance of Hierarchical and Network DBMSs is generally quite good. This is due in part to the system of pointers that connect the entities. Another contributing factor is the maturity of these database

models when compared to the Relational database technology.

## 5.9 Recovery

Recovery is the process of returning the database to a correct state after an error has occurred. An error may be caused by software, hardware, or users, for example: recovery from a crash of the disk containing the database; a computer system crash during a transaction; corruption of data by users, software or hardware; as well as backing out transactions before completion in the case of deadlock.

Recovery is extremly important in a solid modeling database due to the complexity of the data and the difficulty of recovering manually.

Check-pointing and journaling provide a means of recovering to a previous (correct and consistent) state. Checkpointing is done by making a complete copy of the database (a snap shot) on a secondary device (other than the one containing the database). Journaling involves recording all changes (transactions) made to the database since the last check-point. These are placed in a journal file, also on a different secondary storage device. These recovery techniques are generally

supported by DBMSs of all three database models.

Another form of recovery involves error detection routines provided by the DBMS and by the DBA to prevent the database from becomming incorrect.

A problem arises because engineering transactions differ from business transactions. An engineering transaction is generally much longer, and often spans an entire editing session. The traditional journaling generates a significant amount of overhead and is probably unnecessary. In some circumstances it would be sufficient to recover to the previous version, saved at the beginning of the editing session. This should be a user definable option because it affects performance.

Recovery of a Relational database will be simpler then a Network or Hierarchical database because the data is not dependent on pointers and disk location in a Relation database.

## 5.10 Redundancy

Redundancy is duplicated data. Reduced redundancy is desirable to reduce the amount of storage space consumed by the data.

Since the data in any DBMS model is accessable to many users and applications simultaneously, it is only necessary to store the data once. This reduces the amount of storage required and eliminates the problem of keeping multiple copies consistent when compared to non-DBMS solutions.

In Relational databases each column in a table which is used to join another table is redundant. A relational database uses redundant data instead of pointers to provide a link from one table to another. The BR entity-relation diagram necessitates a large amount of redundant data. In contrast, the Network and Hierarchical models use pointers where the relational model uses redundant data.

The Hierarchical database model supports one to many relationships. It does not support many to many relationships. In the BR E-R diagram there are several many to many relationships, for example: 1) faces : edges, 2) edges : vertices. The required relationships could be implemented by duplication of some data. However, this would introduce some undesirable redundancy.

It is necessary to navigate both up and down the hierarchy. Since this is not supported by the

Hierarchical model an edge list must be added to the Vertex entity, thus adding redundant information that the software must maintain. This is undesirable, and shows the dependency of the software on the database's physical organization.

The Network model is better suited than the Hierarchical model to handeling many to many:relationships.

In the Relational model it is handled by:creating a separator entity, such as the Face-Edge entity in figure 4.1.

## 5.11 Security

The four types of access to a database are retrieve, append, update and delete. A DBMS must supply a mechanism to control all types of access. The goal is to provide each user with enough privilege to access the data he needs, but no more, and to protect against any unauthorized access. The mechanism should be simple to use and powerful.

The simplest method is by the type of data or entity. For example: user X is permitted to update the project-schedule table. This is known as value-independent control and is useful in some

situations, but too general for others. The second method, value-dependent control, provides security based on the data's value. For example: User X is permitted to update the project-schedule table only for project P3. This type of security is more difficult to implement but is supported by several Relational DBMSs such as System R and Ingres. This is the flexible type of security mechanism desirable for a solid modeling system. For example, allow user X to update solid 3 (and all entities related to it such as surfaces, faces, edges, and vertices).

Security contraints can be controlled on a per application basis, by the use of database statements within the application. This enables the fine tuning of security to the level appropriate in each situation within each application.

In the Relational model security may be specified on the table level or at the attribute level. In addition, some Relational DBMSs, such as Ingres, allow stipulation of terminal, time of day, specific days, as well as specific database operations.

In Hierarchical and Network DBMSs security is often limited to value-independent control. This is less useful.

Table 5.1 summarizes the findings of the chapter.

## Summary of Database Model Analysis

| Goal | Hierarchical | Network | Relational |
|---|---|---|---|
| Access Flexibility | Very poor, predefined pointers | Poor, predefined pointers | Good, dynamic relationship access |
| Administation & Control | Complicated, done up front by DBA | Complicated, done up front by DBA | Simple, by DBA and others |
| Concurrency | Fair | Fair | Good |
| Data Independence | Poor because of pointers | Poor because of pointers | Good |
| Ease of Use for End Users | Poor | Poor | Good, many tools |
| Ease of Use for Software Development | Poor | Poor | Good, many tools |
| Integrity | Good | Good | Fair |
| Performance | Good, well optimized | Good, well optimized | Improving with maturity |
| Recovery | Fair | Fair | Good |
| Redundancy | Fair | Fair | Fair |
| Security | Fair | Fair | Good, more flexible |

Figure 5.1

## 6. Conclusions

Chapter four analysed the data storage requirements of BR and CSG representations of solids. Chapter five evaluated the ability of the three classical database models to support the solid model representations. This chapter is divided into two parts: a summary and evaluation of the results, and a presentation of outstanding questions, problems and future directions of solid modeling.

### 6.1 Evaluation of Results

It is concluded that Relational DBMSs are useful for storing representations of solid models. DBMSs were created to serve the needs of business users. Until they are adapted to suit the needs of engineering applications, they cannot be expected to be totally suited to this usage. However, unless they are used in engineering applications such as solid modeling, the weak points cannot be identifed and addressed. The strong points of the Relational model are outlined below.

#### Access Flexibility

The Relational model is very flexible in the way users may access the data. This is due to the fact that redundant data (and not pointers) is used to make connections between entities in the database.

Adminstraton and Control

The tasks of administration and control are simpler and done in a more flexible manner when compared to the Hierarchical and Network models.

Concurrency

Highly concurrent usage is possible with a Relational DBMS due to low level locking mechanisms and distribution of data between entities.

Data Independence

Because the entities in a Relational DBMS are not linked via pointers, they are more insulated from addition of, or changes to entities.

Ease of Use for End Users

Userfriendly ad hoc quiry facilities are available, and the already simple Relational model may be further simplified by the use of views, providing access to only what is necessary.

Ease of Use for Software Development

The freedom from pointers, and the availablitity of high level fourth generation languages make software development easier and less error prone in a Relational DBMS when compared to Hierarchical and Network models.

Recovery

Recovery is generally simpler in a Relational DBMS than it is a Hierarchical or Network DBMSs.

Security

Many Relational DBMSs provide simpler and more complete control of security than the other database models.

Although the Relational model was judged the most suitable DBMS model, several short comings of the Relational model are identified in this thesis.

Integrity

Integrity contraints such as referential integrity require additional overhead when compared to implementation with Hierarchical and Network models.

Performance

The performance of Relational DBMSs is somewhat slower then the performance of Hierarchical and Network models. Performance of Relational DBMSs will improve as implementation of the Relational model is further perfected, and hardware speeds increase.

Redundancy

Redundant data is used in place of the pointers used in the Hierarchical and Network models. The amount of

redundancy in implementing a solid modeling system with a Relational DBMS is however, not significantly greater than the redundancy required by the inflexibility of the other models.

The Network model is also suitable for storing representations of solid models, but to a lesser degree than the Relational model. This is because of inflexibility in access methods, and lack of user-friendliness, both the result of using a system of pointers to navigate through the database.

The Hierarchical model is the least suitable of the three classical database models for storing representations of solid models. The weaknesses of the Hierarical DBMS model are the same as those of the Network model, but to a greater degree.

The functionality evaluated by this thesis is limited. Ideally, the functionality supported by a solid modeling system should be expanded to include that discussed at the start of chapter 4 as well as the needs of all applications related to mechanical design, as well as greatly enhanced user friendliness to aid in the design of solid models.

It is also concluded that any database model is better

than using a system of user-controlled files. There are several reasons for this: (1) increased security provided by a DBMS; (2) reduced redundancy and improved control of versions and sharing of data; (3) automated backup and recovery procedures; (4) improved performance; and (5) the ability of the user to interact with the data and leave the storage details to the DBMS.

## 6.2 Future Directions/Research Directions

Solid modeling is a young field, and will be experiencing much activity and growth during the next several years. The following sections discuss several key areas.

### 6.2.1 Graphics and Solid Modeling

A major problem with current commercial solid modeling systems is that they do not represent the interior of the object, or the internal properties and behavior of the solid. Generally, total internal homogeneity is assumed. The ability to *analyze* the model distinquishes a computer-aided design system from a computerized drawing system [PRIN71]. Ue more accurate and complete the model, the better the analysis will be. The lack of internal representation limits the functionality of a solid modeling system.

Current research seems to indicate a hybrid of multiple solid representations may be required to fully support solid modeling. As shown in figure 3.1, many of the current systems use both CSG and BR. We believe that this trend of using multiple representations will continue unless a representation that better suits the needs of all applications related to solid modeling is created. There has been, and will continue to be, a tremendous amount of research in solid modeling [WAGN84].

### 6.2.2 Databases Management Systems

In recent years almost every part of the engineering and manufacturing processes has been computerized. It is clear that the islands of automation must be combined and coordinated into a single (possibly distributed) database to eliminate the time spent converting data from the format of one island to another. This is a task of giant proportion, critical importance, and great difficulty.

The two parts of this task are: (1) To find the best possible methods of storing solid modeling data; and (2) To store the representation in a manner that will enable company wide access when necessary, and local autonomy and control whenever possible.

Especially in large companies, the use of distributed

databases is critical to achieving these goals. It would provide a means of automating the entire design-manufacturing cycle, increase integration, interfacing various DBMSs and extensive sharing of data. This will greatly reduce the costs now incurred due to redundant and repeated efforts, tasks and data.

One current trend is toward high powered, lowcost, single user systems. The first interactive graphics systems were single user systems. This trend may have adverse affects on the trend to integrate databases and communicate between the various phases of the design and manufacucturing process, unless these single user systems are coordinated with a central database.

To satisfy the database needs of the engneering community the following concerns require attention: (1) increased speed; (2) facility to handle engineering transactions; (3) high(er) level facilities to aid in engineering functionality (such as managing product models and engineering data); and (4) the ability to use non-static data structures.

### 6.2.3 Mechanical Engineering

Solid modeling systems in the future must serve both the design engineer and the manufacturing engineer. The

traditional communication gap between these two groups will be solved technically. This will require planning and cooperation by both parties to put new technology to work.

Computer Integrated Manufacturing (CIM) must be put into place by manufacturing concerns that wish to remain competitive. Such automation improves productivity and reduces costs. It is a method of competing with inexpensive foreign labor, and those who do not automate will not be able to compete with those who do so.

Several improvements are needed by the mechanical design community. They include: (1) device independent graphics system standards; (2) spatially oriented database query interfaces; (3) software tools for dialogue generation; and (4) increased realism in computer graphics.

Strategies being developed to implement CIM by the CAM-I Geometric Modeling Program are using a "solid geometric model" as the cornerstone to their approach [CAMI86]. The goal is to provide a generally acceptable product model for use by all related applications through out the engineering community.

## Appendix A: Glossary of Terms and Acronyms

U & - Boolean operators: Union, Intersection, Difference

AI Application Interface being developed by CAM-I for solid modeling transfer of data

ANSI American National Standard Institute: Part of ISO

APT Automatically Programmed Tool

ASM Analytic Solid Modeling

BCNF Boyce/Codd Normal form - third normal form rule for normalizing relational database tables

boolean operators
Union: U, Intersection: &, and Difference: -.

boundary merging
Also called boundary evaluation or object evaluation. Combining the boundaries of two objects to create a single boundary, usually the result of a boolean operation.

BR Boundary Representation of a solid, (also B-Reps)

CAD Computer Aided Design/Computer Aided Drafting

CAE Computer Aided Engineering

CAM Computer Aided Manufacturing

CAM-I Computer Aided Manufactureing - International

checkpoint A snap shot of a database used to restore a corrupt database to the correct state captured in the checkpoint.

CIDF Common Interchange Description Format

CIM Computer Integrated Manufacture

complex objects
An object composed of a multiple of simpler objects, e.g., a polygon is a collection of lines.

convex element
: An enclosed shape where a straight line, from any internal point to any other internal point, does not pass outside the boundary of the object.

CORE
: Core Graphics System: standard proposed by SIGGRAPH in 1977 & 1979

CSG
: Constructive Solid Geometry: represents solids as combinations of primitives

DBMS
: Data Base Management System

DPU
: Display Processing Unit - produces graphics output on screen or printer

Euclidean Space
: Three dimensional space with x, y, z coordinates, sometimes refered to as E3.

FEM
: Finite Element Modeling/Mesh Analysis provides structural or thermal calculations

GKS
: Standard Graphical Kernel System

halfspace
: Shapes that divide 3D space in half, i.e., the space inside the shape and the space outside, or on the wrong side of the shape.

IBE
: Incremental Boundary Evaluator, a PADL-2 utility to convert CSG to BR

image space
: The 2D image of a solid seen in the screen by the user and how the image is created or calculated from the 3D model.

ISO
: International Standard Organization

journal
: A record (audit trail) of every transaction against a database from the time of the last checkpoint.

loft lines
: 2D section curves and longitudinal curves placed in a mesh configuration to create a sculptured surface[FAUX79]. This technique is used to generate automobile bodies and ship hulls.

mass properties
: Enclosed volume, principal axis, moments

around each axis, mass, etc...

mesh — A series of smaller objects used to approximate a larger, usually more complex object.

meta file — A file storing commands to re-create the picture.

moments of inertia — Rotational inertia of a body with respect to the particular axis of rotation.

NBS — National Bureau of Standards

object space — The 3D space in which the object exists.

origin — The reference point in design space located at (0,0,0).

OSI — Open Systems Interconnection

PDES — Product Data Exchange Standard

planar surface — A (flat) 2D surface.

polyhedron — A volume completely enclosed by polygons

RAND tablet — A graphics interaction device first used in 1964.[DUCE83]

ray-tracing (ray-firing) — A technique used to remove hidden lines or surfaces from 3D display.

rectilinear polyhedra — A volume bounded by straight lines

relatability — The ability to define relationships between records or entities in a database at the logical level.

rigid motion — A mapping of E3 into E3, preserving distances between pairs of points and signed angles between ordered triples of points[CHEC79].

ring data structure — A data structure which is circular, and following the pointers eventually leads back to the starting point.

SAGE — Stratigic Air defense system

scan-line (scan-plane) A line or plane which passes across an object, used to determin relative position of parts of a solid. Similar to Ray-tracing.

sculptured surface or part
A surface or object which cannot be represented by any combination of quadratic halfspaces, toriodal halfspaces or process define geometry.

SGE — Simple Geometric Entity

SIGGRAPH — ACM Special Interest Group: Graphics

solid model
A model of a solid object which can be used to answer questions about the real object.

STEP — Standard for the Exchange of Production data

surface — Generally, a sculptured or curved 3D face of a solid object

tablet — A flat surface and a pointing mechanism (stylus or hand cursor) used as an interactive graphics input device

topology — The sturcture of a part or shape, and how the parts are connected.

tori
toroid — Surface shaped like a doughnut.

vector display
Each line is drawn from end to end, faster interactive response than raster display

voxel — A method of diviiding solid objects into smaller and smaller parts until the desired resolution is achieved.

wireframe — A 2D or 3D representation of an object using the edges and vertices. The graphics display looks like a wireframe model of the object.

XBF-2 — Experimental Boundary File - 2 proposed by CAM-I

## Appendix B: BIBLIOGRAPHY

[BARO 81] Baron, N., Bornkessel E., Cullman, N., Klos, W., Magalhaes, L., "An Approach to the Integration of Geometrical Capabilities into a Data Base for CAD applications." in File Structures and Databases for CAD, North-Holland Publishing Co., N.Y., 1981, pp231-243.

[BIRD 69] Bird, S. "Computer Graphics Software Techniques." in Computer Graphics Techniques and Applications, Plenum Press, New York, 1969, pp. 17-28.

[BOWY 83] Bowyer, Adrian, Woodwark, John., "A Programmer's Geometry," Butterworths, Boston, MA, 1983.

[BRAI 75a] Braid, I. C., "Six Systems for Shape Design and Representation - A Review." C.A.D. Group Document No. 87, University of Cambridge, May 1975.

[BRAI 75b] Braid, I. C., "The Synthesis of Solids Bounded by Many Faces." Communications of the ACM, Vol. 18, No. 4, April 1975, pp. 209-216.

[BRIS 97] Bristol, W. A., and Wong, S.,, "A Computer Aided Design Data Base." Proceedings of 16 th Design Automation Conference, 1979, pp 398-402.

[BROD 84] Brodie, Michael, Mylopoulos, John, Schmidt, Joachin, Editors, "On Conceptual Modelling: Perspectives from Artifical Intelligence, Databases, and Programming Languages." Springer-Verlag, New York, 1984.

[BROW 81] Brown, Christopher, "Some Mathematical and Representational Aspects of Solid Modeling." IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-3, No. 4, July:1981, pp. 444-453.

[BROW 82] Brown, Christopher M. "PADL-2: A Technical Summary." IEEE Computer Graphics and:Applications, Los Alamos, CA 90720, 1982.

[BYLE 85] Byles, Torrey, "Selective Update: Fifth Generation CAD/CAM Systems Soon to be Born." by Patrick J. Hanratty, IEEE Computer Graphics and Applications, March 1985, pp. 80-81.

[CAMI 86] CAM-I, Computer Aided Manufacturing - International, "Geometric Modeling Program 1986", Prospectus, 1986.

[CARD 85] Cardenas, Alfonso F., "Data Base Management Systems." 2nd ed. Allyn and Bacon, Inc., Boston, 1985.

[CASA 85] Casale, Malcolmm S., Stanton, Edward L., "An Overview of Analytic Solid Modeling." IEEE Computer Graphics and Applications, February 1985, pp. 45-56.

[CHEC 79] Check, T. F., Hartquist, E. E., Mailler A. H., Tilove, R. B., "CGGM-8: RMPAK: A Set of Fortran Subroutines for Manipulating Rigid Motions." Production Automation Project, University of Rochester, N. Y., 1979.

[CHEC 85] Check, T. F., Dodsworth, J. R., Hartquist, E. E., Tilove, R. B., "CGGM-12: Representation in the PADL-2/N Processor: Low LEvel Entities." Production Automation Project, University of Rochester, NY, 1985.

[CHU 83a] Chu, K., Fishburn, J. P., Honeyman, P., Lien, Y. E., "VDD: A VLSI Design Database System." Proceedings ACM/IEEE Database Week, San Jose. CA, May 1983, pp. 25-37.

[CHU 83b] Chu, K., and Lien, Y. E., "Two Issues in VLSI Design Systems: Technology Independence and Data Management." IEEE ICCAD 1983, pp. 109-110.

[C/CA 85] CAD/CIM Alert, Vol. V, No. 8, Decemeber 31, 1985.

[C/CA 86a] CAD/CIM Alert, "Announcing a two-day International Conference on CAD/CAM Databases '86: Control for the Decades Ahead." Promotional material.

[C/CA 86b] CAD/CIM Alert, "CAD/CIM/CAE Product News Suppliment." Vol. 1, No. 1, March/April 1986.

[DACO 86] D. Appleton Company, Inc., "Data Resource Techniques Workshop: Participant Workbook." D. Appleton Company, Inc., Manhattan, CA, 90266, February 1986.

[DUCE 83] Duce, D. A., Gallop, J. R., Hopgood, F.R. A., Sutcliffe, D. C., "Introduction to the Graphical Kernel System (GKS)." Academic Press, New York, NY, 1983.

[EBER 82] Eberlein, W., Wedekind, H., "A Methodology for Embedding Design Databases into Integrated Engineering Systems." in File Structures and Databases for CAD, North-Holland Publishing Co., N.Y., 1981, pp. 3-43.

[EDIF 85] EDIF Steering Committee, "EDIF Electronic Design Interchange Format Version 1.0." 1985.

[ENCA 80] Encarnacao, J., et. al., "The Workstation Concept of GKS and the Resulting Conceptual Defferences to the GSPC Core System." SIGGRAPH '80 Proceedings, Published as Computer Graphics, 14(3), July 1980, pp. 226-230.

[ENCA 81] Encaracao, J., and Krause, F. L. editors, "File Structures and Databases for CAD." North-Holland Publishing Company, N.Y., 1981.

[EVAN 69] Evans, D. R., "Computer Graphics Hardware Techniques," in Computer Graphics Techniques and Applications, Plenum Press, New York, 1969, pp. 7-16.

[EVER 86] Everest, Gordon C., "Database Management." McGraw-Hill Book Company, N. Y., 1986.

[FAUX 79] Faux, I. D., Pratt, M. J., "Computational Geometry for Design and Manufacture." John Wiley & Sons, New York, 1979.

[FISC 79] Fischer, W. E., "PHIDAS - A Database Management System for CAD/CAM Application Software." Computer Aided Design, Vol. 11, No. 3, May 1979, pp. 146-150.

[FOLE 83] Foley, James D., and Andries Van Dam "Fundamentals of Interactive Computer Graphics." 2nd ed. Addison-Wesley Publishing Company, Menlo Park, 1983.

[FU 77] Fu, K. S., Klinger, A., Kunii, T. L., Editors, "Data Structures, Computer Graphics, and Pattern Recognition." Academic Press, Inc., N.Y., 1977.

[FU 83] Fu, K. S., Lee, Y. C., "A CSG Dased DBMS for CAD/CAM and its Supporting Query Language." Proceedings of the Engineering Design Applications of ACM-IEEE Database Week, San Jose, CA, May 1983, pp. 123-130.

[GRAB 82] Grabowski, H., Anderl, R., Rausch, W., Seiler, W., "DICAD - A CAD System for Geometric Product Modeling." in CAD Systems Framework, North-Holland Publisging Company, N.Y., 1982.

[HANR 84] Hanratty, Patrick J., "Solid Modeling Prospects." Systems International, Vol. 12, No. 10, October 1984, pp. 46-47.

[HARD 84] Hardwick, Martin, "Extending the Relational Database Data Model for Design Applications." IEEE 21st Design Automation Conference, 1984, paper 8.2.

[HART 85a] Hartquist, E. E., Peterson, D. P., Voelker, H. B., "CGGM-20 BFILE/2: A Boundary File for PADL-2." Production Automation Project, College of Engineering and Applied Science, U of R, Rochester, N.Y., 14620.

[HART 85b] Hartquist, E. E., "IBE-4: pp2/2.n Boundary Evaluation."

[HECK 86] Heck, Mike, "PHIGS Hits the Market." Computer Graphics World, January 1986, pp. 49-53.

[ISO 81] International Standards Organization "Graphical Kernel System (GKS), Version 6.6." May 1981.

[JOHN 84] Johnson, Robert H., "Solid Modeling: A State of the Art Report featuring an Evaluation of 21 Commercial Systems." CAD/CAM ALERT, Chestnut Hill, MA. 1984 or 85.

[JONE 86] Jones, Keith, "International Group Prepares Worldwide CAD/CAM Standard." Mini/Micro Systems, January 1986, pp. 44.

[KELL 81] Keller, K., "KIC, A Graphics Editor for Integrated Circuits." Masters Thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, June 1981.

[KITA 84] Kitagawa, H.; T. Kunii; M. Azuma; S. Misaki, "Formgrapphics: A Form-Based Graphics Architecture Providing a Database Workbench." IEEE Computer Graphics & Applications, June 1984, pp. 38-56.

[KORE 75] Korenjak, A. J., and Teger, A. H., "An Integrated CAD Database System." Proceedings of 12th Design Automation Conference, 1975, pp. 399-406.

[LABU 85] LaBuda, Virgil, and Michael Waters, "Interchange Format Solves Problems of Design Transfer." Computer Design, September 15, 1985, pp. 103-110.

[LIEW 85] Liewald, Michaell H., "Initial Graphics Exchange Specification: Successes and Evolution." Computers and Graphics, Vol. 9, No. 1, 1985, pp. 47-50.

[LORI 81] Lorie, Raymond A., "Issues in Database for Design Applications." in File Structures and Databases for CAD, North-Holland Publishing Co., N.Y., 1981, pp. 213-229.

[MACH 69] Machover, C., "Computer Graphics in the United States." in Computer Graphics Techniques and Applications, Plenum Press, 1969, New York, pp. 61-84.

[MACN 84] Machover, C., "Silicon Support for Solid Modeling."

[MARI 86] Marisa, R. J., "SGM-41: CSG Data Structure Management." Systems Group Memo 41, Production Automation Project, University of Rochester, N. Y., 1986.

[MART 84] Martin, James, "3.3 Selection Criteria for 4GLs." The Martin Report on High Productivity Languages, Vol. 2, Section 3.3, Highproductivity Software Inc., Marplehead, MA, 1984, pp. 1-26.

[MASA 74] Masakazu, Soga, et. al., "Engineering Data Management System (EDMS) for Computer Aided Design of Digital Computers." Proceedings of 11th Design Automation Conference, 1974, pp372-379.

[MORT 85] Mortenson, Michael E., "Geometric Modeling." John Wiley & Sons, New York, NY, 1985.

[NEWM 79] Newman, William M., Sproull, Robert F., "Principles of Interactive Computer Graphics." McGraw-Hill Book Company, New York, 1979.

[NEWM 81] Newman, M., "The CADIF Syntax." proprietary specification, Digital Equipment Corp., 1981.

[NIER 83] Nierenberg, Nicolas C., "Unix-based Data Base Fits 16-bit Systems." Electronics, August 11, 1983.

[O'CO 85] O'Connell, L., Personal correspondence.

[PATN 82] Patnaik, L. M., and N. Ramesh, "Implementation of an Interactive Relational Graphics Database." Computers and Graphics, Vol. 6, No. 3, 1982, pp. 93-96.

[PATN 83] Patnaik, L. M., and Shenoy, R. S., "A Relational Database for Computer AIded Engineering." 1983 IEEE International Conference on Computer-Aided Design, 1983, pp. 107-108.

[PRIN 71] Prince, David, "Interactive Graphics for Computer Aided Design." Addison-Wesley Publishing Company, Reading, MA, 1971.

[RELA 86] Relational Technology Inc., "INGRES/QUEL Reference Manual.", Version 4.0, Vax/VMS, January 1986.

[REQU 77] Requicha, A. A. G., Voelcker, H. B., "Geometric Modeling of Mechanical Parts and Processes." Computer, Vol. 10, No. 12, December 1977, pp. 48-57.

[REQU 80] Requicha, A. A. G., Tilove, R. B., "Closure of Boollean Operations on Geometric Entities." Conputer Aided Design, Vol. 12, No. 5, September 1980, pp. 219-220.

[REQU 82] Requicha, A. A. G., Voelcker, H. B., "Solid Modeling: A Historical Summary and Comtemporary Assessment." IEEE Computer Graphics Applications, Vol. 2, No. 2, March 1982, pp. 9-24.

[REQU 84] Requicha, A. A. G., "TM-28: Mathematical Models of Rigid Solid Objects." Tech Memo 28, Production Automation Project, University of Rochester, NY, 1984.

[REQU 85] Requicha, A. A. G., Voelcker, H. B., "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Alogirithms." Proceedings of the IEEE, vol. 73, No. 1, January 1985, pp. 30-44.

[ROBE 63] Roberts, L. G., "Graphical Communication and Control Languages." Second Congress on the Information System Sciences, pp. 101-107.

[RYAN 85] Ryan, Daniel L, "Computer-Aided Graphics and Design." 2nd ed. Marcel Dekker, Inc, NY, 1985.

[SHEP 85] Shepard, Mark S., Spooner, David L., Wozny, Micheal J., "Abstract Data Types for CAD Systems." Proceedings of the 1985 International Conference on Robotics and Automation, St Louis, MO, March 25-29 1985, pp. 359-364.

[SHIG 72] Shigley, Joseph, "Mechanical Engineering Design." 2nd ed. McGraw-Hill Book Company, New York, 1972.

[SIDL 80] Sidle, Thomas W., "Weakness of Commercial Data Base Management Systems in Engineering Applications." Proceeding of the 17th Design Automation Conference, Minneapolis, MN, June 23-25 1980, pp. 57-61.

[SMIT 85] Smith, Thomas R., "A Data Architecture for an Uncertin Design and Manufacturing Environment." IEEE 22nd Design Automation Conference, 1985, paper 21.1.

[SPOO 84] Spooner, David L., "Database Support for Interactive Computer Graphics." ACM SIGMOD 1984, pp. 90-99.

[STON 83] Stonebraker, M., et al., "Application of Abstract Data Types and Abstract Indices to CAD Databases." Proceedings of the Engineering Design Applications of ACM-IEEE Database Week, San Jose, CA, May 23-26 1983, pp. 107-115.

[STON 85] Stoneking, Frank L., "GM/IGES Specification." Personal correspondence.

[STON 86] Stonebraker, Michael, Editor "The INGRES Papers: Anatomy of a Relational Database System." Addison-Wesley Publishng Company. Menlo Park, CA, 1986.

[SUTH 63] Sutherland, Ivan E., "SKETCHPAD: A Man-Machine Graphical Communication System." Proceedings Spring Joint Computer Conference, 1963, Spartan Books, New York, pp. 329-345.

[WAGN 84] Wagner, Patrice M., "Solid Modeling for Mechanical Engineering." Computer Graphics World, September 1984, pp. 10-24.

[WANG 84] Wang, R., Zhang, J., "Some Concideration on the Database Model of Geometric Data Bases." ACM/IEEE 21st Design Automation Conference Proceedings '84, June 1984, pp. 629-633.

[WASL 84] Wasley, Ronald G., and Ismet Erkmen, "Interactive Database Management for Power Systems Computer-Aided Instruction." 1984 Frontiers in Education Conference Proceedings, IEEE, 1984.

[WOO 84] Woo, Tony C., "Interfacing Solid Modeling to CAD and CAM: Data Structures and Algorithms for Decpmosing a Solid." Computer, Vol. 17, No. 12, Dec 1984, pp. 44-49.

[WOZN 81] Wozny, Micheal J., "Solid Modeling in CAD/CAM." IFIP WG 5.2 Working Conference in CAD/CAM as a Basis for the Development of Technology in Developing Nations, North-Holland Publishing Company. N.Y., 1981, pp. 45-47.

## Appendix C: About the Author

The author received her Bachelor's degree in Elementary Education, with a minor in mathematics from Concordia College in 1979. After several years as an elementary school teacher, she changed careers and returned to college to take courses in Computer Science.

In 1982 she started the graduate Computer Science program at Rochester Institute of Technology. Initially her interests were in operating systems and compilers. Her current interests are related to databases, and applications which make use of database management systems.

The author's current work assignment involves software development of database applications for an engineering support community at Eastman Kodak.