THE DESIGN AND ANALYSIS OF BOUNDARY DATA STRUCTURES

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL, ELECTRONIC AND

INFORMATION ENGINEERING

AND THE HIGHER DEGREES COMMITTEE

OF CITY UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Seshagiri Rao Ala

June 1992

I hereby grant powers of discretion to the City University Librarian to allow the thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

Seshagiri Rao Ala
(Author)

Acknowledgements

Foremost, I would like to thank my two supervisors: Dr. T.J. Ellis who anointed me into the world of CAD data structures and has been a source of continuous support and Mr. D. A. Chamberlain who nurtured me with his ever friendly approach, financial support, and openness to new ideas. Thanks also to Mr. John Snell who not only contributed in filling the gaps in mathematical knowledge demanded of me by this thesis but also has been a source of personal friendship and encouragement.

This thesis has been profoundly influenced and shaped by the valuable suggestions of the referees of my publications, whose incisive criticism honed my skills of presentation also. A word of special mention to my Ex General Manager Mr.G.S.Kang, whose zeal for higer education encouraged me to undertake the Ph.D., and was also a fantastic source of moral support during the turbuluent times when I was compelled to trade in my hard won job in the Indian Railways for the continuation of the research.

Special thanks to Prof K. T. V. Grattan, Head, Dept. of Electrical Eng., whose persuasion instrumented this thesis. Without his and my research tutor Dr. B. M. A. Rahman's ever willingness to help, it would not have been possible for a timely finish of this thesis. In addition, Dr. B. M. A. Rahman generously helped with computer hardware resources. I also thank Dr.F.Abdullah for the same.

Dr.J.Chandler generosly helped me with software facilities. Thanks also to the Engineering Design Research Centre for providing me with access to their ACIS geometric modeling software.

Several people contributed in making this daunting task of undertaking a comparative study pleasurable. Special thanks to Dr. Ian Braid of Three-Space Ltd., for discussions on the non-manifold data structure requirements and Dr. Jarek Rossignac of IBM for encouragement with the notation. Several authors have spent time talking to me clarifying their ideas.

I acknowledge with gratitude the financial support of the Science and Engineering Research Council without which this work could not continue.

Abstract

The thesis is concerned with the efficient interrogation of CAD data. CAD data finds use in diverse range of applications which necessitates extension and integration of the CAD data base. By an exhaustive categorization of such application requirements and analysis of various CAD techniques, it is shown that boundary data structures are the most suitable in CAD, CAM and advanced robotic applications.

Several boundary data structures have been proposed since the classic Winged edge data structure, these aimed at reducing the storage requirement and increasing information retrieval speeds. In this thesis methodologies are developed which enable us to discover compact and fast access time schemes and analyze and fine tune for individual applications. We demonstrate how the application of the optimality concepts can lead us to the discovery of more efficient data structures than popular data structures. All the boundary data structures proposed to date have been based on the underlying assumption that all the data resides in main memory. We show that in an integrated CAD environment (characterized by virtual a memory environment or a data base environment), these data structures are inefficient in both storage and time. We propose a new data structure shaped like Δ which is the most compact as well as more efficient in access time, under certain conditions of real memory and virtual memory. Experiments reveal a paradoxical phenomenon: access time increases

with storage, violating the classic law of storage vs. time.

Recently non-manifold boundary geometric modeling has become popular to meet the growing needs such as uniform treatment of wire frame, surface and solid modeling and design by features. We introduce a uniform terminology and notation to distinguish and critically analyze several non-manifold boundary data structures. It is hoped to fulfill the need for a ready reference for the design of efficient boundary data structures. The other aspects dealt with are the validity and conversion of Boundary data structures.

To verify the concepts developed, in practice, a whole suite of fast algorithms have been implemented for model manipulation, visualization and data conversion.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 CAD data uses

Computer Aided Design (CAD) data is used in a diverse range of applications related to computer integrated manufacturing (CIM): machine control, automatic inspection, packaging, vehicle guidance etc. Operations on CAD data may be grouped into two classes: manipulative and non-manipulative operations. Manipulative operations involve the modification (creation, deletion etc.) of CAD entities using a variety of techniques. Non-manipulative operations are mainly concerned with data retrieval for display or interrogation purposes. A good example of non-manipulative use is the area of model based computer vision, where appropriate shape features must be accessed from the CAD model and matched against image data.

A variety of alternative data structures have been used for representing the CAD data. These alternative representations attempt to trade-off storage efficiency against access time. Some are tailored to specific applications. An integrated approach is required for structuring the data so that differing application requirements can be

met from a single data base to ensure integrity of data and to avoid redundancy of multiple data bases each tailored to individual applications.

1.2 Aim and Objectives

1.2.1 Aim

The principal aim of our work is the improvement of CAD data structure efficiency. The CAD data base has voluminous data from which we need to retrieve only a small set (e.g. extraction of the direction of all the edges meeting at a given vertex), in real time, to enable the system controllers to take prompt action to avert potential disasters in critical machinery and environment. The hardware is never powerful enough to keep the user happy, as stated by Rosenthal [1989]. Greenberg [1991] concurrs in his predictions for graphics in this decade: "Despite enormous advances in hardware performance, the demand for processing power will always outpace the supply. With modeling complexity increasing at a faster rate than machine performance and the required computational times increasing exponentially, will the situation get worse". In the quest for real time execution of ambitious applications, data structure efficiency studies assume a more prominent role than the hardware speedup. Organization for efficient retrieval can result in the speed of manifold tasks (e.g. collision detection, real time recognition). Hence because of the fundamental nature of the efficiency problem, the efficiency of the CAD data structures forms the central thesis of this work.

1.2.2 Objectives

Unified treatment of boundary data structures

Ever since the discovery of the classic Winged edge data structure [Baumgart 1975], there have been innumerable variations of it proposed. It is one of the objectives of this work to unite the multitude of data structures under a common umbrella and present a comprehensive, yet clear view of the boundary data schemata.

Optimization of data structure storage and access

The thesis also addresses the optimality problem of all the combinatorially possible data structures. For tractability of this objective the acheivement of the previous objective is a must.

Quantitative methods for evaluation

Although there have been innumerable data structures proposed, there are very few tools available for a designer of CAD systems to guide in the selection of appropriate data structures for his particular task. There has been very little emphasis on quantitative methods for data structure evaluation.

Development of a systematic design methodology

Also the design in the past has been ad hoc with little adherence to any systematic methodology.

A designer is left bewildering with the mass of data structures, but no tools to assess their relative strengths. The thesis aims to remedy this situation by consolidating the work that has been done on design and analysis methodologies of CAD

data structures, particularly the Boundary Representations.

1.3 Research methodology

The problem the thesis is attempting to solve is the development of tools for evaluation of CAD data structures. The methodology we adopt is to first study the various applications of CAD data and hypothesize the requirements of an ideal data structure. Such a requirements analysis needs data collection about the CAD information requirements. I gathered the information based on my personal experience having been trained and worked in the two major disciplines of CAD i.e. civil and mechanical Engineering. Contact with other researchers and vendors at both symposia and trade shows and last but not the least, data gleaned from literature helped to form a good perspective of the requirements. Perhaps a more systematic collection would enable refined requirements, but is too involved because of the diversity of CAD applications.

Armed with the requirements, a systematic and formal approach is proposed for the design of boundary data structures by drawing on the accumulated wisdom over the past in the literature. We supplement the design formalism with a framework for analysis of various existing data structures, which enables us to determine their relative strength and weaknesses and to fine tune the data structures. Although the main emphasis is on the data structures often big improvements come through algorithmic analysis. We delve into common algorithms in CAD modeling. For completeness of the investigation, the thesis also addresses important issues of convertibility and validity of the boundary data structures.

The theoretical approach is supported by an operational account of the implementation of the ideas which required extensive programming in C and C++.

1.4 Organization of the thesis

The thesis has been conveniently organized into five logical parts. Each part is largely self-contained and starts typically with a survey and critique followed by requirements analysis and our recommendations. The first part establishes the supremacy of the boundary data structures and lays down the criteria for assessment of various data structures. It also motivates the thesis. The second part discusses the manifold data structure design and analysis and serves as a gentler introduction to the more complex non-manifold boundary data structures. The third part starts with a comprehensive survey of the non-manifold data structures and then similar to the second part discusses their design and analysis followed by the considerations of validity of non-manifold representations. The fourth part is an account of the implementation of the ideas in the previous parts and also some important algorithms for model manipulation and data conversion. The last part rounds up the conclusions and outlines future work.

Part I

CAD data structures

In this part we briefly survey the requirements of CAD data structures and assess many popular CAD data structures for suitability and establish criterion for the latter parts.

Chapter 2

A Survey of CAD Data Structures Requirements

CAD serves a variety of applications, which are enumerated in the next section. The subsequent section derives the key requirements of all such applications. Such a requirements analysis is a precursor to the identification of the assessment parameters of CAD data structures.

2.1 CAD data applications

1. *Automatic computation of mass properties* CAD data enables the calculation of weight, intertia and other volume properties for use in a variety of applications.

2. *Visualization* Ability to look at a part before it is actually manufactured, is possible by either shaded images or line drawings of the objects.

3. *Finite element mesh* FEA (Finite Element Analysis) is indispensible for assessing the performance of a part under different conditions (such as load, temparature). CAD model aids in the automatic generation of finite meshes of a part.

4. *NC programs* CAD model can be used to generate automatic NC (numerical control) instructions for the control of a machine to actually manufacture a part.

5. *Feature recognition* Geometric data from CAD model can help in the recognition of shape features (such as slots, through holes) which is invaluable in the subsequent process planning.

6. *Process planning* Allied to NC programming and feature recognition is the process planning (the determination of various machining and welding operations) which can be automated with the aid of CAD data.

7. *Design for manufacture* Automation of the NC program generation, feature recognition, process planning and design by features collectively forms a blue print for automated design for manufacture.

8. *Drafting* Automated drafting, which does away with shelves of potentially inconsistent drawings, is made possible by CAD model.

9. *Automatic dimensioning* It is an an important part of drafting.

10. *Data exchange* Without a suitable mechanism for communication of geometric data, a great deal of redundant effort is expended in building and maintaining consistent multiple geometric models of the same part.

11. *Fit, interference and tolerance analysis* Useful for the subsequent assembly planning. An assembly is cut with many parallel planes to check for fit etc.

12. *Assembly planning* Complex parts are built by an assembly of standard components.

13. *Rapid Prototyping* Prototyping enables to physically check parts before launching a full scale manufacture. Rapid prototyping is very useful aid for analyzing form, fit and function and for the manufacture of certain molds. Rapid prototyping is an excellent alternative to the manual, costly and long conventional prototyping.

14. *Simulation* CAD model is a key ingredient for simulation of complex machinery or process.

15. *Virtual reality* Simulation and recreation of natural world systems is a new customer of CAD.

16. *Data storage and retrieval systems* For archival of huge geoemtric data, CAD models are better than shelves of drawings.

17. *Miscellaneous - e.g. mechanism analysis* Kinematic analysis etc. is made possible by CAD data.

A study conducted by Johnson [1986] rated the order of usage as visualization, mass property, layout of tightly packed assemblies and FEA of complex shapes. Unfortunately to the present author's knowledge there are no other studies reported in literature. Although the list has been updated from sifting through general literature, it is possible that some important applications of CAD are missing from the

list. Also the technology is rapidly advancing and every day CAD finds new users and applications. One such recent addition to the list is virtual reality systems with an enoromous appetite for masses of geometric data. An extensive and detailed model is a pre-requisite and a quick access is indispensible to guarantee that the virtual reality is at par with the actual reality and its associated human experience.

2.2 Requirements of CAD system

An understanding of various application requirements is essential for developing sound design methodology and analysis techniques. More specifically, the requirements analysis forms the framework for the evaluation of CAD data structures, in the succeeding chapter. In this section we review the literature on application requirements.

Certain portions of this section are based on the solid modeling study conducted by Johnson [1986].

1. *Good support for user for modeling objects is a prerequisite.*

   (a) *Support for early design.* It should reflect the natural process of design proceeding from a trial and error early stage design, where parameters are not pre-specified and only certain positional constraints fixed, to a final fully detailed CAD model. In the early stages, design works in abstraction rather than full blown and complete specification. Thus we need a facility to defer full specification and work with few constraints. Also in the early stages, the designer starts with a sketch, fleshes it with sheet and volume as and when the user feels the need for it. This implies that the data structure should be capable of maniuplating not only solids but also wires and sheets.

(b) *Support for editing of models* Ability to reuse existing models, calls for automatic purge of unused features and automatic unduplication (i.e. artifact removal). Designers usually make use of existing models for launching the design of a new model instead of starting afresh. Hence it is essential to provide a good editing tool for cut and paste type operations. A familiar analogy is text editor such as *vi* which provides an undo operation, unlike most other PC based. Usually 10 or 20 versions of an existing model are used and hence special efforts are needed to ensure that the representation is minimal and most efficient by getting rid of artifact faces, edges. Without such a mechanism, storage and time complexity of the models grows exponentially and reliability diminishes rapidly with version number. The facility should preferably be automatic (e.g. some modelers require a user command to coalesce two adjacent faces). This and the support for curved surface, imply that faces must be naturally definable (i.e. all of a connected surface corresponds to a single face). Note that storing some redundant information aids efficient editing, but conflicts with the economy requirement.

(c) *Retention of designer's intent* The designer usually works in terms of features rather than geometric entities. Tools to capture designers intent, which is lost with plain geometry, are essential. Hence interactive methods to define and manipulate features (e.g. slots, bolt hole circle, datum reference for dimensioning). Such a high level description is also useful in the down stream manufacture: for process planning and machining. Support of features is easier with non-manifold data structures.

(d) *Modeling tools* such as booleans, sweep and unary operators are essential. Booleans and sweeps are the subject of later chapters. Unary operators involve one solid only and include scaling, rotation and reflection. They do not change the topology and hence do not form part of this thesis.

(e) *Miscellaneous* Ease of learning and use for modeling parts, assemblies, multi levels of assemblies and standard component. Provision of alternative ways to model a part (e.g. basic shape definition followed by refinement) and automatic filleting and blending where fillets meet and definition of purposeful tangencies. For assemblies, ability to measure and use them subsequently is essential. This calls for a hidden line support, rather than a hidden surface, which works on pixels and thus loses dimensioanlity. Hierarchical descriptions of multi levels of assembles is desirable. For standard parts, ease of look up and security against accidental modification should be provided.

2. An ideal modeller must have an *exact representation* of curved surfaces (regular quadratic surfaces are the most frequently used in the manufacture). As discussed in later chapters, curved surfaces dictate that certain minimum relations be stored to ensure data structure unambiguity. Also they are difficult to manipulate. On the other hand polyhedral approximations require excessive amounts of storage (and hence time) for reasonable accuracy (for example, polygonal approximation of a sphere with 81920 faces requires 50 Mega Bytes of topological information). Thus our emphasis for unambiguity of representation in the curved domain. For CNC (Computer numerical control) manufacture (CNC), the accuracy of the polygonal approximation must be greater than CNC

machine.

3. It must be possible to assign *attributes* (e.g. color, surface roughness) to faces and other boundary elements [Pratt 1987, Dietrich, Nackman, Sundaresan and Gracer 1989, Johnson 1986]. It it must be possible to assign dimension and tolerance to edges and implicit features for edges and vertices (e.g. chamfered corner). This implies that it must be easy to access the basic shape i.e. face, edge and vertex.

4. The CAD model must be *robust* against all errors. Unreliability arises due to two problems - impossible objects and latent defects. Visual feedback is adequate for the first, but not for the latter. An automatic detection of the problems is difficult and or extremely slow. A later Chapter (see Section 12.2.1) evaluates the existing automatic methods. Latent defects, lead to a sacrifice in the integrity and remain dormant for a while. Since they do not show up immediately, the simple visual feedback is inadequate. Surface intersections between different types (e.g. quadratic) should be error proof. The CAD modeling must be robust against complex objects and singularities and coincidences i.e. coincidences and unusual topological connectivities. Coincident edges and points leads to non-manifold conditions and latent errors. Such problems can be avoided by adopting non-manifold data structures.

5. *Processing time* is roughly proportional to the product of the number of faces involved with explicit face representation. This again argues against polygonal approximation. Time should preferably be linear with accuracy and complexity.

6. *Analysis facilities*: high accuracy of volume estimates and interference

7. Fast modeler *outputs* such as hidden line, shading as a function of resolution, file transfer size are essential for interactive modeling.

8. To cope with complex *assemblies*, information on the parts list of an assembly and each part's list of assemblies (where it is used) are very important. Each part should have only one solid model and its occurrence in each assembly should be represented by a transformation, instead of duplicating the full blown solid model in each assembly. Standardization encourages sharing parts across assemblies. A bill-of-material kind of structure, which maintains the part list of an assembly and the number of individual parts required in each assembly is helpful.

9. *Processing cost.* For modeling to be economically viable and competitive, it must be cheaper to process than a manual approach (e.g. a comparative wire frame and surface models). This puts a lot of emphasis on the accessibility.

10. *Integrated data base management system*: it should also be possible to access work station based models instead of hard copies of drawings from shelves. This argues for an integrated data structure.

11. Good support for *rapid prototyping* is essential. A CAD system and rapid prototyping bear a relationship similar to a word processor and a printer. It reduces the gap between design and manufacture drastically. Prototyping usually requires a valid solid model with a triangulated boundary. The model is then sliced into very thin layers (0.005 inches to 0.002 inches).

12. *Local modification* should be easy to incorporate. Local operations which are common in engineering objects are tweaking, filleting, blending and chamfer or

bevel. Chamfer and fillet (used in both casting and welding) are usually provided for stress relief and to avoid sharp corners. It involves the replacement of an edge or vertex by a new face. Filleting also involves the same, but usually with a quadratic face. To provide draught, which facilitates withdrawal of a pattern from a mold, a collection of faces are pulled together i.e. tweaked. This usually involves, provision of a slight slant for straight faces i.e. alteration in the geometry and no change in topology. In chamfer, the face may be planar or quadratic (cylindrical for edge and spherical for a vertex), in which case it is known as blending. The usage of splines is not so widespread as quadratic surfaces. In local operations, it must be possible to retrieve the fields of an arbitrary face or vertex or edge and update the appropriate topology and geometry. As an example, for the chamfer on an edge, we need to identify the affected face, edge and vertex identifiers. Local modification will be discussed in Part 4.

13. *Astronomical model complexity.* Greenberg [1991] predicts that increased model complexity will challenge computer graphics during the next decade (see Staudhammer [1991]). Greenberg states that "the size of typical modeled environmenmts when combined with meshing and adaptive subdivison techniques, will increase by two to three orders of magnitude in geometric complexity alone". This means that in the short term we have to opearte in a virtual memory environment which alone can cope with gigantic sized models. The performance of data strcutures in such environments is one of the chief topics of this thesis, in the Chapters.

14. *In summary*, time, output and memory are more important than accuracy and robustness and hence form a major plank of this thesis. For accuracy good

support for curved surfaces is essential. Curved geometry technology is still in its infancy (specially the surface-surface intersections which are intractable). The problems due errors can be mitigated to some extent since one can comeup with a robust alternative with a fast modeler which responds quickly.

2.3 Conclusions

In this Chapter, we described several applications of CAD data and focussed on the demands placed by them on CAD data. In the next Chapter, certain exciting systems which embody several of the applications of CAD will be presented and arguments will be put forward for an integration of CAD with the whole system.

Chapter 3

Integration of CAD data

The previous Chapter surveyed several uses of CAD data and thus argued for integration in a general sense. This Chapter amplifies on the need for integration of CAD data with other activities. To do this we take a specific case of construction robotics and describe its components. This is preceded by a brief description of the role of CAD in computer vision, which is an integral part of advanced automation systems.

Using a common CAD data base for design, planning, manufacture, simulation, computer based inspection, robot navigation and manipulation, enhances efficiency and productivity. This single data structure serves all requirements and avoids data integrity problems. In the past, data has been isolated for specific tasks and, whilst this served the short term goal of optimizing a specific activity, redundancy was inevitable. Integration of CAD has the potential of realizing the ultimate goal of computer integrated work which embraces both CIM and CICI (Computer Integrated Construction and Inspection).

3.1 Two applications of CAD

Two major exciting applications of CAD data which are relatively young: Computer vision and simulation and off-line programming of a robot are described below.

3.1.1 CAD based computer vision

Object recognition is a fundamental area of research in computer vision. It typically involves identifying and labelling each object from a jumble of objects. The process involves image acqisition followed by a search for a suitable match with a model from a model data base.

Approaches for modelling

One approach to sensor based modeling is to take several range images (e.g. by laser range finder from several viewpoints) and combine the object points through the necessary transformations. The errors in the data acquisition step due to calibration, noise etc. affect the quality of model and the manual approach limits the number of possible models and also the number of surface points for each model.

Alternatively, most of the parts already have a CAD model which may be used directly or augmented with information necessary for visual tasks. However, in the past most of the research in recognition involved models built manually by scanners instead of using a CAD data base. Examples from the literature follow.

Literature review in CAD based vision

Many of the papers used inhouse research CAD models, different from the commercial CAD data bases, which don't permit access to the internal data structure. The

matching strategy based on data obtained by an access to the internal data structure is more efficient in the search process.

Arman and Aggarwal [1990] used a commercial CAD data base (the CATIA modeller from IBM) which does not provide access to the internal data structure. The system extracts from the CAD data base surface normal, bounding edges and vertices of each face and the number of faces. From this information, for each face, its area, length of each edge and the angle between every adjacent pair of edges is calculated and stored for matching with similar measures for the surface patches from range data. The range data is segmented and surface patches detected using the curvature and surface normal property of each point. For each of the planar patches, the same information as for the model faces is calculated.

Another recent CAD based recognition work is by Hansen and Henderson [1987] who used a conventional CAD modeller, from which a special purpose CAD model (consisting of information such as the adjacent faces for each edge) is built. Like Arman and Aggarwal [1990] they also use face area and edge length measures but the dihedral edge angle (the angle between the normals of the two faces sharing the edge), instead of the angle between every adjacent pair of edges. Similar to Arman and Aggarwal [1990] they detect surface patches from the range data and calculates the three measures.

Nurre, Hall and Ronig [1988] give an example of integrating computer graphics and Computer aided engineering (CAE) with machine inspection.

3.1.2 Simulation & Off-line programming a robot from a CAD system

Stobart and Dailly [1985] describe the implementation of simulation in the BUILD Solid modeller. The BUILD solid modeller was developed over a period of 10 years at Cambridge University. Simulation is a powerful tool for collision detection, inverse kinematics and handling the uncertainty in the work environment.

For robot simulation, the solid model is augmented with a kinematic model which has the joint parameters (joint rotation and position). The solid model contains the conventional boundary information of the joints (commonly as cuboids and cylinders) and their position relative to the other joints. Also, the enclosing box of each constituent part of the robot and other objects in the work space is included. This avoids repetition of the computation of the enclosing box, when the robot assumes a new configuration. It is useful for fast collision detection. As the robot takes a different configuration the corresponding joint parameters replace the parameter values in the computation of the transformation matrix. This new model can then be manipulated like an ordinary solid model.

The boxes are also transformed and their intersection enables collision detection. Cameron [1984] uses this test to compute the minimum separation for the avoidance of collision between any members in the workspace. The boxes are exaggerated to account for uncertainties. This automatic collision detection saves the programmer from the cumbersome manual process of guessing correct view points, for the graphic display of the work place, to detect possible collisions.

Off-line programming coupled with AI techniques helps the programming to be done with a computer, using a simulation package, while the robot continues to work

undisturbed. In the traditional teach method, the robot is involved in the program development phase also and thus holds up the robot from other useful work.

Simulation as described above facilitates the trajectory planning by collision detection. The CAD model of the workspace and the robot which has already been captured during the design process, may be exploited in the program development. The off-line program may require fine tuning in the online mode using record and playback, before being made fully operational. This fine tuning can be avoided by the provision of sensory feedback (e.g. vision and tactile sensors).

3.2 Integration of CAD and robotics in the future

In the previous section, we described a sample of the interesting applications made possible by CAD data. To achieve the goal of integrated automation it is advantageous to extend and integrate the CAD data model for the whole gamut of applications: from simulation to real manufacture, including the requirements for vision tasks, like inspection. Such schemes should span the gap between the initial design and planning of the project, and the machine intelligence requirement for the ultimate control activity, as exemplified by the following scenario of the construction robotics unit at the City University, London.

Figure 1 outlines an integration strategy currently being pursued in the provision of a data base, the key elements shown in their context. A wall climbing inspection robot and a wall assembling robot [Chamberlain, Speare and Ala 1991] and [Chamberlain, Ala, Watson, Reilly and Speare 1992] are the immediate targets for this work. The provision for finite element analysis arises from the flexible nature of these and the concern for their vibration characteristics. Inhouse and CAD software is being

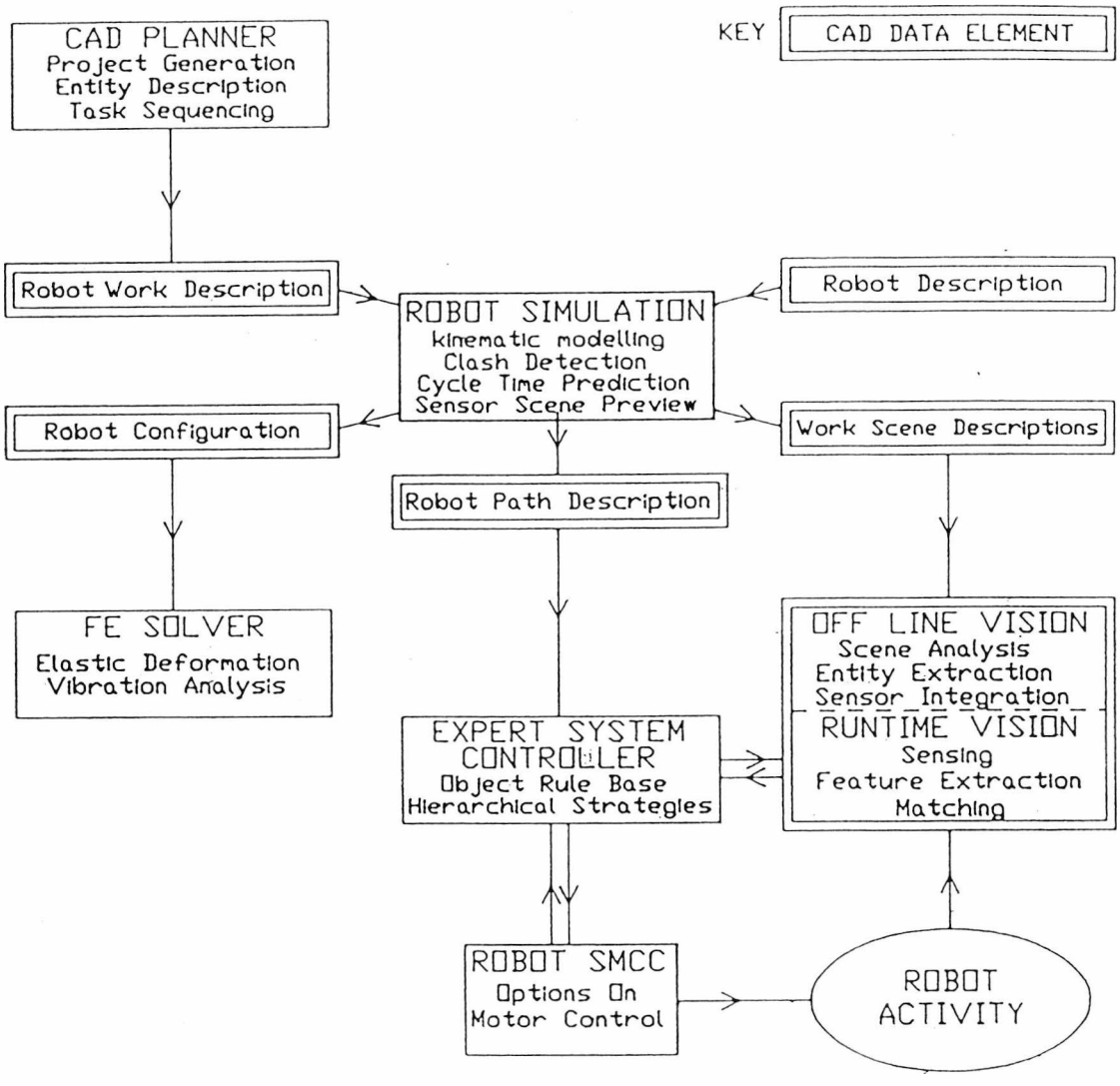

Figure 1: CAD data integration strategy

used for design and planning activities, GRASP [BYG 1992] for kinematic based simulation and in house development of an expert system environment for machine intelligence. Each of these operate on, and extend, the common CAD data base.

The evolution of the CAD data base and its integration with the other data can be classified into five stages, these being apparent in figure 1. These are considered in the context of the two robots previously mentioned.

3.2.1 Cad planner with 3-D draughting

The wall assembly task can be modelled using the "3D block entity" method, this comprising lower order entities such as points and edges. This entity together with other high order entities represents a "parts kit" for the project. A 3D approach is preferred as this overcomes the 2D to 3D interpretation problem. Projects are built up on an interactive basis with setting out aids such as plan grids. Logical task sequencing is built into this by object rule direction. The inspection activity can also be substantially enhanced by the preparation of a terrain and work detail representation.

3.2.2 Robot simulation with runtime attributes

This simulation facility allows the robot and planned task to be integrated and provides assessment of clash detection, production time cycles as well as explicit visualization of the Camera and eye-in-hand sensing which can be modelled with key scenes prestored. At any stage the robot configuration can be passed for structural performance analysis.

3.2.3 FE solver for structural assessment

Data for forward kinematic analysis including large deflection static and vibration analysis, can be transferred from the simulation facility. As construction robots are likely to be more flexible than their manufacturing industry counterparts, this is an important consideration.

3.2.4 Offline and runtime vision for inspection and surveying

In order to enhance the vision sensing performance, it is necessary to maximize the benefits of the accumulated CAD data. Strategies such as extensive masking off of unwanted detail are extremely important in this. The work scene description derived from the simulation process can be substantially preprocessed for the ensuing runtime vision task. Wall unit alignment and potential unit collision stages are the main targets for this.

Construction is highly unstructured compared to the manufacturing industry and hence parts and machines can not be tied down or guided by fixtures and other conventional gadgetry of the established manufacturing industry. The machine vision requirement is thus diverse and complex.

3.2.5 Expert system controller for robot activity

This has been approached at a high level by adopting a rule based, object orientated, expert system shell development environment. In this, the completion of task stages are the top level goals, which in turn must satisfy the sub goals for sensed states and sequence forcing. This facility communicates with the robots smart motion control

card (SMCC) in the case of the wall assembly robot, and a development board in the case of the inspection robot.

These collective and integrated facilities represent a blue print for robot cell development in the construction industries factory of the future and serves as a case study for integrated CAD environment.

3.3 Discussion and Conclusions

The above description revealed the need for additional information beyond the geometric data provided by the CAD system. For example, there are differences in vision and CAD requirements. Recognition is concerned with objects which already exist. On the other hand CAD is concerned with interactive design of new objects. CAD representations also tend to be view independent. Computer graphics is used to render the objects from different view points. In a nutshell, CAD systems stress the interactive design, set operations, rendering, finite element analysis and CNC programming. Vision models need augmentation with imaging and illumination models. Stable orientations, view potential or aspect graph and surface properties (e.g. texture, color, reflectance) information is also needed.

The differences in CAD requirements notwithstanding, this chapter has highlighted the importance of CAD and the diversity of its applications. The background for the requirement of a fully integrated and enhanced CAD data base for robotic inspection and construction has been presented. For maximum benefit the information technology must cover all stages from the initial planning of the task to the multisensor assisted robot activity. Activity simulation is seen to be usefully linked to the vision facility, providing scene previews for off-line processing. However for practical

implementation the nature of the actual data structures merits close attention if fast run time access of vision data is to be achieved. A contemporary progress of construction robot development provides the impetus for the development and improvement of the run time machine vision implementation.

The goal of computer integrated work (CIW) seems feasible. However, it may be too early to expect a completely paperless work environment controlled from design to quality assurance of the final product (be it a construction site or a precision manufactured part) by a bunch of silicon chips.

Because of the enoromous potential of CAD, this thesis is concerned soley with the organization of the geometric data.

Chapter 4

An analysis of CAD data structures

4.1 Assessment parameters of the CAD data structures

In chapter 2 an exhaustive categorization of application requirements was undertaken. Based on the CAD data requirements enumerated, we identify and define the following parameters for assessment of suitability of the 3D object representations for CAD data. Various references on the rating in the literature include Marr [1982], Bhanu and Ho [1987], and Mantyla [1988].

Accessibility Ease of extraction of the shape of the object from the representation. Differs from algorithm simplicity, in that only the basic shape such as face, edge and vertex are considered.

Accuracy Degree of closeness between the object's actual geometry and the representation. Polygonal approximation or spatial enumeration involve a prespecified resolution, a fine resolution has a bearing on the storage, in that it pushes the

storage up drastically. Note that accuracy here does not mean the accuracy of calculations, such as the accuracy of intersections. Such an accuracy being a function of the finite precision of computers, influences very little the choice of CAD data structure type.

Algorithmic simplicity How easy it is to devise algorithms for analysis (e.g. mass properties), and manipulation of the represntation (e.g. combining two objects). This obviously depends on the application and hence the rating is not very instructive.

Closure It is desirable to have a consistent representation when objects are manipulated. This is a measure of how well the representation remains within the original when the objects are manipulated (for example, does the boolean intersection of two objects belong to the same category as the orginal representation).

Conciseness Economy of representation

Connectvity Ease with which it is possible to navigate on the object shape.

Convertibility Ease of conversion to other representation forms.

Meaningfulness How meaningful is the represenatation ? Does it correspond to nonsense objects sometimes.

Scope Representational power i.e. the types of objects representable.

Sensitivity A measure of the representation's ability to depict the differences between similar objects.

Stability A measure of the represenation's ability to depict the similarity between two different objects

Unambiguity Is the representation prone to multiple interpretation i.e. multiple objects.

Uniqueness How well the representation guarantees only one representation for a given object.

User friendliness The ease with which the data structure can be described by a user. With a friendly data structure, it may be enough to specify a cylinder as a cylinder of radius 10 and height 5, but a hostile data structure may require the specification of coordinates of the individual points of the whole space occupied by the cylinder. This is related to compactness, the more voluminous the data the more difficult for the user to input data. But in some cases the data may be avaliable without the user having to describe himself or it may be possible to convert from a secondary representatation. In such an occassion, user friendliness has very little bearing on the choice of CAD representation. Thus it is rated high, if it is not cumbersome for the user to generate the data.

4.2 Representational Methods in CAD

The most popular CAD representations are Boundary representation (B-rep), Constructive solid geometry (CSG), Sweep, Cell decomposition, Spatial, Parametric, Aspect graphs and the Extended gaussian image (EGI).

In Boundary representation, the object is represented in terms of faces, edges, vertices i.e. the boundary of the object. In the CSG, the objects are represented in

terms of boolean operations on solid primitives (cuboids, cylinders etc.). In the sweep representation, a planar cross section and the axis along which it is rotated (rotational sweep) or translated (translational sweep) represent the objects. Generalised cylinders [Brooks 1984], is an example of the sweep representation.

In spatial representations the occupancy of the whole space filled by an object is recorded. Usually the space is subdivided into small cells and the presence or absence of the object material for all these cells is recorded. Spatial occupancy enumeration can be viewed as a special case of cell decomposition where the cells are cubical and occur in fixed orientation but different location. Two popular forms of spatial occupancy are the quad trees and octrees for 2-D and 3-D shapes respectively. In parametric instancing, the object types are predefined, but their parameters (i.e. dimensions) can vary. Parametric shapes encourage standardization and are useful if all the manufactured parts can be classified into a few parametric classes with varying parameters, similar to the concept of group technology.

EGI is a mapping from an object's surface normals into a unit sphere called the gaussian sphere. Gaussian curvature is equal to the area on the gaussian sphere mapped from a unit area on the object's surface. EGI is the inverse of the gaussian curvature (e.g. for sphere of radius r the gaussian cuvature is $1/r^2$ and the EGI is r^2). For concave objects, EGI is not unique. EGI finds application for pose determination of object (i.e. orientation) from images.

Koenderink and van Doorn [1979] originated the concept of the aspect graph or visual potential graph. Visual potential of an object is a representation of the visual perception of rigid bodies by ambulant observers. The aspect graph has a node for each topologically distinct view of an object (termed the aspect). Thus each node represents a discrete cell of the view point space parcellation and has a single

aspect. Each arc connecting two nodes represents a change in aspect (termed as visual event), when the orbit of the observer traverses the border between two such cells. It is important to recognise that only the aspects which differ in exactly one attribute of an aspect, are connected by the arcs.

The above CAD representations can be classified into two modeling schemes: Object centered and Viewer centered. The object centered representations: CSG, parametric, cell decomposition, spatial, B-rep and swept volume share a common property of object centeredness, since they represent the volume of Euclidean 3-D space that the object occupies. The representation of the object as a function of the view point (as against volume of space occupied in 3-D, in the object centered representation) is called viewer-centered. They find application in graphics for rendering puposes (e.g. hidden line), in vision for recognition and in robot path planning for obstacle avoidance . The object-centred representation is well suited for boolean operations and is compact. Marr [1982] argued that an object-centreded representation is suited for recognition also because of view-point independence. Rosenfeld [1987] felt that viewer-centred representations may be necessary to reach the speeds at which human beings recognise. It may be noted that rendering converts data from the traditional object centred representation into viewer-centred representation.

The object centered representations can be further categorized into the following classifications:

- Procedural
  - CSG
  - Sweep
- Volume based or interior based

- Cell decomposition

- Spatial

- Parametric

- Evaluated

- B-Rep

Parameter	Object centered						View cntr.	
	Procedural		Volume			Eval.		
	CSG	Sweep	Cell Decom.	Spatial	Parametric instn.	B-Rep	Aspect Graph	EGI
Accessibility	2	4	1	1	2	5	1	1
Accuracy	5	5	2	1	5	5	1	5
Al. simplicity	2	2	3	4	4	3	1	1
Closure	5	5[a]	2	4	2	4	5	5
Compactness	4	5	2	1	5	3	1	3
Connectivity	2	3	1	1	2	5[b]	1	1
Convertibility[c]	3	3	1	1	3	4	1	1
Meaningful	4	2	2	4	4	2	4	4
Scope	3	1	4	4	1	5[d]	2	4
Sensitivity	3	3	2	2	4	4	4	1
Stability	3	4	4	4	4	4	4	1
Unambiguity	4	4	4	4	2	4	1	2
Uniqueness	2	2	2	4	2	3	3	3
User frndl.	4	4	1[e]	1[f]	5	3[g]	1	1

Table 1 An analysis of various CAD data structures

[a] can be 1 if sweep distance is not constant

[b] can be 2 if an improper data structure is used

[c] depends on the data structure involved, the values form a rough guide only

[d] can be 2 if an improper data structure is used

[e] can be 5 if converted from an auxiliary data structure

[f] can be 5 with mechanical acquisition e.g. image capture

[g] can be as good as CSG with conversion from auxiliaries

The above standard representations can be characterised as shown in Table 1, in

terms of the parameters identified. Each representation is given a star rating against each parameter, ranging from single to five star rating. The rating also illustrates simple parameteric modeling. The rating is prone to misinterpretation. For example, spatial data strucures are rated very high on the scope parameter. However, to get the full picture, one must look at the accuracy with which the objects can be modeled. This means that spatial data structures can model a large class of objects, but at very coarse approximations for comparable space. This highlights the prominence of certain parameters. For example, sweep perfroms better or equal to CSG except on the meaningfulness and scope scales, yet sweep is inferior to CSG because of its poor scope.

The ratings are rather coarse (the scale admits intgers 1 to 5 only) and as such can be argued for alterations. Their purpose is to present a broad assessment perspective of all the representations pictorially in a single table. Even making allowance for potential alterations, the same broad conclusions, as described below, can be reached.

4.3 Conclusion

It is apparent from the analysis, that only CSG is a serious contender to the B-Rep, the rest being useful as auxiliary data structures, but not on their own. CSG models have a big demerit: since the robot can only see the object's surface, vision and trajectory planning inherently involve evaluation of the surface, which is time consuming for the CSG, while the B-Rep explicitly encodes the same information. It is apparent that B-rep is indispensible for a variety of applications. For example, as discussed before, there exist several methods for modelling data belonging to the two categories: object-centred and viewer-centred. However, Besl and Jain [1985] state

that at least one vision module for any vision system will require the determination of the surface characteristics. The reason is that intensity images are dependent on surface geometry and range images consist of surface measurments. It is therefore appropriate to retain the B-rep in conjunction with a suitable viewer centred model to provide for multiple levels of geometric information. So I studied the boundary data structures for data represenatation in great detail.

Part II

Manifold Boundary data structures

We start our development with the simpler manifold data structures, before launching into the complex non-manifold world.

Chapter 5

An Introduction to Boundary data structures

In the preceding chapters it was shown that CAD data is used in a diverse range of applications related to computer integrated manufacturing (CIM) and arguments were presented for an overall approach for data structure design, so that differing application requirements can be met from a single CAD model. It was argued that the virtues of the boundary representations make them an ideal choice for such a CAD model. For brevity, we also refer to boundary data structures by the shorter and popular name B-rep. In this chapter we briefly survey the origins and the evolution and some important boundary data structures. We also introduce definition and a notation which will be used in the subsequent chapters.

5.1 Definitions

5.1.1 Manifold and Non-manifold

Intutively a manifold's surface is locally flat or disk like. A formal definition of a manifold [Mantyla 1988] is *a topological space where every point has a neighborhood topologically equivalent to an open disk of E^2" (i.e. 2-D Euclidean space).* This property enables us to study E^3 models essentially by the simpler 2-D models, termed as *plane models* [Mantyla 1988]. However objects that are outside the scope are also of interest for geometric modeling and are termed as non-manifolds, which form the subject of Part 3 of the thesis.

5.1.2 R-Sets

An R-set is a bounded, closed, regular and semi-analytic subset in 3-D Euclidean non-manifolds, but form a superset of manifolds. Their study is considerably simpler than the general non-manifolds as we shall see in Part 3.

5.1.3 Topology

This thesis, like many of its predcessors (e.g. Weiler [1986], Hanrahan [1985]), is concerned with topology only (not geometry). Weiler defineed topology as *a set of properties invariant under a restricted set of geometric transformations. It can be theoretically derived from complete geometric specification.* The term *evaluated* signifies a measure of the amount of topological information available without such derivation. In other words, adjacency is defined in terms of proximity and order. It was later broadened, by several authors (e.g. Yamaguchi, Kobayashi and Kimura

[1991]), to include neighborhood, and containment (i.e. classification). Multi-graphs (more than one edge between the same vertices) and self loops (same vertex at both the ends of an edge e.g. a circular edge) can occur in manifold objects also. The above definition of the topology applies to such degeneracies also. Topology leads to efficient algorithms (topological checks are more elegant and less error prone than geometric tests) and is less subject to change than geometric specification of the implementation. Thus Weiler argued for separation of topology from the geometry.

However topology has many limitations. Topology alone can not guarantee sensitivity e.g. a cube, a cube at different position and a rectangular parallelepiped all have the same topology. However, full geometry alone can guarantee 1-to-1 relation. In Chapter 11, we discuss the related problems of topological sufficiency and 1-to-1 correspondance.

5.2 Notation

An upper case letter like E, denoting the entity, may refer either to the set of all edges or its cardinality, depending on the context. To refer to a particular instance of an entity we use a lower case letter.

The notation for a relation or mapping is an arrow between two entities, (e.g. $f \rightarrow V$ is the set of vertices adjacent to a given face 'f' and $N_{f \rightarrow V}$ refers to the cardinality of the set. For brevity we also use N_{Vf} to denote $N_{f \rightarrow V}$). N_v and N_f refer to the number of adjacent neighbours which may be edges, faces, or vertices for the given vertex v and face f respectively. 'N' without any subscript refers to N_v and N_f (i.e. the average number of topological neighbors for any relation).

We note that

$$N_{Ev} = N_{Fv} = N_{Vv} = N_v \tag{1}$$

$$N_{Ef} = N_{Ff} = N_{Vf} = N_f \tag{2}$$

$$N_{Ee} = 4, N_{Fe} = N_{Ve} = 2 \tag{3}$$

We obviously have $N_v \geq 3, N_f \geq 3$ and $N_e \geq 2$. Woo and Wolter [1984] proved that the average value of N_v or N_f is at most 6 for any solid. Hence we have $3 \leq N \leq 6$, a result which we use frequently in later chapters.

Note that a different definition of N_{Ee} is possible which includes all of the neigboring edges, not just four as above. Many of the data structures figuring in Part 2, use the former definition. Note: The above equations assume that the objects are linear 2-manifolds.

The notation $O(N)$ means *of the order of N* The term *constant time* connotes $O(N)$ and is order of magnitude lower than *linear time* i.e. $O(E)$. The terms constant time and linear time are preferred since they convey the relative order of magnitude.

5.2.1 Minimum number of relations

Topology is usually given by the adjacency relationships between the different entities. The number of relationships is the square of the number of entities. But we do not have to store all of them, as some can be derived from the rest. For the simple topology of Vertex, Edge and Face (denoted by V, E and F respectively), the minimum number of relations is 1, 2 and 3 respectively in the case of unlabeled, labelled and constant time (i.e. avoiding file inversions) representations. A simple example should clarify how to obtain the full topology from a single relation in the case of unlabeled.

For the general scope i.e. non-manifold, obtaining such minimal sets of sufficient relations is difficult. However, proving the sufficiency of a given data structure is less hard. Also difficult is the task of finding the maximal set of topological adjacencies. The domain of design methodology is limited by such maximal and minimal sets.

5.3 Previous work

5.3.1 Winged Edge

A classic discovery was the Winged Edge (WE) by Baumgart [1975], which sprang from the needs of model based computer vision. Edges were more reliably detectable than vertices and faces, from images. Hence the winged edge was biased towards the edge, with all edge oriented topological information explicitly stored as illustrated in Fig. 2.

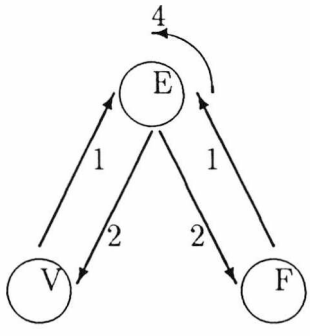

Figure 2: Winged Edge

Note: In the figures 2, 4, 5 and 6 the numbers on the arcs like $V \rightarrow^1 E$ imply that for each vertex, one topologically adjacent edge is explicitly stored. The arcs without a number like $F \rightarrow V$ in Fig. 6 imply that the number of vertices adjacent to a face are variable. A circular arc connects the same entity (e.g. the circular arc in

Figure 2 denotes $E \rightarrow E$ and implies that four neigboring edges are stored per edge).

Though it originated from the needs of computer vision it soon became sine-qua-non of all geometric modeling. However, it underwent several modifications to meet the needs of various applications. The proliferation of edge based data structures, ostensibly to improve access efficiency, enriched the three basic entities (viz. face, vertex and edge) by three new entities: segment (or Half-Edge), loop, and cavity.

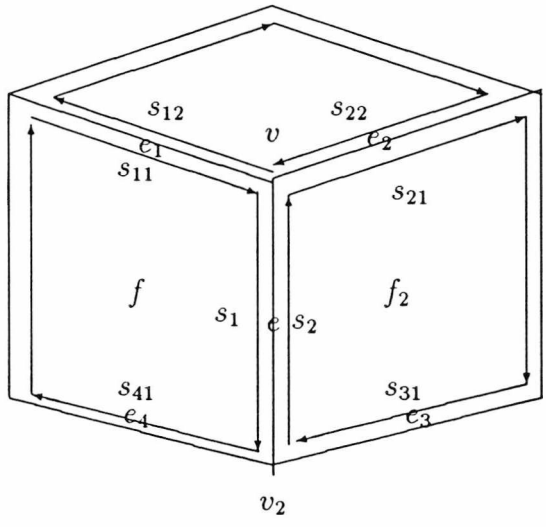

Figure 3: Topology of a Cube

5.3.2 Half-Edge

With regard to face, an edge has the dual roles of describing the face boundary as well as connecting two adjacent faces. In the winged edge a single edge record encodes the information about these dual roles. In the winged edge, a traversal (e.g. extracting the edges of a face) requires an additional check for the direction of traversal of each edge, as each edge occurs in the traversal of two faces (see Fig. 3 where edge e occurs

in both faces f and $f2$) i.e. an edge can be thought of consisting two halves. To obviate from this time consuming check, the single edge entity was split into two records (e.g. e into s_1 and s_2) in Vertex-Edge (VE) and Face-Edge (FE), proposed by Weiler [1985] and three records (two segment records and one edge record connecting the two segments e.g. s_1, s_2 and e) in Hybrid Edge proposed by Kalay [1989] and Half-Edge (HE) proposed by Mantyla [1988] (see Fig 4 for a stripped down version and Figure 9 for a fuller version). Each segment partakes in describing one face only (e.g. s_1 in f). Also a segment has only one vertex associated with it (e.g. In VE scheme, $s_1 \rightarrow V = v$) unlike the edge record of the WE, which has a two vertex array field.

Each segment in VE and FE data structures references its mate segment and two segments around its vertex and face respectively (e.g. In VE, $s_1 \rightarrow S = s_{12}, s_{22}$ while in FE, $s_1 \rightarrow S = s_{11}, s_{41}$). Analogous to segments in FE, each segment in HE and Hybrid edge references two segments around its face, but instead of referencing its mate segment, there is an edge record binding it with its mate). The Hybrid Edge and Half Edge differ only in their support structure (unlike HE, Hybrid Edge has no $v \rightarrow^1 S$ but a cavity entity) and linked list implementation (HE uses doubly linked list while Hybrid Edge uses singly linked). These differences, being a choice of the implementation, have no bearing on topological structure, and hence the two are considered identical in this thesis. For uniformity of comparison we assume that all data structures are implemented either by singly linked lists or arrays (see Section 7.6.2 for details).

The WE aimed at modeling solids or polyhedrons, in which each edge occurs in two faces. However architectural applications involve not only solids but also unidirectional lines and polygons (e.g. a landscape needs a polygon to model not

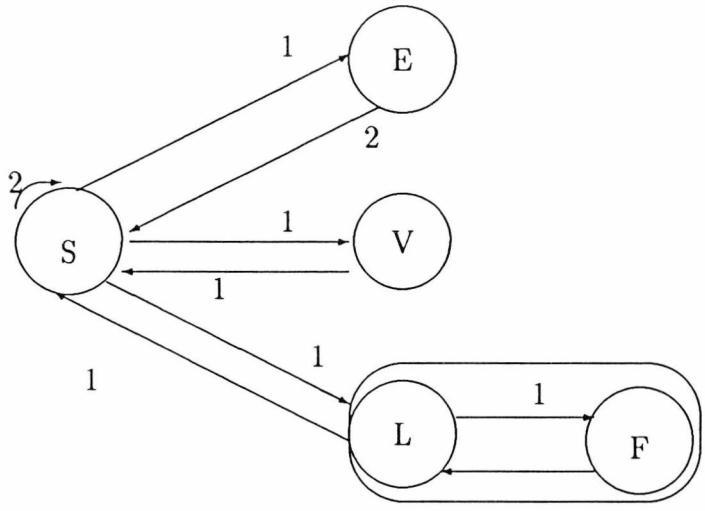

Figure 4: Half-Edge/Hybrid Edge Data Structure

a two sided face). Thus a half edge record, which was termed a segment record by Kalay [1989], can also model single sided polygons without redundancy.

5.3.3 Loop and Cavity

The original WE was extended to cater for multiply connected faces (i.e. faces with holes) by the inclusion of the *loop* entity by Braid [1980]. Each face is a list of loops. Each loop is described by a set of vertices or edges. Note that the half edge described above aptly describes the loop, as both loop and half edge are unidirectional, unlike the bidirectional face and edge. Multiply connected objects (i.e. objects with voids) can be handled by the addition of the *cavity* entity. Body and cavity have relations analogous to face and loop. However, some authors use shell entity to represent the exterior skin (i.e. the boundary) of solid objects aswell as the interior skin of voids. Thus the three basic entities were augmented by three new entities: loop, cavity

(shell) and half edge.

5.3.4 Symmetric Data Structure

Woo [1985] performed a combinatorial analysis of the data structures and proposed a new data structure [Woo and Wolter 1984], termed the Symmetric data structure (SDS), which is shown in Fig. 5. Woo's estimates for storage of different relations are tabulated in Table 2. The table shows that each of the adjacency relations (except $E \rightarrow E$) requires a storage of $2E$. The derivation of the results can be illustrated by $F \rightarrow E$. In summing all edges of all the faces, each edge occurs twice (since in a linear 2-manifold each edge bounds the two faces sharing it) and accordingly the total storage is $2E$. .

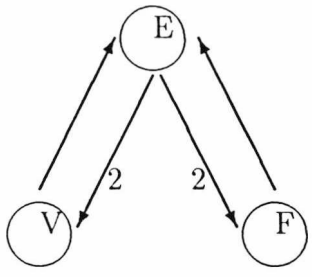

Figure 5: Symmetric Data Structure

SDS stores four relations and hence requires a storage of $8E$ (see Table 2). WE requires a storage of $9E$. SDS has been proved to be more efficient than WE in both space and time and consequently became a popular choice of many current implementations. It has been extended to represent hierarchical feature based geometric modelers by De Floriani and Falcidieno [1988], Falcidieno and Giannini [1989] and $3-D$ triangulation by Bruzzone, Defloriani and Puppo [1989].

Rel.	$F \rightarrow V$	$V \rightarrow E$	$E \rightarrow F$	$F \rightarrow E$	$E \rightarrow V$	$V \rightarrow F$	$V \rightarrow V$	$F \rightarrow F$	$E \rightarrow E$
Mem.	$2E$	$2E$	$2E$	$2E$	$2E$	$2E$	$2E$	$2E$	$4E$

Table 2: Storage values for different relations [Woo85]

5.3.5 Winged Triangle

Winged triangle proposed by Paoluzzi, Ramella and Santarelli [1989] (to be discussed in the next chapter) represents dimension independent polyhedra through simpilical decomposition. Non-manifold polyhedra are mapped to a set of manifold polyhedral surfaces.

5.3.6 Δ data structure

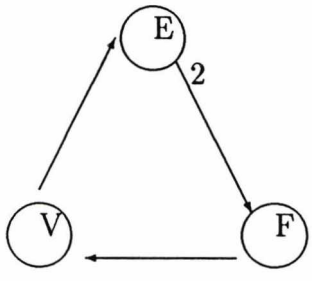

Figure 6: Δ Data Structure

Woo [1985] and Weiler [1985] studied the data structure efficiency. Both, however, lacked the perspective of a virtual memory environment. Wilson [1988] analyzed several data structures for communicability. Ala [1992] extended the analysis for virtual memory environments and proposed a compact Δ shaped data structure. The author also attempted to fill another gap in the literature: a systematic methodlogy for the design of boundary data structures.

Incidentally, the Δ data structure [Ala 1991] shown in Fig. 6, stores only three of the nine relations listed in Table 2) and hence has the least storage ($6E$ as against $8E$ and $9E$ of SDS and WE respectively) amongst the constant time data structures.

Chapter 6

Design Methodology of B-Reps

6.1 Introduction

6.1.1 Need for a universal data structure

As discussed in the preceding chapter, ever since the discovery of the classic Winged edge data structure [Baumgart 1975], there have been innumerable variations of it proposed. The word "edge" found qualified with all conceivable adjectives, and we were inundated with newly coined data structures (e.g. Hybrid Edge [Kalay 1989], Bridge Edge [Yamaguchi and Tokieda 1985], Half-Edge [Mantyla 1988], Split Edge [Eastman 1982], Doubly Connected Edge or DCEL [Preparata and Shamos 1985], Quad Edge [Guibas and Stolfi 1985], Vertex-Edge and Face-Edge [Weiler 1985]).

It is one of the aims of this chapter[1] to unite the multitude of these data structures under a common umbrella (referred to as Universal Data Structure, UDS for short) and present a comprehensive, yet clear view of the boundary data schemata. Also,

[1] Computational Geometry & Applications, 1(3), pp.207-226, September, 1991

once we optimize the UDS, we would have answered the optimality problem of all the combinatorially possible data structures.

6.1.2 Organization of the chapter

Since the chapter is rather long and involved, we describe its organization. In section 6.2 we propose the Universal data structure, which can be regarded as a complete generalization of the boundary data schema. We will also show that any data structure can be viewed as a particular subset of the UDS. Section 6.3 attempts to optimize the UDS and shows how the optimality principles of the UDS can profitably be employed in the design of alternative data structures. In section 6.4 we investigate the two approaches for the design of an optimal data structure: global and special purpose optimal data structure, which were suggested by Woo [1985]. Section 6.5 draws the conclusions. Section 6.6 comments on an optimal boundary data structure and gives an overview of additional considerations in the design of boundary data structures.

6.2 Universal Data Structure

6.2.1 Notation and Definitions

Notation

The notation has already been introduced in the previous chapter. However note that in section 6.3.3, 'N' will also denote the set of nodes i.e. entities (e.g. UDS has eight entities, as discussed in the next section). Also in section 6.3.3, we distinguish the cardinality of a set from the set, such as the set of arcs A, by $|A|$.

Definitions

Some of the terms used in Section 6.3 are explained. Note that they are not standard definitions but explain their semantics as used in this chapter. *OR Problem* is a mathematical formulation which can be analyzed by a set of techniques (chiefly linear and non-linear programming). *Hamiltonian Cycle* is a closed path such that every node is the terminal node of exactly one arc. *Eulerian Cycle* is a closed path that traverses each arc exactly once, goes through all nodes and ends at the starting node. *Self Loop* is an arc from a node to itself (e.g. $E \rightarrow E$). *Transitve closure* of a graph is a graph whose set of arcs is a superset of the orginal set such that there is an arc for every pair of connected nodes. Note that although two nodes in the original graph may not have an arc pairing them, there may exist a path connecting the nodes by transitivity.

6.2.2 Entities of the UDS

The three basic entities (vertex, edge and face) are augmented with five entities: Universe, Body, Cavity, Loop and Segment (denoted by U, B, C, L and S respectively). Universe is at the apex of the hierarchy of the UDS entities, as shown in Figure 7 (Note that UDS is modeled by the complete graph of eight vertices while the Figure shows the hierarchy only. The only significance of the arrows is a downward hierachical relationship between two entities). It represents multiple products or multiple versions of the same product. Otherwise it can be eliminated from the list of entities with an associated reduction in the combinatorial complexity. We include it for the sake of generality.

Each body has its own set of cavities, faces, loops, edges, segments and vertices.

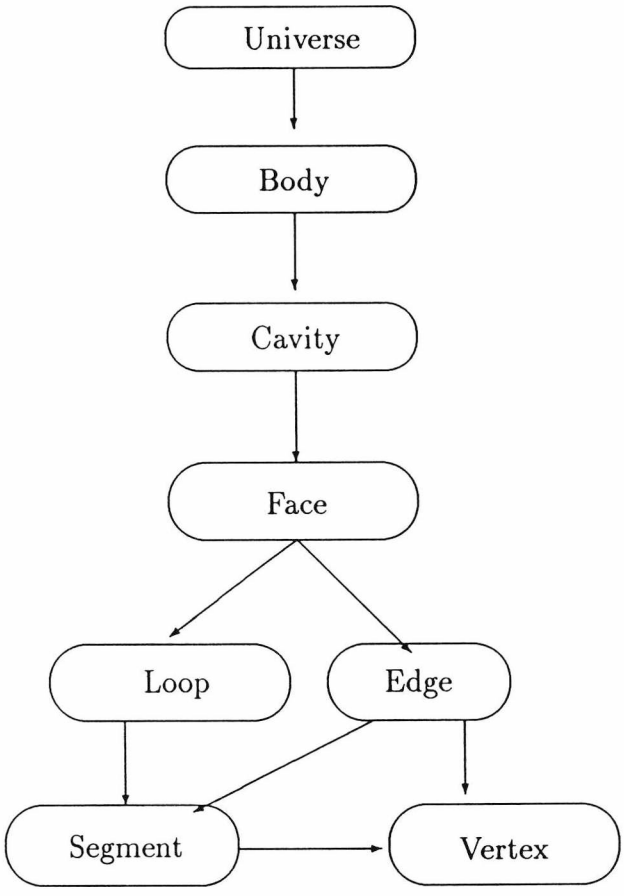

Figure 7: UDS Hierarchy

Objects with internal voids can be conveniently represented by the 'C' entity. Such an explicit representation (as against the implicit representation in terms of the shell) is used for the sake of generality. Bodies with multiply connected boundaries can be represented with a shell representing each of the boundaries. Thus Body and Cavity can be replaced by a single Shell entity, but to infer whether a particular shell is a Cavity, extra computation is required. The loop and segment entities were already discussed in Section 5.3

6.2.3 Relations

The possible number of relations in the UDS are $8 \times 8 = 64$. Some of these relations, for example $V \rightarrow U$, are trivial, since there is only one universe (barring the multiple models or versions mentioned in Section 6.2.2).

WE and several of its other derivatives have assumed that each edge is shared exactly by two faces. Thus each edge has two adjacent edges on each of the two faces sharing it, a total of four neighboring edges. WE stores only these four edges, whereas as mentioned in Section 5.2 an edge can have additional neighbors (which are not part of the two faces sharing the given edge). These assumptions are equivalent to $N_{e \rightarrow E} = 4$ and $N_{e \rightarrow F} = 2$. They do not hold good for all cases of UDS. Since there is only one Universe and any edge has exactly two vertices and two segments, the following conditions hold good in any case of UDS.

$$N_{b \rightarrow U} = N_{f \rightarrow U} = N_{l \rightarrow U} = N_{e \rightarrow U} = N_{s \rightarrow U} = N_{v \rightarrow U} = 1, N_{e \rightarrow V} = 2 \text{ and } N_{e \rightarrow S} = 2.$$

6.2.4 Euler's formula as applied to UDS

Let F^i, V^i, E^i, C^i, H^i and L^i be the total number of faces, vertices, edges, internal cavities, through holes and loops respectively for the i'th body. If we restrict all the bodies to polyhedra with each edge being shared by two faces, we have

$$F^i + V^i - E^i = 2 + 2C^i - 2H^i + L^i, 1 \leq i \leq B$$

For the UDS as a whole we have

$$\sum^B (F^i + V^i - E^i) = \sum^B (2 + 2C^i - 2H^i + L^i)$$

$$F + V - E = 2B + 2C - 2H + L \quad (1)$$

Since $E = 2S$, the above Euler equation can also be formulated in terms of S

6.2.5 Special cases of UDS

We show that any data structure can be represented as a special case of UDS, with example data structures drawn from current literature.

Case 1: Winged Triangle (WT)

This [Paoluzzi et al. 1989] is based on the triangulation of the faces and it is claimed that it is more compact than WE in the linearized representation of solids with curved boundaries.

By restricting UDS such that

(1) all faces are triangles,

(2) entities are limited to F, V, and E,

(3) relations stored are limited to $F \rightarrow V$ and $F \rightarrow F$ (shown in bold arcs in Figure 8 and others shown in light arcs) and

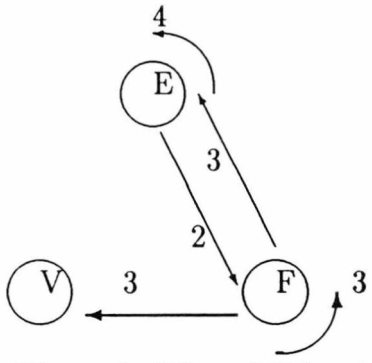

Figure 8: Winged Triangle

(4) $N_{f\to F} = N_{f\to E} = N_{f\to V} = 3, N_{e\to E} = 4,\ N_{e\to F} = 2$ (these are illustrated by the numbers on the arcs in Figure 8)

we get WT as a special case of the UDS.

Case 2: Half-Edge data structure

In Figure 9 , we use B and S, to represent the 'Solid' and 'Half-Edge'.

By restricting UDS such that

(1) all the entities except C are considered,

(2) relations stored are limited to $U \to B, B \to F, B \to L, B \to E, B \to S, B \to V, F \to B, F \to L, L \to S,\ L \to F, S \to E, S \to L, S \to V, S \to S, E \to S$ and $V \to S$,

(3) $N_{f\to B} = N_{l\to F} = N_{s\to E} = N_{s\to L} = N_{s\to V} = N_{v\to S} = 1, N_{e\to S} = N_{e\to F} = N_{e\to V} = 2, N_{e\to E} = 4, N_{s\to S} = 2, N_{l\to S} = 1$

we get Half-Edge data structure as a special case of the UDS.

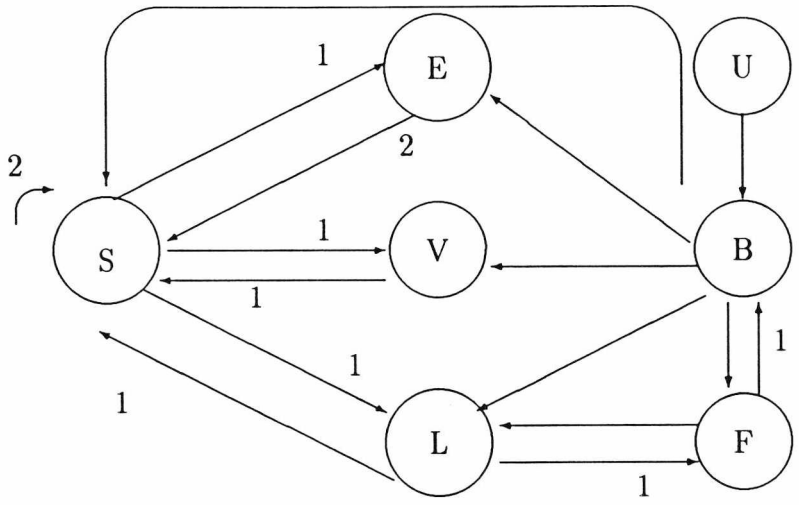

Figure 9: Half-Edge

Case 3: *3D* **Symmetric data structure**

An interesting extension of SDS to $3D$ triangulation was proposed by Bruzzone et al. [1989]. It uses an extra primitive B, the tetrahedron, in addition to the three primitive topological elements V, E, F. The $3D$ SDS stores 6 of the 16 adjacency relations of a $3D$ tessellation.

By restricting UDS such that

(1) all the bodies considered are tetrahedra,

(2) entities are limited to B, F, E, V,

(3) relations stored are limited to $B \rightarrow F, F \rightarrow B, F \rightarrow E, E \rightarrow F, E \rightarrow V$ and $V \rightarrow E$ and

(4) $N_{f\rightarrow E} = N_{f\rightarrow V} = 3, N_{f\rightarrow B} \leq 2, N_{b\rightarrow B} \leq 4,\ N_{b\rightarrow F} = N_{b\rightarrow V} = 4, N_{b\rightarrow E} = 6, N_{f\rightarrow F} \leq 6$

we get the $3D$-SDS as a special case of the UDS, as shown in Figure 10.

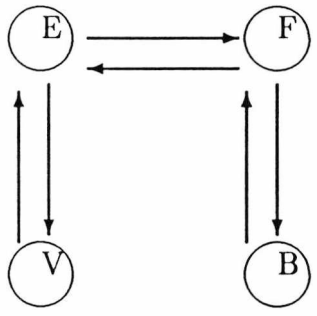

Figure 10: 3-D Symmmetric Data Structure

6.3 Optimization of UDS

6.3.1 Methods of optimization

We have 64 possible relations for the 8 entities of the UDS. It may be recalled that WE has used the concept of fractional relations whereby WE stored only one of the neighbours for the $v \rightarrow E$ and $f \rightarrow E$ relations. We thus have $64 \times 2 = 128$ potential number of possible relations for the 8 entities, including the fractional relations. The obvious approach for the optimization of UDS is the application of a backtracking algorithm, involving comparison of the storage and time for all possible data schemata of UDS. The number of possible data structures for n entities is 2^{n^2}.The backtrack algorithm will have a time complexity $O(p(n)2^{n^2})$, where $p(n)$ is a polynomial function of 'n'. For UDS $n = 8$, so we clearly have an algorithm not amenable even to the fastest CPU available. We therefore abandon the exhaustive enumeration approach and look for ways to reduce the search space for the optimal solution.

We can use the classical OR techniques [Wagner 1975] for formulation and solution of

the UDS optimization. An alternative approach uses the graph theoretic techniques [Harary 1972]. We discuss in detail both these methods.

6.3.2 OR approach

Problem formulation

Given a storage of M and the frequencies p_i of different queries, find the set of relations which have to be stored to minimize the total time required for the execution of the queries.

$$x_i = \begin{cases} 1 & \text{if the } i\text{th relation is explicitly stored} \\ 0 & \text{otherwise} \end{cases}$$

Let $1 \leq i \leq 64$ and $65 \leq i \leq 128$ represent the full relations and fractional relations respectively.

Note: In the above notation, x_i relation corresponds to its fractional relation x_{i+64}.

Let m_i = space required for the i'th relation.

Let t_i = time required for the execution of the i'th query once.

The classical OR formulation is

$$Minimize \sum_i p_i t_i \tag{2}$$

subject to

$$\sum_i x_i m_i \leq M \tag{3}$$

$$t_i = t_i(x_1, x_2, \ldots, x_{128}), 1 \leq i \leq 128$$

$$x_i x_{i+64} = 0, 1 \leq i \leq 64 \tag{4}$$

$$x_i = 1 \text{ or } 0, 1 \leq i \leq 128 \tag{5}$$

We have equation (4), since we can't have both the fractional relation and the corresponding full relation stored simultaneously.

A simple illustration of the formulation may be found in Section 6.4.2.

Computation of m_i and t_i

The general method for UDS is illustrated below. However, for clarity, we restrict UDS to be having only three entities: V, E and F. The possible relations, their index values i, for use in x_i, m_i and t_i and the storage m_i are tabulated below (as discussed in Section 5.3)

Rel.	$V \rightarrow V$	$V \rightarrow E$	$V \rightarrow F$	$E \rightarrow V$	$E \rightarrow F$	$E \rightarrow E$	$F \rightarrow V$	$F \rightarrow E$	$F \rightarrow F$
i	1	2	3	4	5	6	7	8	9
m_i	2E	2E	2E	2E	2E	4E	2E	2E	2E

(6)

Each t_i is a function of $x_1, x_2, \ldots, x_9$. Again for simplicity of exposition, we do not consider the fractional relations $x_{10}, x_{11}, \ldots, x_{18}$.

Let us consider t_2 corresponding to $V \rightarrow$E. It depends on x_2, x_3 and x_8 only (storing the other relations makes no difference).

Let k = time required for one operation (e.g arithmetic comparison). k includes CPU and main memory access time. Note that k, the time for one atomic operation of CPU, is much smaller than the time required for a linear scan of all edges.

In *C* 'language like' code

If ($V \rightarrow E$ is stored explicitly)

$t_2 = O(1) \times k$

else If ($V \rightarrow F$ and $F \rightarrow E$ are stored explicitly)

$t_2 = O(N^2) \times k$

else $t_2 = O(E) \times k$

$$t_2 = x_2 O(1) + (1 - x_2)x_3 x_8 O(N^2) + (1 - x_2)(1 - x_3)(1 - x_8)O(E) \qquad (7)$$

It is thus possible to express each t_i in terms of $x_1, \ldots, x_{128}$.

Solution

Substituting for t_i in equation (2) and m_i in equation (3), with expressions similar to those obtained in equations (6) and (7) and applying Operations Research techniques (e.g. Non-linear programming) yields the *optimal* values, for the $x_i, 1 \leq i \leq 128$.

6.3.3 Graph theoretic approach

Problem formulation

The problem can be formulated as the selection of a sub-graph (sG) from the complete graph (CG) = (N, A) such that

$sG = (sN \mid sN \subseteq N, sA \mid sA \subseteq A)$. N and A denote the set of nodes (e.g. entities E, F) and arcs (e.g. relation $E \rightarrow F$) respectively.

Optimality conditions

For the sG to be valid, its transitive closure $T(sG)$ must be CG.

$T(sG) = CG \Rightarrow$

1. $sN = N$ (Note that sN is the set of nodes of a subgraph sG of the complete graph).

2. $\forall n_i \mid n_i \in N$ must be reachable from all nodes $n_j \mid n_j \in N$.

In other words the graph must be strongly connected.

For the sG to be optimal it must have the least weighted path length. The weights are derived from the query frequencies . We can adopt two courses: either consider all the weights to be constant i.e. optimize for general purpose or for a specific application gather some statistics on the usage pattern (the query frequencies). In the subsequent discussion we opt for the first course, eloborate justification for this having been already furnished in Part 1, at a general level. Part 1 described a CIM model at considerable length and the gist of it is that different queries, not just individual edge or face or vertex based queries, but a good mix of all of these queries arise in such an environment. As each query is involved this is a better indicator of the overall performance of a typical CIM environment where different users query a common data base.

It can be easily shown that (for example, see [Reingold, Nievergelt and Deo 1977])

1. $\mid A \mid = \mid N \mid^2$.

2.(a) There are $2^{|N|^2}$ possible sub-graphs of CG.

(b) CG contains

$$z_i = \begin{pmatrix} n \\ i \end{pmatrix} (i-1)!$$

cycles having i arcs , where $n = \mid N \mid$.

Total number of cycles $= \sum_{i=2}^{n} z_i$

3. The transitive closure of an $mSG = (sG \mid sN = N)$, which has a cycle

involving all the nodes N is the CG. Of all such cycles, Hamiltonian or Eulerian trails are the minimal.

The observation 3 leads to the following necessary conditions for an optimal sub-graph $mSG = (N, SA \mid SA \subset A)$

It must be a minimal sub-graph satisfying the two conditions below

1. Has a *self* Hamiltonian cycle or a *self* Eulerian trail. Note that the qualifier self implies that all the nodes of the complete graph CG are involved.

2. Has no self loops.

Let n^h and n^e be the number of self Hamiltonian and self Eulerian sub graphs of CG.

It can be proved that

$$n^h = (\mid N \mid -1)! \tag{8}$$

$$n^e = \mid N \mid !/2 \tag{9}$$

Thus we drastically reduce the search space for the optimal sub-graph from $2^{|N|^2}$ to $(1 + N/2)(\mid N \mid -1)!$.

Comparison between Eulerian and Hamiltonian sub-graphs

Lemma 1 *Hamiltonian cycle is the cycle with the least path length for connecting a given number of points with directed arcs.*

Proof

The result is intutively obvious. Before elaborating on the proof, it is worth noting the difference between directed and undirected. In directed, to be able to move from A to B, is not the same as the ability to move from B to A. However, in undirected A and B are connected would imply the ability to move from A to B and vice versa. The minimum number of undirected arcs to connect n points is n − 1 (analogous to acyclic chain). The minimum number of directed arcs to connect n points is n (analogous to a closed polygon). Also it is possible to construct a Hamiltonian cycle of directed arcs around n points and the path length is n. Thus of all possible cycles for the CG, Hamiltonian cycles require the minimum number of arcs.

We derive the formulae for the path lengths and the sum of the path lengths for each of the nodes to every node for the Eulerian and Hamiltonian sub-graphs.

(1)Path lengths

It can be proved that for a Hamiltonian sub-graph $HSG(hN, hA)$

$$\mid hA \mid = \mid hN \mid .$$

It can be proved that for a Eulerian sub-graph $ESG(eN, eA)$

$$\mid eA \mid = 2(\mid eN \mid -1).$$

$$\mid eA \mid / \mid hA \mid = 2 - 2/ \mid N \mid \tag{10}$$

For any boundary graph $\mid N \mid$ varies from 3 to 8 (UDS). Thus the ratio of number of relations of a Eulerian to Hamiltonian sub graph varies from 4/3 to 7/4

(2) Total path length

Let q_{ij} = path length from node i to j for $1 \leq i \leq \mid N \mid, 1 \leq j \leq \mid N \mid$,

q_i= Sum of the path lengths for node i from each of the nodes

$= \sum_{j=1}^{\mid N \mid} q_{ij}$ for $1 \leq i \leq \mid N \mid$, and

q = Total of the sum path lengths for all nodes $= \sum_{i=1}^{\mid N \mid} q_i$.

We now derive formulae for q^h, q^e the totals for hamiltonian and eulerian sub graphs respectively.

$$q_{ij}^e = i - j, j < i$$

$$= j - i, j > i$$

$$= 2, j = i$$

$q_i^e = \sum_{j=1}^{i-1}(i-j) + \sum_{j=i+1}^{n}(j-i) + 2$ for $1 \leq i \leq \mid N \mid, \mid N \mid = n$

$$q^e = (n^3 + 5n)/3$$

$$q^h = n^2(n+1)/2$$

$$q^h - q^e = (n^3 + 3n^2 - 10n)/6 \qquad (11)$$

$$= 4, \qquad n = 3$$

$$= 12, \qquad n = 4$$

$$= 44, \qquad n = 6$$

We discuss how the above formulae can guide us to evaluate the Hamiltonian and Eulerian sub-graphs. The number of relations (path length in graph theoretic notation) $\mid A \mid$ in the sub-graph is a crude measure of the storage. The total of the path length

for each of the nodes from each of the nodes is a crude indicator of the total time for all types of possible queries. Thus formulae (10) and (11) are crude indicators of relative storage and time respectively of Eulerian and Hamiltonian boundary data structure schemata.

However note that the number of relations $\mid A \mid$ doesn't constitute an exact criterion for the amount of storage, as some of the relations may require different storage than the rest (e.g. in the WE, $E \rightarrow E$ requires $4E$ storage while all other 8 relations require $2E$ each - see Section 5.3. The best criteria for the storage is thus the storage weighted path length of the graph (not to be confused with the access weights which are query frequency dependent, as discussed in the Sub-section on 'Optimality conditions' above), but to keep the discussion simple the formulae were restricted to crude measures.

6.3.4 Application of the Optimality concepts

We consider application of the above discussion to develop an alternative to the SDS, which is shown to require 25% less storage, in case 1. In cases 2 and 3 we evaluate the WT and the $3D$-SDS, which were shown to be special cases of the UDS in Section 6.2.5.

Case 1: An alternative to the SDS

From (8) the number of possible hamiltonian sub graphs with 3 entities is (3-1)! = 2.

These two data structures are shown in Figure 11. It may be observed that they are both Δ shaped, one can be obtained from the other by reversing the direction of the arcs, hence we refer to the first one as Δ and the other one as reverse Δ. Reverse

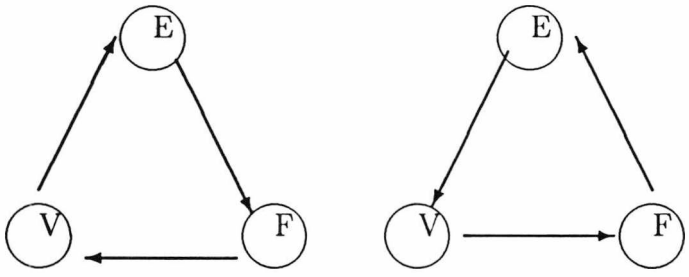

Figure 11: Delta and Reverse Delta

Δ, which stores explicitly $F \rightarrow E, E \rightarrow V, V \rightarrow F$ is a hierarchical representation, each relation, except the last one, being a mapping from an entity to its immediately lower level entity.

However, Δ is more efficient for extension to multiply connected faces [Ala 1992] (see also the next Chapter for an explanation). It can be shown that the remaining 6 relations can be obtained from the three relations stored explicitly in time $O(N)$. Woo and Wolter [1984] proved that the average value of N is less than 6 (see also Section 5.2). Thus clearly all the queries can be answered in nearly constant time as $N \ll$ E.

From equation (9) the number of possible eulerian sub graphs with 3 entities is $3!/2 = 3$. These three data structures are illustrated in Figure 12 . Of these, the SDS [Woo 1985], illustrated in Figure 12(b), is the only one which has a hierarchical structure described before.

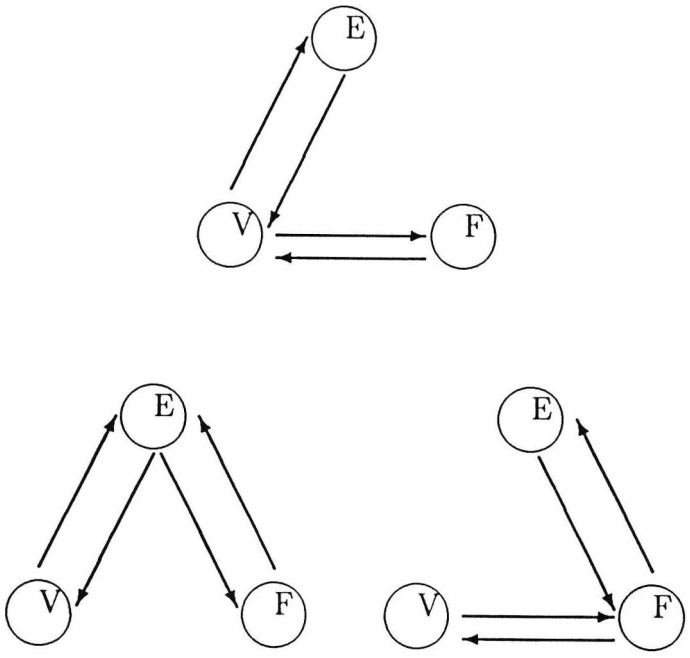

Figure 12: Variants of Symmmetric Data Structure

Comparison between Δ and SDS

From equation (10) we find that the storage ratio for SDS and Δ is 4/3. It is clear that both have constant time access, but Δ requires 75% storage of SDS.

Case 2: Winged Triangle

WT is limited for polyhedra. Neverthless, it is intersesting to compare with WT, since as stated in Section 6.2.5 it requires less storage than WE for solids with polygonal approximation.

Comparison of WT and Δ for space

The storage required for WT is $12(E-F+L)$ (see Paoluzzi et al. [1989]). The storage for Δ which has a total of L loops in its faces is $6E + F + L$ (see [Ala 1992]). Δ is space efficient if

$$6E > 13(F - L) + 2L.$$

In certain situations (e.g. a cube with a square through hole has 10 faces, 2 loops, 24 edges and 16 vertices) Δ may be more space efficient than WT. Substituting $E = 3F$ and $L = 0$ (these expressions are empirical values from study of common engineering design objects by Wilson [1988] - his loop count includes outer loops also and hence his expression $L = F$ is equivalent to $L = 0$, since each face has atleast one loop i.e. the outer loop) in the resulting condition, we observe that Δ is more compact than WT for common engineering objects.

Comparison of WT and Δ for time

In Section 6.4 Δ is shown to be more efficient in access time also.

Case 3: Data structures for *3D-Triangulation*

It has been proved [Preparata and Shamos 1985] that a $3D$ triangulation of n points has $O(n^2)$ vertices and edges. In our case $n = V$, since we are triangulating a body with an initial number of vertices $= V$. Thus after triangulation, T (the number of tetrahedra) and F are both $O(V^2)$, and E is $O(V)$. Thus $3D$ SDS (introduced in Section 6.2.5), i.e. the SDS enhanced for tetrahedrization, will have $O(V^2)$ storage complexity. Since E is $O(V)$ the storage complexity can also be expressed as $O(E^2)$. Thus $3D$-SDS has a quadratic storage complexity, compared to linear space complexity of Δ and SDS.

Extension of Δ to represent the $3D$-Triangulation

Extension of Δ to represent the $3D$-Triangulation will require an extra entity T (a special case of the B enitity of UDS), the tetrahedron. From (8) the number of possible Δ extensions are equal to the number of different Hamiltonian sub graphs with 4 entities i.e. (4-1)! $= 6$. Of these 6 data structures, the schema of storing $T \rightarrow F, F \rightarrow E, E \rightarrow V$ and $V \rightarrow T$ is a hierarchical representation each relation (except the last) being a mapping from an entity to its immediately lower level entity. This is similar to the reverse Δ, and is illustrated in Figure 13. The reason for not choosing the Δ schema is that we don't have multiply connected faces in a $3D$-triangulation (in the next Chapter it is shown that extension to holes is more efficient with Δ rather than the reverse Δ).

It can be shown that the 12 relations, which are not explicitly stored, can be obtained from the four explicitly stored relations. As an example, let us obtain the $e \rightarrow F$ for the edge e. We first obtain the two end-points v_1, v_2 of the edge e by using

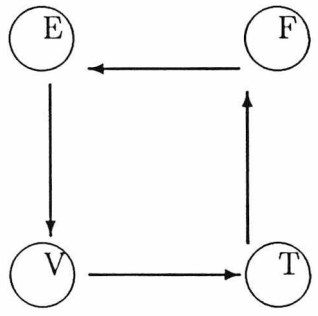

Figure 13: 3-D Delta

the explicitly stored relation $e \rightarrow$V. We then obtain $\{v_1 \rightarrow T\} U \{v_2 \rightarrow T\} = T_e = \{T_1, .., T_k\}$. For each of these $t \in T_e$ we find their faces. We thus have 4 faces per tetrahedron, which may have e as one of their edges. We obtain for each of these faces 3 edges by the explicitly stored relation $F \rightarrow$E. We thus have at most $O(12)$ edge comparisons to make. Thus clearly any query requires constant time as $O(12) \simeq O(1) \ll E$.

Extension of SDS to represent the $3D$-Triangulation

From equation (9) the number of possible SDS extensions are equal to the number of different eulerian sub graphs with 4 entities i.e. $4!/2 = 12$. It may be noted that the $3D$-SDS proposed in [Bruzzone et al. 1989] is one of these 12 possible extensions of the SDS. It has the hierarchical structure described in the preceding paragraph. Extraction of the 10 relations which were not stored explicitly, requires a constant number of operations i.e. $O(1)$ for any constant relation or a number of operations which is linear in the output size i.e. $O(N)$ for any variable relation.

Comparison between Δ and SDS for $3-D$ triangulation

As discussed under Case 3, $T = O(V^2)$ and $F = O(V^2)$. Hence although Δ explicitly stores four relations the storage due to the three relations $V \rightarrow T$, $T \rightarrow F$ and $F \rightarrow E$ is quadratic in V, while that due to the other relation, $E \rightarrow V$ is linear in V. Hence the main storage is because of the 3 quadratic storage relations. Similarly in the SDS extension, it is because of 4 quadratic relations.

It is clear that both have constant time access, but Δ has approximately $3/4rd$ storage of SDS.

6.4 Globally versus Special purpose B-Rep

6.4.1 Special purpose design

Woo suggested two approaches for the design of optimal data structures. The first approach is the design of a general purpose data structure that is globally optimal. This was the approach adopted in the optimization of the UDS in Section 6.3.3. The second approach is the design of a special purpose data structure for a specific set of applications. This approach requires knowledge of p_i (the distribution of the topological queries), which may then be used for problem formulation and optimization, as described in Section 6.3.2.

For example, in the WT, Paoluzzi et al. [1989] state that edges are never used by set operations, only in line drawing do they find some use. Based on this assumption the WT stores $F \rightarrow V$ and $F \rightarrow F$ only. Thus face-based queries can be answered instantaneously, but the other 7 queries require linear time.

In an integrated environment (Computer Integrated Manufacturing, discussed at

length in Part 1) several users will need access [Ala 1992] to information other than $F \rightarrow V$ and $F \rightarrow$F (e.g. in computer vision edge oriented queries are quite common). With WT each of these queries will require linear time, clearly an undesirable situation.

6.4.2 An example

Let us assume that an application requires 8000, 1000, 999 and 1 numbers of queries of $F \rightarrow V, F \rightarrow E, F \rightarrow F$ and $E \rightarrow F$ respectively. This is representative of an application predominantly face based, but also exhibiting the realistic situation of an infinitesimal frequency of 0.01%, of the occurrence of the edge-based queries.

Given the above number of queries let us find the best strategy of storing. Assume all the data resides in main memory, which is limited to 6E.

We can formulate the problem with the restriction of 3 entities only, similar to Section 6.3.2, in which the meaning of x_i, m_i and t_i may be found. Proceeding along the lines of Section 6.3.2

$$\text{minimize } (t_5 + 8000t_7 + 1000t_8 + 999t_9) \tag{12}$$

subject to

$$(m_1x_1 + m_2x_2 + m_3x_3 + m_4x_4 + m_5x_5 + m_6x_6 +$$

$$m_7x_7 + m_8x_8 + m_9x_9) \leq 6E \Leftrightarrow$$

(substituting for m_i from (Eq.6))

$$x_1 + x_2 + x_3 + x_4 + x_5 + 2x_6 + x_7 + x_8 + x_9 \leq 3 \tag{13}$$

$x_i = 1$ or 0 for i = 1..9

In equation (12), ignoring t_5 (in tune with the claim of Paoluzzi et al. [1989]) we obtain the optimal solution as, $x_7 = x_8 = x_9 = 1, x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = 0$

This optimal solution stores the two relations $F \rightarrow V$ and $F \rightarrow F$, of the WT, and an additional relation $F \rightarrow E$, requiring $6E$ total storage. The three face-based queries require negligible time, however, $E \rightarrow F$ requires file inversion, hence t_5 is $O(E)$. Thus total time, which is mainly t_5, is $O(E)$.

To sum up, time for Δ is $O(N)$ which is independent of E (See section 5.2 for N which denotes the average number of topological neighbours) while the time for WT is a multiple of E. As discussed by Bentley [1986] the coefficient of E, even if very small, does make it more expensive than a constant time Δ, in the asymptotic case. WT runs 99.99% in constant time, but is still not better than Δ, even in the event of occurrence of a small probability (1 in 10,000) of a query ($E \rightarrow F$ in the example).

6.4.3 Comparison of graph-theoretic and OR approaches

The above discussion revolved around the global versus special purpose debate which employ graph-theoretic and OR techniques respecitvely. Their relative merits are summarized below. A special purpose optimization *requires knowledge of query frequencies* which involves extensive data gathering and statistical analysis over a wide variety of user applications [Weiler 1985].

In a CIM environment several people will be using the CAD data base. It is only natural that the data base cater to all the users, be it the designers of parts or the users of a robotic cell for recognition of parts. Integrity of the data can best be guaranteed by having a single CAD data base. Having multiple data bases each fine

tuned to individual application can not only play havoc with the data integrity but also requires extensive data gathering (for optimization) over a wide variety of user applications. Hence it can be safely assumed that for CIM a single data base is used and different types of queries arise in practical environment. This is the main justification for choosing general purpose optimization where the optimality function is the sum of the different queries. Graph-theoretic techniques are very useful for the design of such general purpose data structures which are optimal in a overall sense but may perform suboptimally for specific applications. However, with future advances in data abstraction, expert systems and learning systems, OR techniques can be configured to be adaptible to the changing mix of query frequencies yielding a more efficient data structure. At present the modelers and applications are tightly coupled to a data structure (see the first Chapter in Part 4) and it is not possible to experiment with a different data structure without causing an upheaval. However with the advances in data abstraction (e.g. the C++ facilities) it is possible to hide data structure from users (a good example of such a commercial system is [Spatial-Technology 1991]) and change them to suit the current query frequency mix and thus OR techniques could be useful, in future modeling systems.

6.5 Conclusions

A data structure which generalizes boundary data schema was proposed. Examples drawn from currently popular data structures were used to demonstrate the versatility of the proposed Universal data structure (UDS). Methods of optimization of UDS and their application, which lead to the discovery of efficient data structures were also discussed. It was shown that a global optimization approach based on UDS is

superior to a special purpose data structure designed, for a specific set of applications.

6.6 Comments on Δ data structure

In this section we briefly mention additional factors to be taken into account in the design of boundary data strucures and how well the Δ meets them. The Δ data structure has been shown to be a good choice for Bounadry Representation. It was claimed that Δ occupies only 2/3 rds storage of the Symmetric Data Structure. However this estimate was based on the number of entities. When the number of pointers is considered the actual storage values are slightly higher for both the data structures ($8E$ and $10E$ respectively). A detailed discusion may be found in the next Chapter. It is interesting to note that Winged Edge and Winged Triangle have the same storages. However, even in terms of the actual storage, Δ remains the most compact amongst the constant access time data structures exemplified by Winged Edge and Symmetric Data Structures and counterexemplified by Winged Triangle Data Structure. In an integrated environment [Ala and Chamberlain 1991] constant average access time is very desirable and thus clearly Δ is a right candidate because of its compactness. As discussed by Ala [1992] compact storage has an important side benefit. The predecessors to Δ assumed implicitly that the whole of the data resided in main memory. Hence, the only cost was the computation by the CPU and disk access cost does not figure in the optimization criterion.

Modern computers operate in a virtual memory environment which makes it unnecessary for the all data to be simultaneously held in the main memory, and makes it possible to query data bases much larger than the main memory. This however comes at a price i.e. it requires frequent transfer of data between the main memory and the

disk to get the sought after data from the vast disk store and put in main memory.

Since Δ requires the least storage, a large proportion of its data can be held in the main memory. For example, if the main memory is $2E$, a third of Δ 's data can be held in the main memory while only a quarter of the SDS's data can be held. Because a large fraction of data is memory resident, fewer disk transfers are required for Δ and consequently savings in the costly disk operations result. This will be the topic of the next Chapter.

A major drawback of the special purpose optimization is the difficulty in gathering statistics on usage patterns [Weiler 1985]. Weiler [1985] has shown that for a B-Rep data structure to be sufficient in a curved surface environment, storage of $V \rightarrow E$ is adequate. Since Δ is a superset of it, it is also suitable for a curved surface environment. Also Δ is suitable for a non-manifold environment. A comprehensive survey of all non-manifold data structures, and their sufficiency and efficiency is the subject of Part 3. Also in later Chapters, Δ will be shown to be more efficient against many of the mainstream data structures (e.g. [Spatial-Technology 1991, Weiler 1986]).

Chapter 7

Performance Anomalies in B-Reps

7.1 Introduction

Previous analysis of the performance of the data representation have assumed implicitly, the data to be present in the main memory, allowing simple estimates of data access (based on RAM i.e. random access memory time) to be made. However, for access to very large data bases (e.g. a nuclear power plant) it is not likely that all the data can be held in memory and it is more likely that data will be accessed using a paging mechanism in a virtual memory environment. This chapter[1] seeks to estimate the performance of accessing CAD data structures within such a CIM environment, and also presents results of similar performance measures associated with an alternative representation that we have developed [Ala 1991] in the previous chapter. We show that not only this representation has excellent storage efficiency (6E), but also that in a virtual memory environment, access time can be improved.

[1] ©1992 IEEE. Reprinted, with permission with alterations, Computer Graphics and Applications, 12(2), pp.49-58, March, 1992

7.1.1 Organization of the chapter

The organization of the chapter is as follows. In Section 7.2 we discuss the virtual memory and data base environments in the context of CAD. In Section 7.3 we compute the number of record accesses for several constant time data structures. In Section 7.4 we show that the number of the record accesses is a better estimator of the performance of the data structures. In Section 7.5 we show that the extra entities that were introduced in Section 5.3, do not actually meet their avowed objective of lower access time. In Section 7.6 we show that the theoretical estimates of storage can be misleading, followed by summary.

7.2 Virtual Memory and Data Base environments

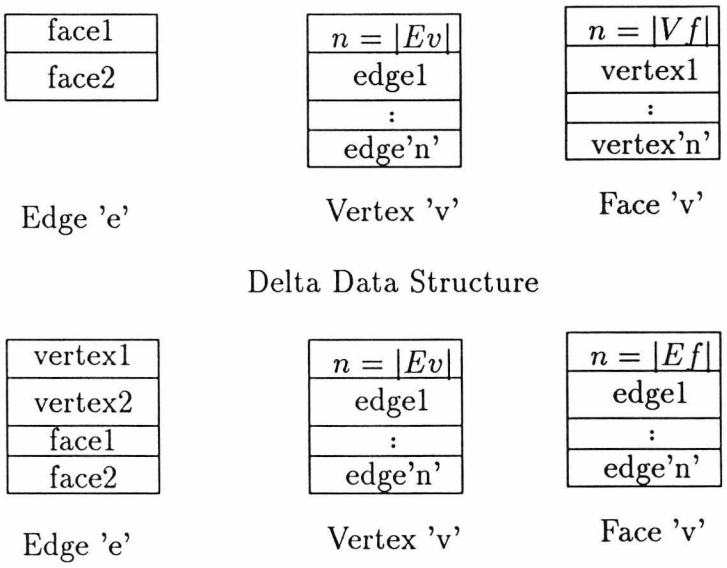

Figure 14: Record Structure

Each of the entities has a record encoding its fields (e.g. see Fig. 14). The fields

can be pointers but we use the simpler integer identifier numbers (see Section 7.6.2) - for example, the face1 field of the 'e' record in Δ data structure contains the identification number of the first adjacent face. Note that the term record signifies a collection of related information all of which tends to lie in the same physical storage unit (i.e. a page) and must be retrieved before the fields can be accessed. Once a record is available in main memory, field comparison cost is negligible in comparison with the disk access cost [Weiler 1985].

7.2.1 Virtual memory reference mechanism

The advent of virtual memory concept, made it possible to run programs much larger than the main memory. Both the main memory and the program instruction and data space, are divided into pages of fixed size (e.g. $4Kb$ in SPARC workstations). To access any item, the page to which the item belongs must be resident in the main memory. The program pages are placed into the physical frames (main memory pages) to make them available for programs requesting data. Requests which can not be satisfied with pages residing in the main memory cause page faults.

7.2.2 Data base reference mechanism

To access any data base record, the disk block to which the record belongs must be resident in the main memory. A data base management system (DBMS) maintains an area in the main memory for data base buffer pool. Slots in this buffer are physical frames into which disk blocks are placed to make them available for programs requesting data. Data requests which can not be satisfied with blocks residing in the buffer cause data base faults.

These are analogous to page faults in virtual memory. Also in an analogous manner DBMS must implement block fetch and replacement policies. Stretching the analogy further, virtual memory maps virtual memory pages to physical frames while DBMS maps the block address space (a block is identified by a number) to the buffer pool slots.

7.2.3 Empirical results

Experiments previously reported in literature, in a virtual memory environment [Coffmann and Varian 1968], have demonstrated that data references induce more paging than instruction references, because of their greater randomness. Also, page turning frequency decreases with the proportion of data and instruction space resident in the main memory.

Some database studies conducted at IBM [Rodriguez-Rossel 1976] have reported strong sequentiality (appearance of sequences of increasing data base block addresses, in a reference string) and weak locality in contrast to virtual memory which exhibits strong locality (references to a subset of the block space). Locality in data bases arises when the data which have been used by one transaction are also used by another. Unlike the virtual memory, very little re-referencing occurs within a given transaction. Blocking several records increases locality [Rodriguez-Rossel 1976] . This is analogous to the increase in the locality (and the consequent reduction in page faults) of virtual memories with dense packing of active storage area [Hatfield 1972].

In the case of virtual memory the working set size exhibits a rapid decline in slope beyond a window size. But the above database studies have reported that working set size continues to be a linear function of window size, thus indicating the absence

of locality. If the buffer management policy uses an LRU (Least Recently Used) policy, which is biased towards locality then the data access time may approach the secondary storage access time.

Sequentiality behavior in data base references can be exploited by prefetching (in a single I/O operation) the next consecutive P blocks in anticipation that they will be referenced in the near future. This policy consumes P buffer slots and incurs extra time to transfer them into main memory. However it is advantageous in high latency storage media.

7.2.4 I/O speeds have lagged far behind the CPU speeds

According to Kearns and DeFazio [1989], while CPU processing times have diminished to 1/20th, disk seek times have halved and transfer times have reduced by a factor of 4. Thus seek time has become more critical than CPU or disk transfer rate, over the years. Field comparison depends solely on CPU and main memory access time, while record access time depends primarily on seek time. Thus over the years record accesses have gained more importance. We work out the record accesses for various data structures in the following Section.

7.3 Determination of Record Access Costs

NOTATION: The notation has already been introduced in section 5.2. We use the following equations derived therein:

$$N_{Ev} = N_{Fv} = N_{Vv} = N_v \tag{1}$$

$$N_{Ef} = N_{Ff} = N_{Vf} = N_f \tag{2}$$

	$v \rightarrow E$	$e \rightarrow F$	$f \rightarrow V$	$v \rightarrow F$	$e \rightarrow V$
Δ	1	1	1	$N_{Ev}+1$	$N_{Fe}+1$
SDS	1	1	$N_{Ef}+1$	$N_{Ev}+1$	1
WE	$N_{Ev}+1$	1	$N_{Vf}+1$	$N_{Fv}+1$	1
HE	$1+3N_v$	$1+2$	$1+N_f$	$1+3N_v$	$1+2$
VE	$1+N_v$	2	$1+2N_f$	$1+N_v$	2
FE	$1+2N_v$	2	$1+N_f$	$1+2N_v$	2

	$f \rightarrow E$	$v \rightarrow V$	$e \rightarrow E$	$f \rightarrow F$
Δ	$N_{Vf}+1$	$N_{Fv}+1+N_{Ev}$	$N_{Fe}+1+N_{Ve}$	$N_{Vf}+1+N_{Ef}$
SDS	1	$N_{Ev}+1$	$N_{Fe}+1$	$N_{Ef}+1$
WE	$N_{Ef}+1$	$N_{Vv}+1$	1	$N_{Ff}+1$
HE	$1+2N_f$	$1+4N_v$	7	$1+3N_f$
VE	$1+2N_f$	$1+N_v$	2	$1+2N_f$
FE	$1+N_f$	$1+2N_v$	2	$1+N_f$

Table 3: Record accesses for diferent data structures

$$N_{Ee} = 4, N_{Fe} = N_{Ve} = 2 \tag{3}$$

We also make use of the inequality deduced therein: $3 \leq N \leq 6$.

For simplicity of exposition, we defer the treatment of extra entity loop. Thus the estimates assume only three basic entities V, E and F and in the case of HE an additional *segment* entitity (collapsing the loop and face entities into one as shown in Fig.4). In Table 3, we compute the number of record accesses for different schema. The example below clarifies the method of computation.

Example: $e \rightarrow V$

In WE, $e \rightarrow V$ requires accessing the record corresponding to a single edge record while in HE, we need to access first $e \rightarrow S$ to get the two half edges $s1$ and $s2$ of the edge e and then access two more records $s1 \rightarrow V$ and $s2 \rightarrow$V. Thus we incur two extra record accesses by adopting HE instead of the WE for the relation $e \rightarrow$V.

In Table 4 , we find the range of record accesses, simplifying the figures in Table 3,

		Δ	WE	SDS	HE	VE	FE
N	3	33	27	23	67	39	39
	6	51	45	35	115	66	66

Table 4: Sum of record accesses for different data structures

using the equalities in Eqs. 1, 2, and 3 and summing record accesses for all the 9 queries . We also assume $N_v = N_f = N$ for comparison purpose.

The estimates of Weiler [1985] for VE, FE (see Section 5.3 for a brief description) and WE are per adjacency element (and hence need multiplication with the appropriate N and increment of unity for initial conditions) except for $e \rightarrow F$, $e \rightarrow V$ and $e \rightarrow E$. The minimum number of record accesses occurs for SDS ranging from $23 - 35$. The worst case occurs for HE which is nearly three times the SDS record cost. In Table 5, we find the the minimum record accesses, for $3 \leq N \leq 6$, in each storage class starting from $4E$ up to 20E. It also gives example data schema for each class (e.g. the $6E$ class requires a minimum number of 51 record accesses and stores the first three relations in Table 2, i.e. the relations $F \rightarrow V$, $V \rightarrow E$ and $E \rightarrow F$). With storage limit of 4E, it is possible to store at most two relations (e.g. $F \rightarrow V$ and $V \rightarrow E$). Since the minimum number of relations to connect 3 entities is 3, queries other than the two explicitly stored relations (e.g. $E \rightarrow F$) require file inversion i.e a sequential scan of the entire file which has $O(E)$ entries. This is shown by the E in Table 5.

Δ Gives Best Return on Storage

Figure 15 reveals that Δ (storage $6E$) has the maximum reduction in number of

Storage		$4E$	$6E$	$8E$	$10E$	$12E$	$14E$	$16E$	$20E$
i		$1-2$	$1-3$	$2-5$	$1-5$	$1-6$	$1-7$	1-8	$1-9$
N	3	E	33	23	20	17	14	11	9
	6	E	51	35	29	23	17	11	9

Table 5: Storage Class vs. number of record accesses

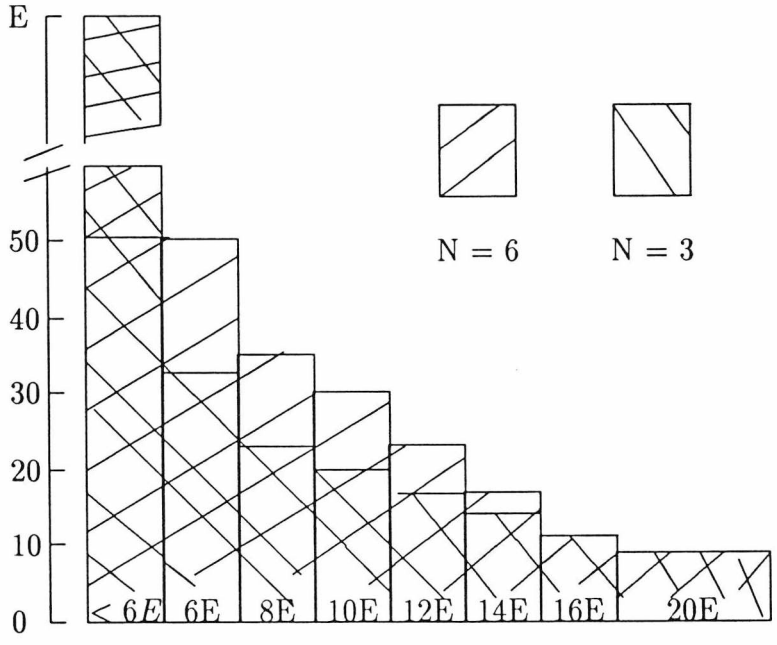

Figure 15: Reduction in Number of Record Accesses

record accesses per unit storage. Thus it is the elbow of the storage-time curve. The next best change occurs for the SDS (8E). The table exhibits the law of diminishing returns from $6E$ onwards. Of interest is the $20E$ class. This is the maximum possible storage. We term it as the Generalized Data Structure (GDS). GDS has the minimum number of record accesses.

7.4 Comparison of data schema which have constant time

Comparison with the data structure storing all possible relations and hence requiring the maximum storage of $20E$, but possessing the least record accesses, reveals that Δ is more efficient in time also. It is because Δ requires only 30% of storage, can hold very large fraction of its total data space in the main memory, and hence has fewer page faults. This will be discussed in great detail in Section 7.4.3.

It may be noted that the SPARC workstation, which we have used for conducting experiments may not be representative of all computers, but in general, the same broad conclusions may be reached on any computer operating in a virtual memory environment.

7.4.1 The criteria for optimality

Time for the execution of a query depends on the number of field comparisons, number of disk accesses, main memory access time, CPU processing speed and the disk access (mainly seek) time. The number of disk accesses is dependent on the number of records referenced.

Let p = ratio of the number of faults (termed as page faults in the virtual memory and data base faults in the data base environment) and total number of records

referenced. In other words, p is the proportion of records requiring disk accesses.

Let f_n, r_n = Number of field comparisons and record accesses respectively.

Note: Some records needed to be examined may already be present in the main memory. The number of page faults is only $p \times r_n$ (each fault induces one disk access).

Let f_t, r_t = Time for each field comparison (main memory access and CPU processing) and fault service time (mainly disk access time - see Section 7.2.4) respectively.

A general formula is given below.

$$T = f_n \times f_t + p \times r_n \times r_t \tag{1}$$

T =Time for execution of all the 9 possible queries, one time each.

We take the cost to be the sum of all 9 individual queries, the justification has already been established in Part 1, at a general level. Part 1 described a CIM model at considerable length and the gist of it is that different queries, not just individual edge or face or vertex based queries, but a good mix of all of these queries arise in such an environment. As each query is involved this is a better indicator of the overall performance of a typical CIM environment where different users query a common data base.

Note: If we assume that all the data is resident in the main memory, then $p = 0$, we have, $T = f_n \times f_t$. This is the implicit assumption in most of the earlier attempts (e.g. [Kalay 1989]) to enhance the access efficiency of the WE.

7.4.2 Condition for record access cost to dwarf the field access costs

If the fault rate is very low then the predominant cost is that due to the field comparison, which was considered to be a measure of the speed in [Weiler 1985]. We find the range of p for which record access cost dominates field comparison cost. Amongst the constant time data structures, listed in Table 4, Δ requires the maximum number of field comparisons. We have estimated the number of field comparisons to be 250 and the number of record accesses to be 51 (see Table 4) for Δ. Hence the worst case requires 250 main memory accesses and CPU comparisons. Taking $f_n = 250, r_n = 50$, we conducted the experiments on a SPARC station, r_t and f_t were estimated to be $30ms$ and $3\mu s$ respectively. Substituting these in (1), we have,

$$T = 250 \times 3\mu s + p \times 50 \times 30ms \tag{2}$$

Thus the ratio of field comparison cost and the disk access cost is 1/2000p. We conclude that if the page fault percentage is greater than 1%, the cost of field comparisons is less than 5% of disk access cost and hence can be ignored. This will be substantiated by the experimental data below. As discussed in Section 7.2, the page fault rate depends on the amount of main memory and the size of the data.

The SPARC station we used had a Main Memory of $8Mb.$ (some of which is occupied by the UNIX system and hence is unavailable to programs). We varied the data size from $8Mb.$ to $26Mb.$ as shown in Table 6 . Thus x, which is the ratio of number of physical frames available (i.e. the main memory available for the program) and the total number of virtual memory pages occupied by the data (i.e. the data size) was varied from 0.87 to 0.27. We measured the corresponding page faults and

Mb	8	12	16	18	20	22	24	25	26
Time	2	60	840	1110	1290	1460	1620	1690	1760
p	0	0.013	0.26	0.34	0.40	0.45	0.49	0.51	0.53
x	0.87	0.58	0.44	0.39	0.35	0.32	0.29	0.28	0.27

Table 6: Experimental data (Data space in Mb, access Time in Secs)

T which are tabulated in Table 6 .

When the data structure occupied $8Mb.$ most of the data (i.e. 87%) resided in the main memory and hence there were negligible number of disk accesses (i.e. $p \simeq 0$). Thus the T value shown against x of 0.87 is mostly due to field comparisons.

However when the data size was increased to $12Mb.$ only 58% of the data could be held in the main memory and this lead to a large number of disk accesses. Although there were only 1.3% page faults, the access time rose by a factor of 30, as shown in Table 6. The difference in T for x of 0.87 and 0.58 is due the extra disk accesses. So the determining factor is the number of record accesses for $p > 1\%$. Thus the experiments corroborate the statement that followed eqn.(2).

7.4.3 Comparison of Δ with GDS

The number of page faults decrease with the fraction of total space (mainly data space since instruction space is neglible in our case) held in the main memory. Other secondary factors (e.g. page size, page replacement policy) influencing the page fault rate are beyond the scope of this thesis. Thus the number of disk accesses decrease with x, which is the proportion of the data size held in the main memory. Since GDS and Δ require $20E$ and $6E$ storage respectively (see Table 5), the proportion for GDS, x_g, is $6E/20E = 0.3$, times that of x_Δ of Δ.

$$x_g = 0.3x_\Delta \tag{3}$$

From Table 6 we observe that as x decreases by a factor of 3 (e.g. from 0.87 to 0.29), access time increases by a factor of 800. Hence it is likely that GDS will have substantially higher faults (for the same main memory), resulting in higher virtual memory overhead than for Δ. Let T_Δ and T_g refer to T and p_Δ and p_g refer to p in Δ and GDS respectively (p and T were defined in Section 7.4.1).

As remarked in the previous section, f_n for Δ was estimated to be 250 and accordingly we took 250 in (2). To show that the following discussion applies even when the actual value of f_n is 10 times our estimate, we take 10×250 in equation (2), to yield

$T_\Delta = 2500 \times 3\mu s + p_\Delta \times 50 \times 30ms.$

Since GDS stores all the information explicitly, it requires no field comparisons and from Table 5 , it accesses 9 (we take $r_n = 10$ in equation 1, for simplicity) records. Thus

$$T_g = p_g \times 10 \times 30ms$$

Thus the condition for Δ to be more efficient than GDS is that

$$p_g > 5 \times p_\Delta + 0.025 \tag{4}$$

If $p(x)$ describes the fault rate curve then from equation (3), we have $p_g = p(x_g) = p(0.3x_\Delta)$. We find the solution of $p(x)$ curve (x refers to x_Δ) which satisfies $p_g = 5 \times p_\Delta + 0.025$ or equivalently

$$p(0.3x) = 5 \times p(x) + 0.025$$

to be

$$p(x) = 0.025(x^{\ln 5/\ln 0.3} - 1)/4 \text{for} 0 < x < 1 \tag{5}$$

This curve represents the lower bound for the region, in which Δ is more efficient than GDS. As shown in Fig. 16, the experimental curve lies above this curve and GDS is inefficient compared to Δ. However, for certain intervals (e.g. $x_g > 1$) GDS is more efficient than the Δ. In the next section, we find the interval for x, for which this is true.

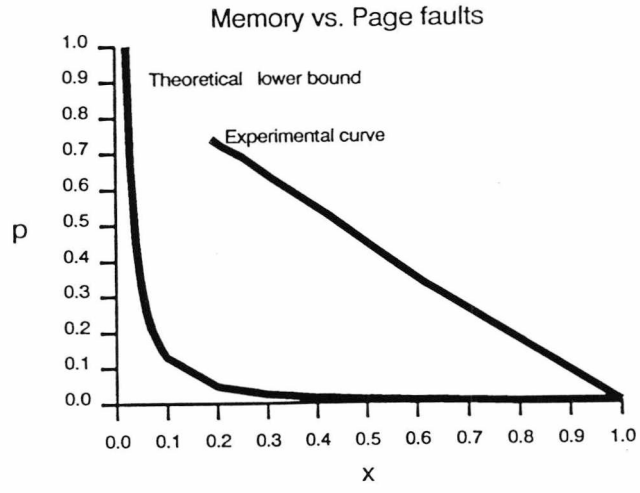

Figure 16: x (Ratio of main memory and data size) Vs. p (proportion of records requiring disk access)

Note: If we took f_n to be 250, we have

$$p(x) = 0.0025(x^{\ln 5/\ln 0.3} - 1)/4 \text{ for } 0 < x < 1$$

which also leads to the same conclusion that followed eqn.(5).

Condition for memory size for which Δ is more efficient than GDS

It is interesting to note that though the number of relations stored varies from 3 (in Δ) to 9 (in GDS) the total number of records stored is

$$V + E + F = 2E + 2 \cong 2E$$

This holds for SDS, WE, Δ and GDS (but not for the Vertex- Edge, Face-Edge and Half-Edge Data Structures). The number of records of Δ and GDS that can be held in the main memory, of size M, are $(M/6E) \times 2E = M/3$ and $(M/20E) \times 2E = M/10$ respectively.

Assume that neither locality nor sequentiality is present.

$$p_\Delta = (2E - M/3)/2E = \begin{cases} (1 - M/6E) & \text{if } 0 \leq M/6E < 1 \\ 0 & \text{if } M/6E \geq 1 \end{cases}$$

$$p_g = (2E - M/10)/2E = \begin{cases} (1 - M/20E) & \text{if } 0 \leq M/20E < 1 \\ 0 & \text{if } M/20E \geq 1 \end{cases}$$

Since $x_g = M/20E$, we have (we refer x_g to be x) ,

$$p_g - 5 \times p_\Delta = \begin{cases} 47x/3 - 4 & \text{if } 0 < x < 0.3 \\ 1 - x & \text{else if } 0.3 \leq x < 1 \\ 0 & \text{else if } x \geq 1 \end{cases}$$

Maximum of $(p_g - 5 \times p_\Delta) = 0.7$, which occurs at $x = 0.3$. From Equation (4),

$$p_g > (5 \times p_\Delta + 0.025) \text{ for } 0 < x < 0.3 \Leftrightarrow x > 0.257$$

$$p_g > (5 \times p_\Delta + 0.025) \text{ for } 0.3 \leq x < 1 \Leftrightarrow 1 - x > 0.025 \Leftrightarrow x < 0.975$$

The condition for Δ to be more efficient than GDS is

$$0.257 < x_g < 0.975$$

which applies for the case when there is no locality or sequentiality. However, in the presence of locality or sequentiality p_Δ and p_g are expected to be lower.

7.4.4 Comparison of Δ with SDS

Proceeding as above, we find that Δ is more efficient than SDS for

$$0.474 < x_{SDS} < 0.999$$

.

7.5 Multiple Entities

We show that it is inefficient to include extra topological and geometric entities. We have 3 basic topological entities E, F and V. As mentioned in Section 5.3 several additional entities resulted from attempts to extend and enhance the efficiency of the Winged Edge.

We also examine the effect of multiple geometries. Each of the three basic entities have an associated geometry. It is not necessary to store the geometry of all the three entities: the geometry of any two entities can be derived from the geometry of the third entity.

7.5.1 Multiple Topology

We discuss how data structures can be extended to faces with holes without resorting to the extra loop entity and the half edge record's effect in increasing the record

accesses.

Half Edge entity

We compare HE (which was created by exploding the edge enity of WE) for storage and time with respect to WE. Excluding the loop entity, HE differs from WE in only two additional arcs $S \rightarrow E$ and $E \rightarrow S$.

Effect on storage Extra storage because of the bifurcation of edge entity into two half edges is given by

$$E \rightarrow S + S \rightarrow E = 2E + 2E = 4E$$

Note: $|S| = 2E$ as each edge has 2 half edges.

Effect on access time From Table 4, the extra record accesses of HE over the WE are 40 (70 when $N = 6$). We therefore conclude that extra entities decrease the main memory accesses and comparisons done by the CPU but increase storage and record accesses substantially. Proceedings along the lines of Section 7.4.3 a comparison of WE and HE (storages of $9E$ and $13E$, number of records of HE = 2E+2E = 4E, f_n of 10 and 100 and r_n of 67 and 27 for HE and WE) reveals that WE is more efficient for $0 < x_{HE} < 0.99999$.

Multiply connected faces

We discuss two important earlier attempts for extension of data schema to represent holes and an alternative scheme, which avoids the usage of extra entities like the loop, yet possessing the storage and time efficiency.

Bridge - Edge representation (BE)

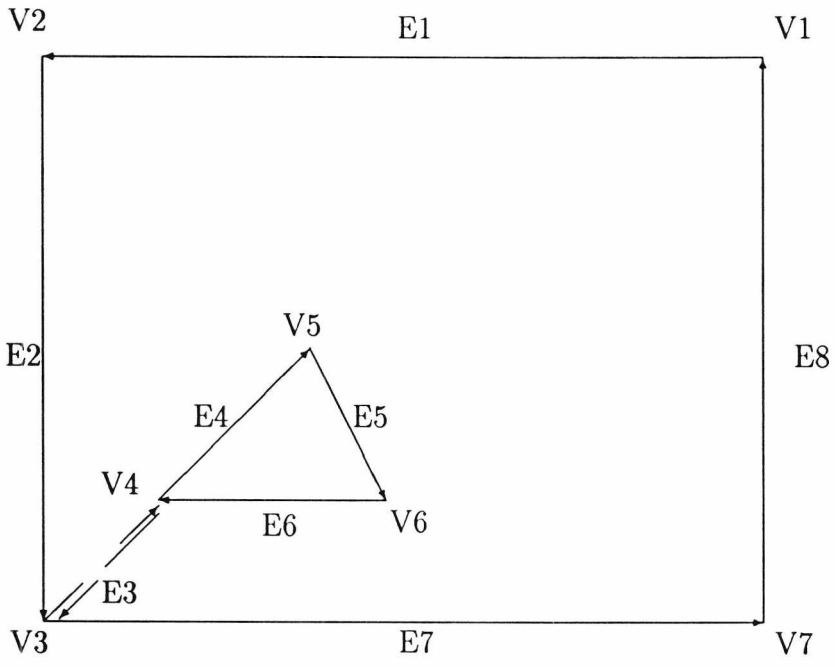

Figure 17: Extension of Δ for multiply connected faces

Holes are implicitly represented by fictitious edges (termed as Bridge-edges) connecting loops of multiply connected faces. Because the fictitious edges ($E3$ in Figure 17) occur twice in visiting a face, i.e. the face is described by

$$E1, E2, E3, E4, E5, E6, E3, E7, E8,$$

it is ambiguous to determine the next edge if only an edge (e.g. $E3$) is given. The same holds true if vertices are used for representing holes, i.e. the face is described by

$$V1, V2, V3, V4, V5, V6, V4, V3, V7.$$

Hence the bridge edge [Yamaguchi and Tokieda 1985] uses edge and the next vertex in alternation, i.e. the face is described by

$$V1E1, V2E2, V3E3, V4E4, V5E5, V6E6, V4E3, V3E7, V7E8.$$

In this scheme a pair of vertex and edge is unique. Thus the topological position of a vertex or an edge can be uniquely specified by VE pair not by V or E alone. The scheme involves extra storage because of the necessity to represent both edge and vertex in a face list. The extra storage

$$\text{for } L \text{ holes} = 2E + 2L \times 2 = 2E + 4\text{L} \tag{1}$$

Add to this the storage due to the fictitious edges (1 per loop) which is $8L$ in the case of WE.

Loops

The holes can be explicitly represented (e.g. [Weiler 1985]) by an additional entity called a loop. A face is a list of loops. Let L_i refer to the number of loops in the i'th face and L to the total number of loops for the whole object. Note that L excludes the outer loop. The extra storage in the case of SDS is because of $L \rightarrow F$ and $F \rightarrow L$ and is given by

$$\sum_F 2(L_i + 1) = 2L + 2F \tag{2}$$

in the case of SDS.

An alternative scheme to represent faces with holes

The face in Figure 17 is represented by

$$V1, V2, V3, -V4, V5, V6, V4, -V3, V7.$$

We represent holes by a string of vertices, the sense of the string being opposite to the outer contour. Each hole is connected to the outer boundary by two vertices one of which has a negative sign, to symbolize that it is a fictitious edge (e.g. $V3 - V4$ and $V4 - V3$). To get edges we take two consecutive vertices. If the second vertex is

positive then it constitutes a valid edge (e.g. $V1V2$), otherwise an artificial edge (e.g. $V3-V4$). Also the members (denoted by the vertex IDs) of the vertex list are unique because each number in the vertex string may appear twice but with opposite signs. This alternative scheme has the best of both BE and loop schemes viz. uniqueness of face representation and least storage.

Comparison with respect to storage

The extra storage incurred is

$$\sum_F (L_i + 1) = \begin{cases} L + F & \text{if } L > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Comparing this with (1), there is no penalty if $L = 0$. But in the bridge-edge and the loop schemes even in the absence of holes, the extra storage cost is $2E$ and $2F$ respectively.

Comparison with respect to time

Note that there is no increase in the number of record accesses. But more comparisons are required by the CPU because of the need to test the sign of a vertex in the list of vertices for faces. This extra test for the sign is however required for processing $F \rightarrow E$ only. It is not required in others like $E \rightarrow$V.

Extra record access in BE and loop schemes: The BE extension involves no extra record accesses. Let F_L = 1+ Number of holes in face F. SDS with Loop scheme involves an extra cost of F_L each for $F \rightarrow V, V \rightarrow F$ and $F \rightarrow E, 1$ for $E \rightarrow F$, and $2F_L$ for $F \rightarrow F$, a total of $5F_L + 1$. Thus even when there were no holes in a face the extra record accesses are $5 \times 1 + 1 = 6$. Thus in the alternative scheme you pay for what you use i.e. if no inner loops then we do not need any

loops at all, not even outer. Also all the vertices of a face are maintained at one contiguous location and hence the record accesses are lower than with the explicit representation of loops. We therefore conclude that the alternative scheme is more efficient for extension to holes. All the algorithm implementations of this work use this alternative representation. Note that this scheme can not be used in the case of SDS and the reverse Δ, because they do not store $F \rightarrow V$.

7.5.2 Multiple Geometry

We assume polyhedral models only and the criteria to be the access time of the geometry of all three entities. We investigate two schemes: storing only the vertex coordinates and storing all (e.g. WE implementation in [Baumgart 1975]) the geometry i.e. the vertex coordinates, line and plane equations for the edge and face respectively.

Storage estimation

Schemes 1 and 2 require $3V$ and $3V + 4E + 3F \simeq 7E$ respectively. Thus storage of all geometry requires approximately 4 times the storage.

Time estimation

Scheme 1 Computation of line equation requires retrieval of one geometric record for each of the end points. Computation of face plane requires access to 3 geometric records to retrieve coordinates of three vertices. For simplicity we do not consider the topological record accesses. Substituting 6 (including 1 for the vertex geometry) for r_n and 100 for f_n in eqn. 1 of Section 7.4.1

$$T_1 = 100 \times 3\mu s + p_1 \times 6 \times 30ms$$

where

$$p_1 = (V - M/3)/V \simeq 1 - M/2E$$

Scheme 2 f_n is zero since no computation is involved, $r_n = 3$ for accesing three geometric entities. Substituting these , we have,

$$T_2 = p_2 \times 3 \times 30ms$$

where

$$p_2 = (2E - M/3.5)/2E = 1 - M/7E = 1 - x_2$$

Thus for $0.167 < x_2 < 0.997$ storing only the vertex coordinates is more efficient in time also. Note that consideration of topological record accesses (one for an edge to get its end points and 7 for a face to get its vertex identifiers, in the case of WE, as seen from Table 3) slightly tilts the scale in favour of schema 2. The same happens as value of f_n is raised from the assumed value of 100, as discussed in the following paragraph.

Sensitivity to numerical computation In the above discussion we arbitrarily assigned a value of 100 for f_n which is a measure of the numerical computation. For polyhedral domain the value of 100 for f_n is quite adequate, for curved surfaces the value ought to be substantially higher. The range for progressively higher values of f_n, may be computed as below.

$$0.167 < x_2 < 0.997 \text{ for } f_n = 100$$

$$0.172 < x_2 < 0.967 \text{ for } f_n = 1000$$

$$0.222 < x_2 < 0.667 \text{ for } f_n = 10000$$

Thus it appears that the results are not very significant for the curved geometry because the numerical computation overwhelms the virtual memory overhead. However as discussed in Section 7.2.4, the advances in disk performance are lagging far behind CPU, relegating numerical computation to a minor role. Thus the future trend and relevance of our work for curved surface geometry is unclear. Further work is required to investigate the performance when the whole model including curved surface geometry is involved.

7.6 Methods of Implementation

The other anomaly shows that theoretical estimates of the storage are misleading (e.g. the Symmetric Data Structure requires more storage than the winged edge in practice) because some data structures do not permit implementation with arrays. Several different methods for implementing the data structures are surveyed for their effect on storage.

7.6.1 Linked lists and Arrays

We show that though SDS takes $8E$, a practical implementation requires 10E. Because of the variable nature of N_F, N_V an array implementation is not possible for SDS. However WE can be implemented with arrays because

$$N_{Ve} = N_{Fe} = 2, N_{Ee} = 4 \text{ and } V \rightarrow E \text{ and } F \rightarrow E \text{ store 1 edge each.}$$

Array implementation of WE requires 9E. But the SDS requires linked (we assume singly linked) list implementation which increases the theoretical storage by $4E$, as

calculated below. Storage for $SDS = F \rightarrow E + E \rightarrow F + E \rightarrow V + V \rightarrow E$

$$= 2E + 2E + 2E + 2E + 2E + 2E = 12E$$

Storage for $\Delta = F \rightarrow V + E \rightarrow F + V \rightarrow E$

$$= 2E + 2E + 2E + 2E + 2E = 10E$$

The actual storage for SDS tallies with the implementation in [De Floriani and Falcidieno 1988]. Also note that linked lists may cause the data to be non-contiguous unlike the array implementation, where the data will be contiguous.

Winged Triangle [Paoluzzi et al. 1989] is an example of a recently published data structure which permits array implementation. However, as discussed in [Ala 1991], Winged triangle representation does not belong to the class of constant time data structures and hence is not used for comparative studies in this Chapter.

7.6.2 Dynamic allocation of memory

We already described this scheme in Section 7.2.2 and Figure 14. As the data is being read we can dynamically (e.g. C language) increase the storage for variable relations like $F \rightarrow$. The net increase is the storage for the pointers which point to pointers to each face and vertex. Storage increase $= F + V \simeq E$ for the pointers. Assuming that a pointer occupies the same memory as an integer the net increase is E, for a total of $7E$. We must also store the number of vertices in a face as the first element in the face array and similarly the number of edges adjacent to a vertex as a first element in the vertex array. These changes will make storage of Δ and SDS to be $8E$ and $10E$ respectively.

7.6.3 Relational Implementation

Kalay [1983] argued for an auxiliary relational data structuring for non-manipulative operations. We limit our discussion for space and time estimation for pure relational implementation of the three basic entities. As a relational data base requires tuples of constant length (called degree of the relation in the relational parlance), we need 2 relations or tables to store each of the variable relations like $V \rightarrow$E.

If the ordering of the edges around a vertex is not important one table suffices. Also if a system defined ordering (clockwise order of the edges) of the tuples can be maintained with a suitable operator for retrieving the clockwise neighbour from a given edge for a given vertex, then one table with 2 attributes suffices. This table (schema 1) will have two attributes vertex_id and edge_id, the key being the concatenation of the two attributes.

The two schema for $V \rightarrow E$ are

Schema (1)

Relation Vertex_edge

(vertex_id: integer,

edge_id: integer)

Schema (2)

Relation Vertex_1st_edge

(vertex_id : integer,

1st_edge_id : integer)

Relation Vertex_next_edge

(vertex_id : integer,

edge_id : integer,

next_edge_id : integer)

Schema 1 requires $4E$ storage, while schema 2 requires $6E + 2$V. Similarly $F \rightarrow V$ requires $4E$ or $6E + 2F$ depending on whether Schema 1 or 2 is employed. For the constant relations $E \rightarrow V$ and $E \rightarrow F$, one table suffices (each requires $2E$ storage).

Thus relational implementation of Δ will require $10E$ under schema 1 and $14E + 2F + 2V \simeq 16E$ (since $F + V = E + 2$) under schema 2 . The corresponding figures for SDS are $12E$ and $16E + 2F + 2V \simeq 18E$. Thus relational implemenatation involves significant increase in the storage.

Queries on relational data bases require frequent join operations which require a great deal of data retrieval from the permanent store with the consequent increase in number of page faults and time .

7.7 Conclusions

We considered several data schema which exhibit constant access time . They reveal space and time anomalies in a virtual memory environment. A study of the data structure which stores all the possible relations reveals a startling phenomenon: more storage does not necessarily mean less time, because of the virtual memory overhead. On the other hand a new data structure has the minimum storage amongst all the constant time data structures, yet requires less access time.

It calls for a rethinking of the data structure space vs. time phenomenon. An immediate consequence is that many popular data structures which are modifications of the Winged Edge Data Structure incur extra storage but do not actually meet their avowed objective of reducing the access time. It argues for compact data structures not only for less storage but also for lower access time. We conclude that compactness

has beneficial side-effect: it reduces the interrogation time by having fewer page faults and clustering the related data. In a nutshell 'Small is beautiful'.

Part III

Non-Manifold Boundary data structures

Many of the ideas presented here are mere extensions of those in the previous part. However there are few additional considerations in non-manifold, which form the crux of this part. Like the manifold part we start this part with an introduction, a survey and design of several data structures before introducing the analysis. Design is inherently iterative and overlaps with analysis. Data structures are no exception to this fundamental nature of design and analysis.

Chapter 8

An introduction to non-manifold modeling

Non-manifold modeling is rapidly gaining prominence because of its huge potential. This Chapter serves to elaborate on this and acts as a prime motivator for non-manifold modeling.

8.1 Non-manifold configurations

In non-manifold objects, two or more objects may share a vertex (e.g. 18 shared by the tetrahedron and cube, in Figure 18) or an edge (e.g. 9-10 shared by the two upper cubes, in Figure 18), or a face such as in cellular objects (e.g. 1-2-3-4, in Figure 18). the three situations referred to jointly by regular non-manifold conditions (i.e. R-sets). Also there may exist single vertices in a face (e.g. 22 in face 1-2-6-5, in Figure 18) or as stand alone shells (e.g. 23, in Figure 18), wire like objects (e.g. pipe work) with no adjacent faces (e.g. 25-8, in Figure 18), sheets or laminae i.e. paper like with no solidity (e.g. 5-6-24, in Figure 18), and internal partitions such as

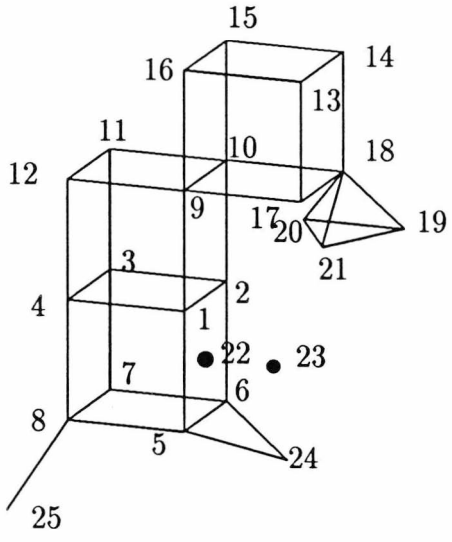

Figure 18: Non-Manifold conditions

in composite materials (e.g. 1-2-3-4, in Figure 18). We make a distinction between internal partitions (or structures) and non-manifold face: the former involves different materials on the two sides of the separation face and is accordingly classified as an irregular non-manifold condition.

Scope in the context of boundary data representations is given by the tree in the Figure 19. Each of the wire, sheet and solid may further be qualified, as shown in Figure 20. For a partially surfaced wire and a full (or pure) wire, we have $0 \leq N_{Fe} \leq 2$ and $N_{Fe} = 0$ respectively. They are exemplified by 25-8-5 and 25-8, in Figure 18, respectively. Wire is a profile if it lies on a single plane, such as the cross section of a solid, wire frame otherwise. It may be open, closed (i.e. loop like) and unbounded in one direction (i.e. semi-bounded) or in both directions (i.e. unbounded). Stroud [1990] distinguishes sheet objects lying on a single surface and several surfaces as lamina and shell objects respectively.

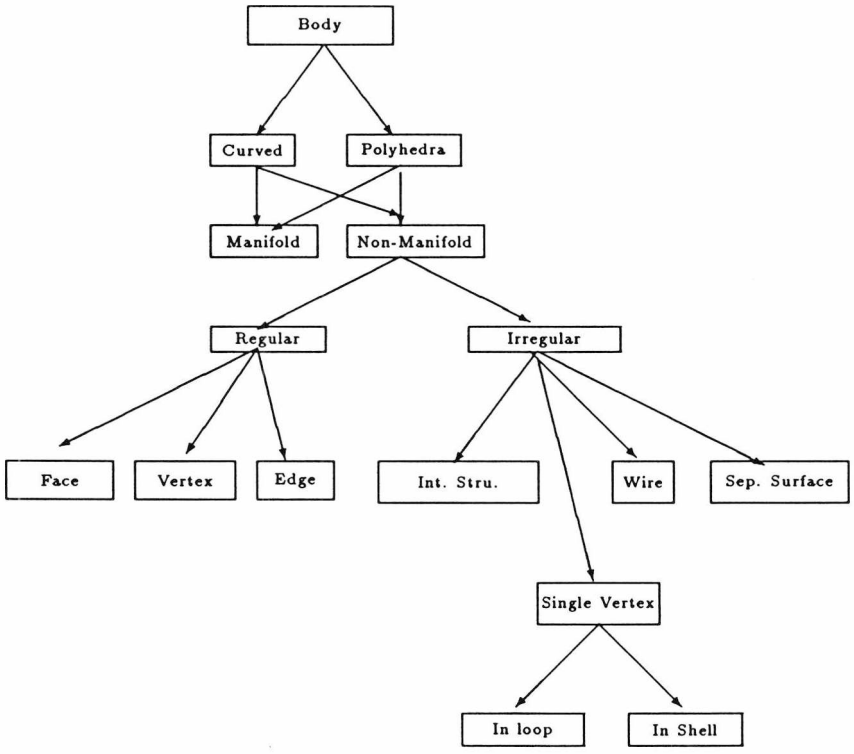

Figure 19: Taxonomy of Boundary Representations

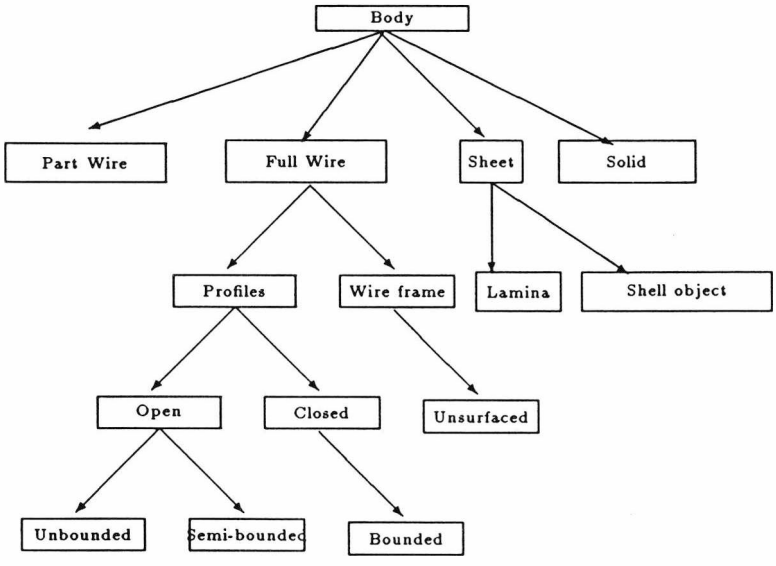

Figure 20: Body Tree

8.2 Advantages of non-manifold B-Reps

Traditional solid modeling presupposes the objects to be 2-manifolds . It was justified on the grounds that non-manifold objects are not manufacturable. However Boolean operations on manifold objects can result in non-manifolds and although the final model is a manifold, intermediate results can be non-manifolds.

In addition, non-manifold modeling offers several exciting possibilities. Non-manifold modeling reflects the actual design process [Stroud 1990]: start with a sketch, flesh it out with sheet and volume as necessary. It possesses unrestrained modeling domain (ranging from cabling to an aircraft). For example, it enables the coexistence of the representation of intrinsically wire-like objects and solid models such as pipe work and its conversion to and from solid models. Also editing is facilitated by allowing for representation of overlapping regions and hence several authors (e.g. [Kawabe, Shimada and Masuda 1989, Masuda, Shimada, Numao and Kawabe 1989, Crocker and Reinke 1991]) adopted non-manifold data representation in their implementations. Arbitrary reworking demands usage of hybrid modeler (B-Rep and a CSG tree like operation history) which is possible with a non-manifold only. All booleans are readily available through appropriate marks on the constituent volumes, unlike the conventional B-Rep where the volume for a single Boolean is retained. Another advantage of non-manifold modeling is the support for design by features, which is the topic of the next Section, by allowing for representation of cellular decompositions [Pratt 1988].

8.3 An introduction to Form features

8.3.1 Evolution

Traditional geometric modeling concerned with the representation of pure geometry without regard to functionality. But the designer usually works in terms of features (a circular through hole rather than as a cylinder of a given radius and height, for example, the widget in Figure 31, has two bored holes, a slot and a protruding boss). Because of the modeling system's preoccupation with the geometry, the designer's intent was lost during the modeling stage. Such a strait jacket approach was adequate in the past since the primary objective was to get the model into a computer representation. The data structure needs augmentation for preserving the designer's intention which is very valuable for the downstream manufacturing activities. This demands a flexibility and speed for the user to enable him to experiment with various features and hence fast response is essential. Pioneered by various researchers design by features is now an active research area. For an account of the earlier work see [Pratt and Wilson 1985]. Most of the commercial modelers are attempting to provide feature based modelers (e.g. Parametric Technology's Design Engineer [Potter 1992, Porter 1991]).

Some background on the features and few definitions by Pratt [1990] are helpful in gaining greater insight.

Features may be associated with

- Faces, which may be unconnected or incomplete, e.g. A block protruding from the face of another block. Usually a face participates in at most one feature, so the number of features is far less than the number of faces,

- 1 or more edges, e.g. bevelled side,
- 1 or more vertices, e.g. chamfered corner.

8.3.2 Generation of form feature information

There are three general methods for generation of form feature information.

1. *A-posteriori feature recognition*, uses rules like convex and concave edges, similar to line junction labeling in vision, for component identification
2. Recognition during the model creation stage.
3. Design by features, enables capture of designer's intent but also requires extensive geometric tests.

8.3.3 Representation of form feature information

There are two schemes, an implicit representation which is concise, or an explicit but verbose representation. In the CSG, a feature may not necessarily correspond to a primitive, while in B-Rep, it is usually defined by a collection of faces. Neither of them is a good choice, they provide too much or too little information. So Pratt [1990] recommends a hybrid approach (Note: it is not CSG + B-Rep, although the implicit representation is some what analogous to CSG but is more closer to the parametric representation). The implicit representation is the ideal version and known by Canonical feature volume (CFV). The explicit representation, which is the closed volume in the B-Rep form, has been termed as Attached Feature Volume (AFV). It may require closure faces and hence non-manifold cellular boundary representation was recommended by Pratt [1988].

In B-Reps there are two options to represent features: either to represent as a collection of faces or as closed solids. We often need extra faces to close the feature shells into solids. One disadvantage of a volume based approach is the need for additional entity (e.g. volume closure faces) and the need to distinguish features lying on the exterior from those on the interior. Feature interactions and their deletion are simpler with the volume approach and the partial face problem will not arise.

8.3.4 Comments

Feature technology is rapidly replacing flat geometric modellers. For an overview see [Shah 1991]. In this thesis we do not concern ourselves with the design and analysis of CAD data structures for feature technology. The interested reader is referred to [Pratt 1987], which gives an indepth analysis and design of data structures for feature manipulation. In Part 4, we exploit the spatial orientations of the features to devise a fast model manipulation algorithm.

8.4 Conclusions

The representation being at the heart of any modeler has an all pervasive effect (such as the satellite dependent applications of model) and a great deal of thought and deliberation is essential before a suitable representation is chosen and thus the value of this work is clear. Pioneered by the visionary Weiler [1986], non-manifold modeling made rapid strides. This thesis reports on the progress made since. The next Chapter compares and contrasts the various representations proposed. It serves as a comprehensive reference and as a survey of recent developments. The coverage is extensive (nearly 20 representations) and broad based (includes commercial modelers

and evolving standards). The subsequent two Chapters cover the design and analysis of various non-manifold boundary data structures.

Chapter 9

A survey of Non-manifold B-Reps

9.1 Current Data Structures

We have surveyed a variety of data structures which are identified by their first author or the organization wherever appropriate, as listed below in alphabetical order.

1. Spatial-Technology [1991]
2. Ala [1992]
3. Schelechtendahl [1988]
4. Crocker and Reinke [1991]
5. Dobkin and Laszlo [1987]
6. Gursoz, Choi and Prinz [1990]
7. Hanrahan [1985]
8. Hoffmann [1989]

9. Karasik [1988]

10. Laidlaw, Trumbore and Hughes [1986]

11. Luo and Lukacs [1991]

12. Masuda et al. [1989]

13. Murabata and Higashi [1990]

14. Stroud [1990]

15. Weiler [1988]

16. Wu [1989]

17. Yamaguchi et al. [1991]

Some of the data structures are not well documented in the literature, and also we were unable to elicit any information in some cases. So the description below in some cases interpolates and thus may depart from the original author's intentions. The purpose of the survey is to introduce the original authors terminology and highlight the difficulty of specifying such disparate ideas, in a coherent manner, a task which is undertaken in the next chapter. Many of the terms are self-explanatory, however, the authors use the same term with subtle difference in meaning which will be brought into light in the next chapter. Due to space limits, we only highlight the salient features of each represenatation. For a fuller understanding, the reader should consult the next chapter also.

9.2 A Survey of Data Structures

9.2.1 ACIS

Isolated vertices are represented through degenerate edges (i.e. both the end points same). Non-manifold vertices have one edge stored per volume. Both Stroud and ACIS use a separate entity wire and a shell may contain a mixture of faces and unembedded edges connected together as in wire. In contrast to Karasik's representation faces may be in several disjoint portions but can be considered connected.

9.2.2 Ala

The Δ data structure has been described in the previous Part. As such it is limited to regular non-manifold conditions only. Its extension to a full non-manifold domain will be described in the next Chapter.

9.2.3 CAD*I

A feature special to this is the ability to represent composite materials (such as aircraft). Allows faces to be shared by two internal shells (but only one peripheral shell). Like Karasik a loop is a list of edges and isolated vertices. Allows curved bodies (both ends of a closed curve refer to the same vertex). Compactness is a primary consideration for communicability and hence has no backpointers.

9.2.4 Dobkin

One of the earlier extensions of the Winged edge to model polyhedral cell complexes in 3-D and surfaces of the 4-D polyhedra. The facet-edge considers polygon-edge pair

as an atom and considers the polygon and edge rings to which it belongs. It connects two polyhedra and two vertices. The half edge is similar to Weiler's edge-use (see the sub-section on Weiler below): employs one radial pointer and one loop pointer. Similar to Hanrahan in concepts, but stores each facet-edge's vertex instead of the two vertices of an edge.

9.2.5 Gursoz

Vertex based instead of the conventional edge. Cusp C denotes the usage of an edge and a vertex as a pair in the loop cycle, may or may not have an associated edge and face side. Dangling edge, vertex and face do not introduce a zone. However a set of faces enclosing a vertex introduce a new zone. Hierarchical representation of local vertex, zone and disk is similar to the global model, region and shell. Both walls of a face are oriented but an isolated vertex and a wire edge have an unoriented wall each referencing them (instead of the usual reference to a face) and the mates of their associated loop and cusp are set to null. In addition a cusp associated with an isolated vertex has its edge orientation set to null. The edge orientation T, has a tail vertex field and $T \rightarrow C$ represents the radial cycle of face sides around the common edge. The shell can contain wire frame and open surfaces if they are connected. ACIS has a separate wire entity to represent the same. Weiler's radial edge requires computation for some vertex neighborhood information.

9.2.6 Hanrahan

The data structure resembles a great deal with that of Dobkin, described above. Differs from Dobkin in the support structure only: stores the two vertices of an edge

instead of one vertex of the facet-edge.

9.2.7 Hoffmann

Uses a combination of Δ and reverse Δ data structures. $e \rightarrow F$ is a radial order of the faces and pairs of the adjacent faces represent volume enclosing pairs, all in a single data structure. *Irredundant* geometric data (coefficients of face plane) are used to mitigate the problems of the numerical uncertainty. Because boolean operations do not create any new surface geometry, no new geometric data is constructed, with the choice of face plane geometry. Since boolean operations generate new edges, adoption of edge geometry implies an update in the geometry of the resultant model, the update is subject to the vagaries of the machine precision. Instead with the adoption of face plane geometry, all such new edges implicitly derive their geometry as the intersection geometry of the two adjacent face planes. Similar to many others the relative direction of an edge and face loop are also encoded. Faces can be disconnected unlike Karasik and many others, however, such faces must belong to only one shell. Uses cycles and directed edges, similar to Karasik. Permits infinite volumes but finite surface area, as it helps to treat union operation as an intersection and several complementation steps.

9.2.8 Karasik

Karasik's star-edge, Hanrahan's face-edge and Dobkin's facet-edge are based on the interaction of edges with faces but the latter two require that adjacent cells intersect at least an edge and hence can not represent non-manifold vertex. Note that, for manifold object each edge gives rise to two directed-edges (DE) and each vertex has

two directed edges per neighboring face. A face is defined in terms of a list bounding half edge loops and isolated vertices if any. $E \rightarrow DE$ is ordered by the orientation of the owning face-side of the DE (similar to Weiler's radial order of faces around an edge). The directed edges of a face are ordered radially around the incident vertices on that face. The two relations $V \rightarrow DE$ and $V \rightarrow F$ are sorted by the face.

9.2.9 Laidlaw

The data structure is meant for polyhedral surfaces and faces need a preprocessing step into convex polygons, which will increase the number of faces quadratically, as discussed by Karasik [1988]. With dangling edges and faces it is not possible to distinguish between the interior and exterior and hence are disallowed. However regular non-manifolds are allowed. Similar to ACIS bounding boxes for each face and object are stored which help quickly eliminate non-overlapping faces and polygons. Each of the vertex's status (inside, outside, boundary) with respect to the other object is maintained. $v \rightarrow V$ helps to traverse the edges of the object to find connected regions of vertices with the same status. Geometric information consists of face planar coefficients as well as vertex coordinates.

9.2.10 Luo

Extended the concept of multiply connected faces to non-manifold vertex and edge neighborhoods. Each of the cell complexes have an entity coupling it with an immediate neighbor. An edge and a vertex have a wedge and a bundle for each of the adjacent volumes.

9.2.11 Masuda

Based on a rigorous definition of model domain which states that cell complexes are better suited than R-Sets for non-manifolds (to be discussed in section 12.2.1). Because of the mathematical characterization of the domain there are subtle differences: to represent a face and an edge piercing through it, the edge needs to be split into two with the intermediate vertex at the point of intersection. Also we can not have infinite and semi-infinite edges and infinite and semi-infinite faces (i.e. a face unbounded in one side, which is supported in the [Spatial-Technology 1991]). Distinguishes between empty spaces and material filled volumes e.g. an empty cube does not have an associated volume or region entity. An interesting point: we can represent both an empty and a full cube, the former is a collection of 2-D cells while the later also includes a 3-D cell (i.e. a volume). In the conventional B-Reps (including Radial edge), they have identical representations since both an empty and a full cube have the same boundary. A face with isolated vertices and embedded edges are represented in the same way as a face with holes or a region with cavities.

9.2.12 Murabata

For uniqueness in sheets, both sides of a face O, i.e. $F \rightarrow O$ is stored but for solids, one of the two sides of O, say O_2 is set to null. Thus for manifolds $|O| = |F|$ but $F \rightarrow O$ requires $2F$ storage. In surface state, geometry of the sides O_1 and O_2 are obtained through $O \rightarrow F \rightarrow$Geometry, in solid state geometry is obtained directly. Each half-edge H has a start vertex field and each copy of a muliply connected vertex, has a pointer to one of the H's, the others being obtainable by the next(mate(H)). The global hierarchy of *Shell*, F, V, E, O are called boundary entities and the rest

are local and are known as cycle entities.

9.2.13 Stroud

Uses the same data structure as Braid [1980] which is derived from Winged Edge by inclusion of Loop entity and using only two instead of four of the edge pointer links. Stroud also employs two additional entities for Sheet and Wire bodies. Non-manifold vertices are disallowed and non-manifold edges are represented by an adhoc mechanism: multiple copies of the non-manifold edges (each copy points to two unique faces). This has a drawback in that to obtain all the adjacent faces and edges of non-manifold edges requires error prone and cumbersome geometrical tests (instead of the elegant topological checks) but is useful for treating sheet objects as degenerate solids (and hence the same Euler operators can be used for them). Sheets differ from volumes in that they have one face group corresponding to each side unlike one face group (i.e. shell) for each shell of a volume object. Unlike sheets which are treated as Eulerian wires are treated non-eulerian to allow for flexibility and avoid the overhead of the Eulerian operators.

9.2.14 Weiler

An early proponent of non-manifold modeling. Weiler discovered the first non-manifold data structure termed the Radial edge, because it is based on the radial order of faces around a non-manifold edge. From Section 5.3, recall that the edge was split into two halves, each half partaking in the description of one face only. Weiler extended this basic concept to encompass all the entities. Each of the entities have an associated *use* corresponding to its participation in the immediately higher level

entity (e.g. the edge-use is analogous to the the half-edge which finds use in one of the two neighboring faces of an edge). The four basic entities V, E, L, and F, in the order of increasing level, gave rise to vertex-use (vu), edge-use(eu), loop-use(lu) and face-use(fu) respectively. The entity is not oriented but its use is. In Gursoz loop has a mate loop but here L has no mate. However, each L has two mating lu.

The degenerate case like single vertex and wire edge are treated differently. For an isolated vertex in a face and a shell replace $vu \rightarrow eu$ by $vu \rightarrow lu$ and $vu \rightarrow su$ respectively. For isolated vertex in a loop replace $lu \rightarrow eu$ by $lu \rightarrow vu$. Wire edges have back pointers to the shell instead of a loop and wire shells have down pointers to their edge-uses. An edge may have the same vertex at both the ends. However unlike ACIS, no two vertices can exist at the same geometric location. Mate of an edge-use of a wire edge points to the other edge-use at the other end of the edge instead of the other loop.

9.2.15 Wu

Can represent single vertex in a face and non-manifold edge and vertex. Edges around a vertex relative to a shell and also loops around an edge can be cyclically ordered. Can represent open or closed faces and loops.

9.2.16 Yamaguchi

The basic data structure is similar to Gursoz. Instead of using linked lists for loop, disk and radial cycles uses three mate pointers of the cusp. In addition to the Cusp mate, a cusp on the same face but opposite side O and vertex and another cusp on the same volume and edge but opposite face and vertex is also used.

9.3 Comments

Some of the data structures maintain a pseudo-hierarchy between the entities and are termed as hierarchical exemplified by the Radial edge of Weiler [1986] and counter-exemplified by the tri-cyclic data structure of Gursoz et al. [1990]. Karasik [1988] classified them into representations for cell complexes (e.g. Dobkin and Hanarahan), non- manifold solids (e.g. Laidlaw). Their design and analysis are the topics of the two following Chapters.

Chapter 10

Design Methodology of Non-Manifold B-Reps

10.1 Approach

Ala [1991] proposed a generalization of the Manifold Boundary data structures and the design of boundary data structure is posed as the selection from the combinatoriallly possible data structures. We extend the same to achieve a concise description of a multitude of non-manifold representations. Explosion of the interrelationships amongst the four fundamental (physical) entities (i.e. cell complexes: $0-D$ Vertex, $1-D$ Edge, $2-D$ Face and $3-D$ Region) enables us to identify all the combinatorially possible entities, as explained below. In this chapter, we also give some analytical estimates of the storage values for non-manifold domain, which are very useful for analysis of various data structures, as discussed in the next chapter. Finally we show how to extend the domain of a data structure by illustration of the Δ for various non-manifold conditions.

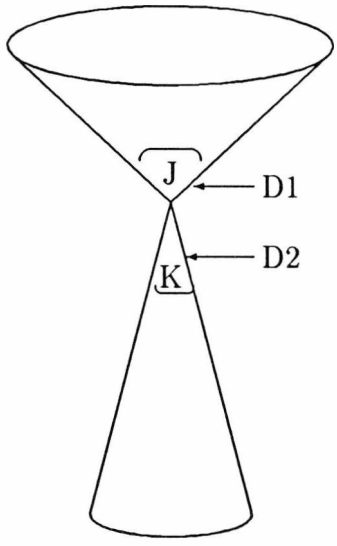

Figure 21: Illustration of disk

10.2 Entities, Relationships and Notation

A cell complex is bounded by lower level cell complexes but does not include the boundaries (e.g. an edge includes every point up to but not its end points). These four basic entities have an associated geometry unlike their derivatives (e.g. Weiler [1986] *uses* introduced in the previous chapter).

The basic relations are between these 4 entities i.e. a complete graph with 4 nodes and additional entities are derivable from these basic entities. However each of the cell complex participates in more than one other cell complex (not necessarily higher level). A familiar example from the manifolds is an edge belonging to two faces. To ensure 1-to-1 mapping, intermediate entities are essential between each pair of primary entities (e.g. a half edge participates in 1 face only). Thus it is possible to have $4_{C_2} = 6$ intermediate bridges. In manifolds only the half-edge suffices. However in general topology a vertex participates in more than one region (excluding the infinite exterior region) and hence the need for a zone to bridge them and a disk

to represent the boundary (e.g. in Figure 21, the common vertex has two zones J and K bounded by the disks $D1$ and $D2$ respectively), similarly an edge occurs in more than one region and hence the need for a Wedge to bridge them. A vertex and an edge have neighborhoods of an infinitesimal sphere (e.g. in Figure 21, the common vertex is the centre of the two truncated spheres) and cylinder centered at them respectively. Each of the connected spaces in the neighborhood is a zone and a wedge respectively. A vertex also participates in more than one edge and hence the need for an intermediate apex (denoted by A) to bridge them.

Some of these bridges are between entities differing in one level i.e. hierarchical and are termed as uses by Weiler. Thus the uses of Weiler are a subset of the general bridges. These also have the property of belonging to the boundary category, being hierarchical (e.g. loop). The others belong to the neighborhood category (e.g. wedge and zone).

Multiply connected regions and faces can be represented by either decomposing them into their constituent connected components or resorting to shell and loop respectively. The second option has the advantage of making available the boundary and is common in the data structures surveyed. Each face has two sides (or used by two shells, in Weiler's terminology) i.e. orientations O and $O1$.

Similarly loop, shell and disk are the boundaries used twice by face, region and zone respectively. We denote the main loop associated with F by $L0$ and its two derivatives by L and $L1$ which are associated with the two face orientations O and $O1$ respectively.

Analogous to the Face an edge has two orientations (i.e. $e \rightarrow F$ can be two sets) denoted by $H0$. Also an edge partakes in the description of both the sides of a face: H and $H1$ (collectively referred to by $H2$) in O and $O1$ respectively. Half-edge and loop

have mates only when associated with a face. A further variation is possible on how one defines the associated mate of $H2$: in Gursoz and Yamaguchi's representations the mate is on the other side of the face and has a common vertex but different edge, in contrast to the conventional mate which has the same edge but opposite vertex and face side.

To represent disconnected volumes a higher level entity model is required. The potential number of entities, relations and data structures are 11, $11^2 = 121$ and $121_{C_2} + .. + 121_{C_{120}} \simeq 2^{121}$ respectively. Although a vast number of data structures have been proposed, they constitute a minuscule proportion of the potential number. Manifold objects and non-manifold regular objects have an identifiable inside and outside and hence it is enough to store one use, say O. However general non-manifolds have to store both the uses. In regularized non-manifolds it is enough to store the usage of an edge by one side of the face, say $H1$ and the region bridges only because they deal with regular sets. A further classification is based on cyclic order between two entities: Disk, loop and Radial Cycles. In manifolds the Radial cycle is implicit because there are only two faces sharing an edge. Weiler catered for Loop and Radial cycles only, while Yamaguchi and Gursoz catered for all three cycles.

10.3 Notation and Diagrammatic conventions

We consider *non-manifold vertex* and *edge* and *sheet* conditions in detail below. One of the most useful contributions of non-manifold modeling is the facility to model *solids with internal faces* i.e. the *non-manifold face* condition. This capability is acheivable in any data structure, by storing the two shells sharing a face and as such is a simple extension. Non-manifold vertex and edge involve more significant

conceptual diffrences and hence form the primary focus in this section. However, the next Chapter identifies the data structures which cater for the non-manifold face.

We use diagrams abstracted (condensed) from the connectivity to show the entities and the relationships which are explicitly stored. In the diagrams, the first letter in the name of each entity identifies the widely differing entities - a suffix (a digit 0, 1, or 2) discriminates the closely resembling entities with fine nuances. A lower case letter stands for an instance of an entity. In later sections depending on the context, an upper case letter for an entity may also mean the cardinality of the set.

For uniformity (in all the data structures the levels at and above the shell are simple extensions with no conceptual difference) and to avoid clutter the diagrams do not show shell and the levels above it. Also we don't show the numbers on the arcs - the number of topological neighbors for an entity is evident from a closer examination of the semantics of the entities and the arcs pairing them. It usually assumes three values: 1 (e.g. pointer to parent or mate which is denoted by a circular arc or the start or terminal vertex of a half edge), 2 (e.g. the two vertices of an edge), and variable (e.g. a loop may have an arbitrary number of edges). Only when there is a departure from these implicit rules do we indicate it on the vector (e.g. in ACIS each vertex has only one edge per disk explicitly stored). Authors have used two techniques for implementing a variable (e.g. $|l \rightarrow H|$) relation: variable record where L has a pointer to a linked list of H's as in CAD*I, and a fixed record where each loop has a pointer to an arbitrary half edge and each H has a pointer to the next H in the list of H's for L, as in Karasik. Again authors have used both a singly linked (e.g. Karasik) and doubly linked (e.g. Weiler). For uniformity of comparison, we assume a singly linked variable record scheme for all the data structures in the subsequent analysis on size and accessibility, in the next chapter. The various data structures

are shown in Figure 22. Circular arcs default to mate pointers whose implicit value is 1, however, an explicit *1* signifies a pointer other than the mate (usually the next element in the cycle of a higher level entity), an explicit 2 signifies storage of both the mate and the next element in loop (e.g. see Karasik, in Figure 22), and a higher value if more pointers are stored (e.g. see Weiler, in Figure 22). Depending on the context a circular arc may represent the storage of mate as well as the next element in a cyclic list of zero or one or more higher level entities (e.g. the circular arc on $H2$, in Gursoz's representation, denotes storage of a mate while that in Yamaguchi's representation indicates the next Half-Edges in the loop, edge and disk cycles).

For brevity L and $L1$, O and $O1$, D and $D1$, and Z and $Z1$ are collectively referenced by $L2$, $O2$, $D2$ and $Z2$ respectively. Karasik and Wu have used three entities in storing a relation (e.g. Karasik stored faces interspersed with the incident half edges on that face, for each vertex, i.e. $v \rightarrow < (f_1, H_1, ..), (f_2, H_{1,}..).. >$). The diagrammatic convention is self explanatory.

10.4 Storage estimates for non-manifold objects

As discussed in Section 5.3, Woo [1985] obtained the storage values for a manifold environment: $E \rightarrow E$ relation requires $4E$ while each other requires $2E$ storage. These formula are elegant for assessing the manifold data structures but do not hold good for non-manifold and curved objects except the three relations $V \rightarrow E$, $V \rightarrow V$ and $E \rightarrow V$ which hold good for linear non-manifold objects. It may be possible to obtain the storage values for the other relations but one needs to define the face (whether it can have 2 outward normals, can consist of single vertex etc.). Also the class of bodies (e.g. whether to allow wireframe) allowable is to be decided. The

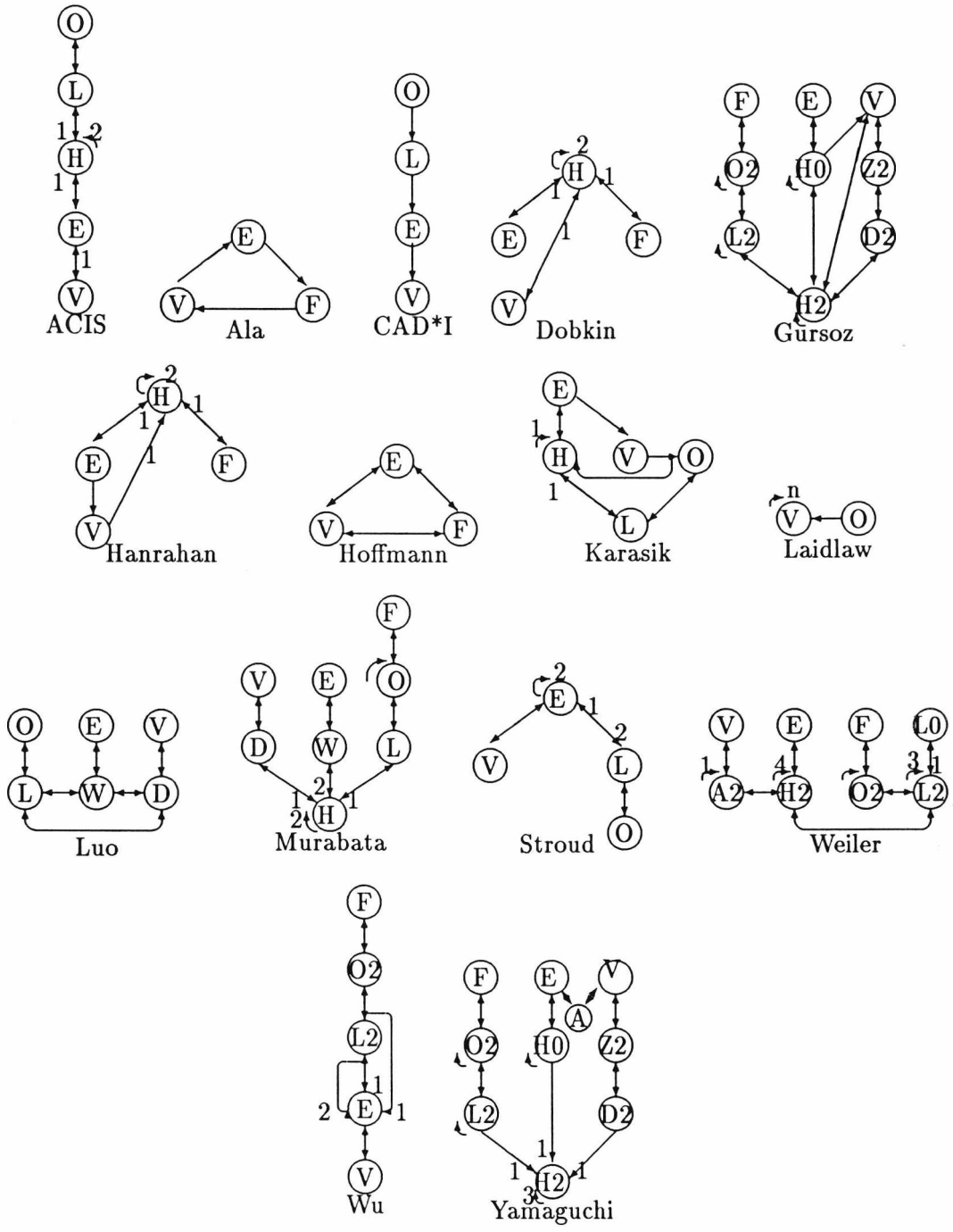

Figure 22: Various Representations (For Crocker and Masuda see Weiler)

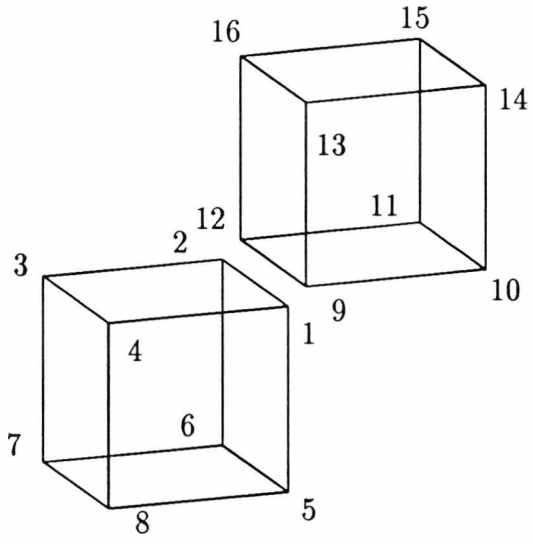

Figure 23: Two cubes with a non-manifold edge duplicated

expressions are likely to involve a large number of variables like the number of isolated vertices and whether they occur in a face or as singe shells.

10.4.1 Storage estimates for R-sets

Restricting to regularised non-manifolds (i.e. R-sets) the manifold formulae appear to be approximate. We consider the three basic entities V, E and F. Let E_t be the total number of edges with the bodies separated from each other i.e. in manifold state (e.g. the two cubes are in manifold state in Figure 23 and non-manifold state in Figure 24, $E_t = 24$, $E = 23$). We show that $V \rightarrow V$, $V \rightarrow E$ and $E \rightarrow V$ require $2E$ storage each while the rest (except $F \rightarrow F$ and $E \rightarrow E$) require $2E_t$ storage each. For $F \rightarrow F$ and $E \rightarrow E$ we can not give such a formula, but fortunately they are stored by none of the data structures surveyed.

The *proof* is easy to see: the manifold formulae hold good for any number of non-overlapping bodies (e.g. the two cubes in Figure 23), for bodies touching at vertices

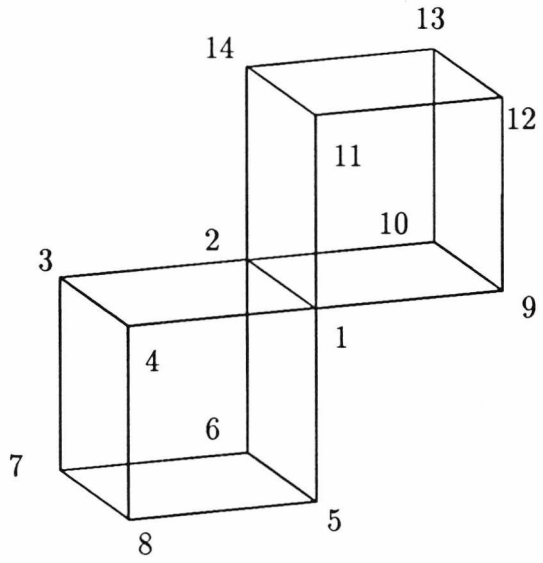

Figure 24: Two cubes with a non-manifold edge

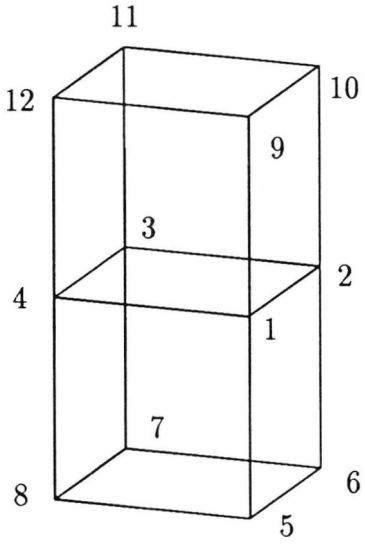

Figure 25: Two cubes with an internal separation face

and edges (e.g. the two cubes in Figure 24) the number of edges and vertices are likely to be lower than if they were separated (e.g. $E = 23$ and $V = 14$, in Figure 24). The overlapping vertices and edges increase the number of neighbors for the vertices, edges and faces adjacent to the overlapping vertices and edges but no more than the total for separate representation in which each multiply connected vertex and edge needs to be duplicated. Thus the total number of neighbors $\leq 2E_t$. A formal proof based on induction on the number of bodies B, whose value is 2 in Figure 24, can be given. Let us consder $E \rightarrow F$. When $B = 1$, $E_t = E$, and the formula holds because, we have only one body which is a manifold. Let us suppose that the formula holds when there are B bodies touching each other. i.e. the storage is $2E_t^B$. Now addition of a $B + 1$ th body causes the $E \rightarrow F$ to increase by $2e_t$, where e_t is the number of edges of the new body. The total stoarge is $2E_t^B + 2e_t = 2(E_t^B + e_t) = 2E_t^{B+1}$, and thus the formula applies to $B + 1$ bodies also. Hence by induction the result is established. We can similarly prove the other storgae results, we stated above.

The above discussion applies to non-manifold vertex and edge. For objects in non-manifold face condition i.e. one complete face is common between two objects, as shown in Figure 25, the formulae are diiferent. For example, $F \rightarrow E$ and $F \rightarrow V$, require a storage of $2E + |F_s \rightarrow E|$ each, where F_s stands for all the separation faces. However, all the formulae are approximately same as those of manifolds, because E_t and E, do not differ siginificantly (e.g. 1 in Figures 23, 24).

10.4.2 Storage estimates for other non-manifold conditions

We now consider the other non-manifold conditions: a single vertex in a face or a shell and wire bodies. Recall that B-Reps were explicit encodings in contrast to CSG. A

natural extension of this philosophy for non-manifolds leads to two options: to include extra entities and not to include them. For example, an isolated vertex can either be represented explicitly by a separate entity or implicitly by the same vertex entity but with an extra type field to indicate its special status. We examine both the options with regard to storage.

Free standing Wire has no faces and shells. The data structures differ with respect to the wire implementation. Some authors (e.g. Stroud, ACIS) have used an extra entity to deal with wire situations. This scheme is shown to be more compact compared to implicit representation. A wire has two entities (E and V) and four topological relationships between these two entities.

Explicit representation of wire For the graph to be connected we need at least two arcs. The number of such pairs are $4_{C_2} = 6$. The data structure chosen must possess a cycle, there is only one possible cycle with two entities and the cycle corresponds to $E \rightarrow V$ and $V \rightarrow E$. If E_W is the number of wire edges each relation requires $2E_W$ storage for a total of $4E_W$ storage. We also need back pointer to the wire which can be $E \rightarrow wire$ or $V \rightarrow wire$. Since $V_W > E_W > V_W/2$, the former requires less storage. Similarly we prefer wire's down pointer to E rather than V. The net saving on storage on account of these two choices is $2(V_W - E_W) < V_W/2$. The total storage for this scheme is $6E_W$.

Implicit representation of wire Several authors use single uniform edge and half-edge records (by the usage of variable record) to represent wire edges in addition to the default solid edges. We analyze Weiler's schema. Wire shells point to half edges and each shell has a pointer to indicate the type. Wire half-edges point to vertex-use, instead of the default loop-use. The storage for implicit representation is

Entity	S	E	H	V	A	*Total*
Storage	$S + E_W$	E_W	$2E + 4 \times 2E_W$	V	$4 \times 2V$	$S + 2E + 10E_W + 8V$

Comparing with the explicit representation the scheme incurs an extra storage of $S + 2E + 4E_W + 8V > 14E_W$. The difference can be large depending on the value of E. The explicit scheme is independent of E and hence incurs no penalty when the body is a not a wire frame. However, in the explicit representation scheme there is a need for maintaining the connectvity information with the solid vertices and edges, discussed in the next section. The storage saving then is $S + H + 4E_W$.

Single vertex shells can be efficiently represented as explicit entities analogous to the explicit representation of wires. The only topology required of a single vertex shell is to its shell. Storage saving by adopting an explicit representation rather than using a single variable vertex record is $> S + 4E$. Note that single vertex do not have a neighboring shell, otherwise they belong to the single vertex loop category.

Single vertex loops can be efficiently represented as explicit entities analogous to the explicit representation of wires. [Karasik 1989] is an example of this scheme (face is represented as a list of half-edge loops and isolated vertices). The only topology required of a single vertex loop is its loop. The above discussion will be crystallized into the extension of Δ's extension for various non-manifold conditions, in the next section.

10.5 Design of Δ for sheet and wire conditions

Non-manifold edges can be handled elegantly because Δ stores $E \rightarrow F$ which is necessary for the radial order of faces around an edge. According to Weiler [1986] for encoding *wire frame* association of shells, $E \rightarrow E$, $E \rightarrow S$, $E \rightarrow V$ are necessary.

Such encoding is straight forward in Woo's symemtric data structure as ($E \rightarrow E$ can be derived from $E \rightarrow V$ and $V \rightarrow E$). With Δ only $V \rightarrow E$ is available for a wire frame and obtainment of others: $E \rightarrow V$, $E \rightarrow E$, $V \rightarrow V$ requires file inversion. Thus Δ like Luo and Lukacs [1991] is limited to regular sets.

It was seen that Δ is not suitable for wire edges and sheet objects. Radial edge data structure of [Weiler 1986] is an extension of the face-edge (FE) data structure discussed in Part 2. In an analogous manner, in this section, we show, how to extend Δ to take care of all the non-manifold degeneracies. In an analogous manner, other data structures may be extended for all the non-manifold conditions. The relations to be stored are shown in Figure 26. Note that a lamina face may have a regular non-manifold edge also (e.g. in Figure 18, 5-6). The overhead because of catering to all the non-manifold conditions is given by $3V+2S+F+2E$ which is siginificantly less than the overhead of the radial edge from a face-edge (see the next chapter for storage estimate of radial edge and other data structures). Note that the approach presented here is to represent a special kind of entity, as a separate entity, as exemplified by the introduction of a new enity IV for a special kind of vertex, while Weiler [1986] uses a switch to denote special kind from a default entity. Thus in Weiler's scheme, the same vertex-use entity denotes all possible special vertices such as loop vertex and shell vertex.

Comparison with respect to access While in the proposed approach, we need as many as the number of special entities, the queries commonly occuring are facilitated with our approach (enumeration of all shell vertices is straight forward matter while in the Weiler's approach we need to scan the entire vertex list for the siwtch) and also virtual memory overhead is significantly lower because all the related data is kept contiguous.

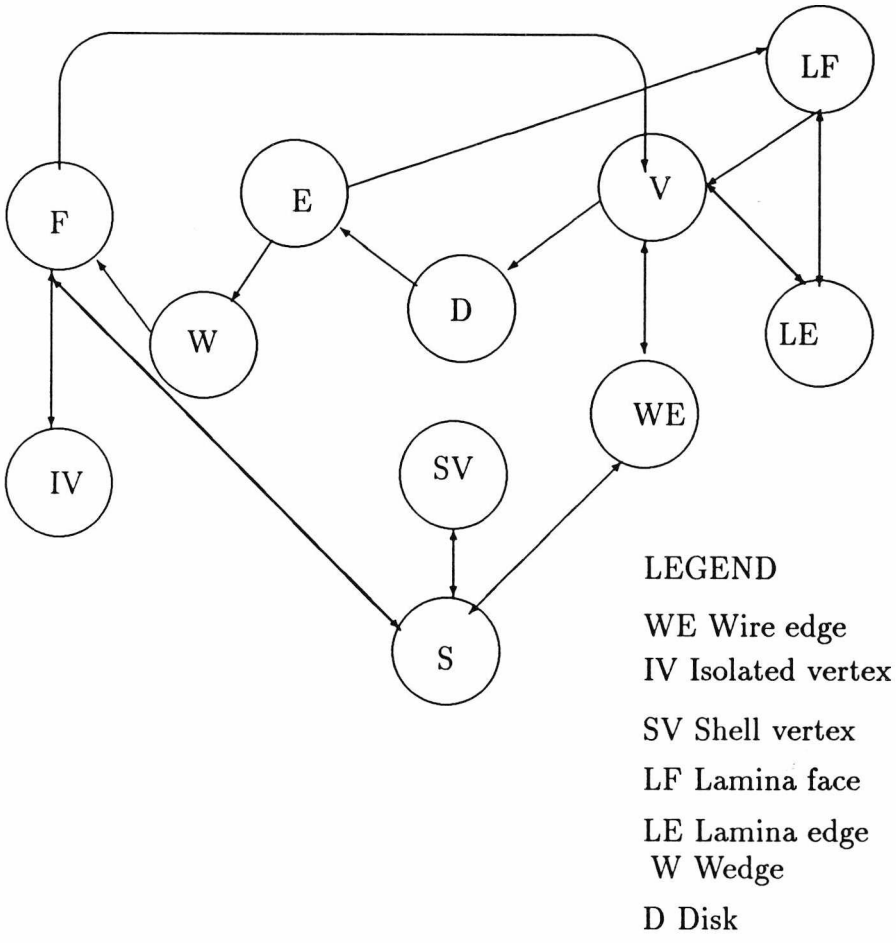

Figure 26: Extension of Δ for non-manifold objects

Weiler and McLachlan [1991] described selection filters (i.e. useful queries) which are useful with their radial edge data structure. The filters mostly consist of the special non-manifold conditions such as lamina face, wire edge, shell vertex. This information is directly available in the proposed extension without recourse to procedures to navigate through the data structure [1]. With implicit representation to know whether an edge is a wire edge it is necessary to examine the type pointer of one of its half-edges while in the explicit representation such a query never arises. Also it is a straightforward matter listing all the wire edges with explicit representation while in the other scheme it requires examination of the type pointer of all the shells. However, when an entity undergoes a metamorphosis (i.e. a change from one kind to another kind of entity, such as a wire edge becoming a solid edge), we need to delete and recreate the entity, while in the Weiler's approach it involves resetting a type field. Our approach makes implementation of algorithms simpler without special cases.

10.6 Conclusions

Several useful conclusions emerge from this work: a complex task of assimilating apparently disparate representations has been simplified by the adoption of an intuitive generalization based on a combinatorial view of the non-manifold topology and a uniform notation. In the previous chapter we have seen that a rich set of entities resulted from various attempts. In Part 2, we developed a systematic methodology for the design - the UDS served to consolidate the key ideas and made possible the specification of the data structures. For solution to any problem, often the most difficult

[1] Crocker and Reinke [1991] base their boolean algorithms on radial edge and hence require these procedures - see Part 4

part is the description or specification of the problem itself. This chapter serves that very purpose in the data structure design - captures the accumulated wisdom and expresses it conveniently. We obtained some elegant and useful analytical estimates of the storage under non-manifold domain. We also showed how Δ data structure can be extended for non-manifold wire and sheet conditions, without incurring the storagae overhead of Radial edge data structure yet possessing excellent access efficiency. In the next chapter, we however consider Δ to be devoid of such an extension for comaprative analysis.

Chapter 11

An Analysis of Non-Manifold B-Reps

11.1 Evaluation

In Part 1, we idenified several parameters for assessment of data structures. Accuracy is a measure of the support for exact geometry and appears under a different guise: 1-to-1 correspondence which is a broad term to encompass stability, sensitivity, unambiguity and uniqueness also . We defer the treatment of algorithmic complexity and convertibility until the next Part. Meaningfulness of the data structure will be studied at a general level in the following Chapter. The other criteria differ very little within boundary data structures and hence are omitted.

With the aforementioned pruning, we arrive at the following list of criterions for the evaluation of non-manifold boundary data structures.

1. Scope

2. 1-to-1 correspondence

3. Size and

4. Accessibility.

Each of these are detailed below.

11.1.1 Scope

In the previous chapters, we have identified the non-manifold conditions and described the principles of design philosophy. Thus non-manifold representations posess a much wider scope compared to manifold representations i.e. permit a great variety of configurations of the toplogical elements and degenarcies. The various representations analyzed are tabulated below with their non-manifold scope identified.

It should be noted that there are two columns in the Table, with similar heads: Internal face and Internal structure. As mentioned in the previous Chapter, provision can be made for the former by storing the two shells of each face. The latter requires, in addition, material attributes for different layers (i.e. shells) of the composite materials, such as those employed in the air-craft industry. The entries should be viewed with caution: their purpose is not brand the data structures, but to illustrate the rich diversity in the literature, as conceived by the original authors. The scope of any of the data structures can be widended by inclusion of few additional fields, as illustrated by the extension of Δ, for various non-manifold situations in Section 10.5.

Data Structure	Scope								Conciseness	
	Regular			Irregular					Storage	%
	Int. Face	Vertex	Edge	One vertex		Wire	Sheet	Int. Str.		
				loop	shell					
ACIS	×	×	√	√	√	√	√	×	$11E + 3L + V$	26
Ala	√	√	√	√	√	×	×	×	$6E$	12
CAD*I[a]	√	×	×	√	×	×	×	√	$4E + L$	9
Crocker	√	√	√	√	√	√	√	×	$41E + 7F + 13L + V$	98
Dobkin	√	×	√	×	×	×	×	×	$11E + F + V$	24
Gursoz	√	√	√	√	√	√	√	×	$40E + 6F + 6L + 8V$	100
Hanrahan	√	×	√	×	×	×	×	×	$11E + F + V$	24
Hoffman	?	√	√	√	×	×	×	×	$12E$	24
Karasik	×	√	√	√	×	×	×	×	$16E + 3L$	24
Laidlaw[b]	×	√	√	√	?	×	×	×	$4E$	8
Luo	?	√	√	×	?	×	×	×	$14E + 2L + 2V$	32
Masuda	√	√	√	√	√	√	√	×	$41E + 7F + 13L + V$	98
Murabata	√	√	√	?	×	×	√	×	$14E + 5F + 2L + 3V$	37
Stroud	×	×	× [c]	√	×	√	√	×	$6E + 3L + V$	16
Weiler	√	√	√	√	√	√	√	×	$41E + 7F + 13L + V$	98
Wu	√	√	√	√	?	√	√	×	$16E + 2F + 8L$	39
Yamaguch	√	√	√	√	√	√	√	×	$32E + 6F + 6L + 8V$	84

Table 7Scope and Storage

[a]Supports separate wire-frame and sheets

[b]Requires convex decomposition

[c]duplicated edge topology

11.1.2 1-TO-1 Correspondence

It is closely related to scope and means that for a given scope, an object has only one possible representation (i.e. unique) and no two objects have the same representation. None of the representations guarantee a unique representation. Karasik [1989] defined a canonical boundary representation to ensure uniqueness and solve the same object problem (i.e. if two representations correspond to a single object) for polyhedral models . Since this problem is not common in geometric modeling (more prevalent in computer vision and robotics), it has not caught the attention of researchers. It

can only be conjectured that an additional mechanism similar to Karasik's canonical representation will assure uniqueness of the other representations also.

The second part has been studied by Weiler (coined the term *sufficiency* to indicate the ability to completely and unambiguously interpret the representation), for manifold representations, which require the storage of at least one of the relations: $v \rightarrow F$ or $f \rightarrow F$ if any two faces share at most one edge, or $v \rightarrow V$ or $f \rightarrow V$ if there is at most one edge between any two vertices (polyhedra automatically meet this), $v \rightarrow E$, $f \rightarrow E$, $e \rightarrow E$. These conditions are met in all the data representations except that due to Liadlaw which meets the conditions under a restricted scope of polyhedra. Sufficiency and generality of Wilson [1988] are similar to sufficiency for polyhedra and curved objects respectively of Weiler [1985].

The topology of non-manifold objects is a super set of manifolds and hence it follows that the sufficient set (i.e. for unambiguous interpretation) for the non-manifolds is also a super set of the manifold set. However, obtaining such minimal sets of sufficient relations is difficult and hence we content ourselves by presenting heuristics (Weiler [1986] termed this, a practical minimal sufficiency). The additional set for general scope (excluding the internal structures) is region associations of single vertex shells and face associations of single vertex loops and wire-frame portion of a shell.

Single vertex shells and loops necessitate the minimum sufficiency to be augmented by $S \rightarrow V$ and $L \rightarrow$V, as exemplified by Ala and Karasik. However, since the degenerate vertices commonly occur in curved domain, geometric support is also essential. ACIS implements the same through degenerate edges which have only 1 end point.

The most general representations appear to be that due to Weiler, Gursoz and

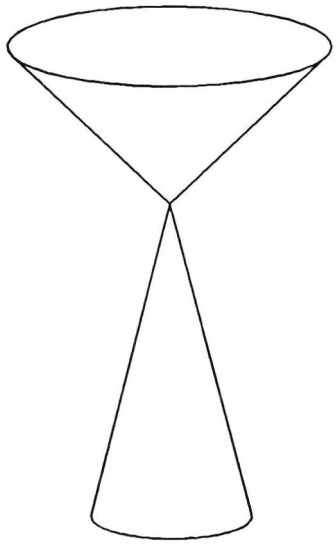

Figure 27: Cones with a non-manifold vertex

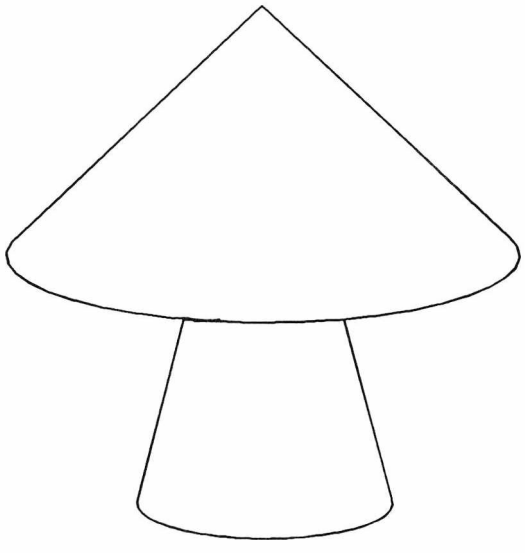

Figure 28: Two cones with a non-manifold vertex and one inside the other

Yamaguchi. Weiler proved only the completeness, unambiguity is still unproven. Consider the case of two cones touching at a vertex. Both the configurations i.e. the two cones overlapping at a vertex (see Figure 27) and one inside the other (see Figure 28) have the same representation in Weiler. Gursoz and Yamaguchi obtain a different representation. However for both representations the completeness and unambiguity has not been formally established. They (e.g. [Choi 1989]) often rely on the fact that no counter example has been found, to invalidate the premise of sufficiency.

11.1.3 Size

It refers to the storage values associated with the topological relations explicitly stored. Formulae applicable to the full scope involve a large number of variables (e.g. the number of isolated vertices) and hence are of little use for comparative study. Also several authors (e.g.[Weiler 1986, Choi 1989]) opine that the degenerate conditions other than the non-manifold vertex and edge are the exception rather than the rule. Thus both storage and accessibility performance estimates under a restricted scope (i.e. non-manifold vertex and edge) appear to be good indicators of the overall performance in a general scope. We know that $H = 2E$, $H2 = A = 4E$, $L2 = 2L$. Additionally we assume that $D = V$, $Z2 = D2 = 2V$, $W = E$, to obtain meaningful expressions which are comparable. The coarse estimates form a rough guide to the relative storage performance and are shown in the last but one column of Table 7. These are simplified by applying Euler's formula and substituting $E = 2F$ and $F = 0.5V$ (these expressions are empirical values from Wilson's study of common

engineering design objects [Wilson 1988]) in the resulting expression. The last column shows percentage of maximum storage (i.e. of Gursoz's representation). Storage estimates for other non-manifold conditions were already discussed in the previous chapter.

11.1.4 Accessibility

As seen in Part 2, for manifold representations it is commonly held that the accessibility of the adjacencies of the three elements (i.e. Vertex, Edge and Face) is a good indicator of the overall performance. We believe that they form a reasonable criterion for non-manifold models also. As mentioned in the previous section the occurrence of degeneracies is rare, in typical modeling operations. Also it is important that a representation support not only modeling but also applications (e.g. star edge suitable for modeling but not for visualization [Karasick and Leiber 1991]). Weiler stored vertex, edge and face based adjacencies which can be derived from the other information (i.e. their *use* entities). The redundancy is justified on the grounds of insulating the application programmer who is at ease with the intuitive Vertex, Edge and Face, rather than their incomprehensible *use* concept. However the extra entities (e.g. use, disk, zone) make the modeling task easier i.e. enable the design of elegant boolean algorithms.

D. Str.	$v \rightarrow E$		$e \rightarrow F$		$f \rightarrow V$		$v \rightarrow F$		$e \rightarrow V$		$f \rightarrow E$		$v \rightarrow V$		$e \rightarrow E$		$f \rightarrow F$		Total	%
ACIS	2	2	1	2	2	3	2	3	1	0	2	1	2	3	5	2	2	2	$16N + 21$	50
Ala	1	0	1	0	1	0	1	1	1	1	1	1	1	3	7	0	1	2	$7N + 16$	25
CAD*I	-	-	-	-	-	-	-	-	1	0	1	0	-	-	-	-	-	-	–	-
Crocker	1	4	1	4	3	2	1	8	1	2	3	1	1	8	1	4	3	5	$36N + 25$	100
Dobkin	1	2	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	$10N + 11$	30
Gursoz	1	4	2	3	3	1	1	6	1	1	3	2	1	4	2	3	3	4	$27N + 23$	77
Hanrahan	1	2	1	1	1	2	1	2	1	0	1	1	1	3	1	1	1	1	$13N + 9$	36
Hoffman	1	0	1	0	1	0	1	0	1	0	1	0	1	1	3	1	1	1	$4N + 9$	14
Karasik	1	0	1	1	2	2	1	0	1	0	2	1	1	2	2	1	2	2	$9N + 13$	29
Laidlaw	-	-	-	-	1	0	-	-	-	-	-	-	1	0	-	-	-	-	–	-
Luo	1	1	1	2	2	1	1	1	2	1	2	1	1	2	1	3	2	3	$15N + 17$	45
Masuda	1	4	1	4	3	2	1	8	1	2	3	1	1	8	1	4	3	5	$36N + 25$	100
Murabata	1	3	1	4	3	2	1	4	2	2	3	2	1	3	4	3	3	3	$23N + 29$	71
Stroud	1	0	1	1	2	1	1	2	1	0	2	1	1	1	3	0	2	2	$7N + 14$	24
Weiler	1	4	1	4	3	2	1	8	1	2	3	1	1	8	1	4	3	5	$36N + 25$	100
Wu	1	0	3	0	1	2	1	2	1	0	1	1	1	1	1	1	1	3	$11N + 9$	31
Yamaguch	1	1	2	3	3	3	3	4	1	1	3	2	1	2	1	3	3	4	$22N + 25$	66

Table 8Record access estimates.

Number of record accesses for initial conditions and per adjacency element are given in the first and second halves of each column.

In Chapter 7, it was seen that there are two types of costs involved in accessing the adjacent elements: field and record access costs. Previous studies [Ala 1992] have reported that the latter dominate in a virtual memory environment. As discussed in chapter 7, virtual Memory exhibits anomalies: the higher the storage the more the number of page faults. Since non-manifold data structures store large data compared to manifolds the findings are more significant for the non-manifold models. Although main memory is getting cheaper and larger, for quite some time to come, we will have to operate in a virtual memory environment. This is because the size of the programs is also getting larger in tune with user's ambitions and although main memories may be larger than a single user's application, it is likely that several users will be contending for the same memory and each may not be left with enough memory to escape the virtual memory overhead. Record accesses is important in main memory environment also because they make it difficult to utilize cache memory, but the

effect is not so much as in virtual memory, they can at most double the access time. Because of their dominance, in this thesis we tabulate the record access costs only, in Table 8(an entry of - indicates that file inversion is required i.e. number of accesses is linear in the number of edges). They thus constitute the worst case access figures. Field access costs may be worked out similarly.

Overall Performance

Each representation has 18 entries (two each for the nine adjacency relationship) and as such are not comparable at a gross level. It would be instructive to obtain an overall rating for each of the representations. Since we assume that the data structure supports satellite applications (e.g. display) also, we can safely assume that all types of nine queries arise with equal probability (see [Ala and Chamberlain 1991] or Part 1 of the thesis) and thus their sum forms the chief criterion for an overall rating, as we did before for the manifold data structures in Part 2. We also know that for each edge there are at most two adjacent vertices and the adjacent number of edges is the sum of the number of adjacent edges at either end minus two, hence we can substitute $2(|v \rightarrow E| - 1)$ for $|e \rightarrow E|$. For the rest of the seven adjacent relationships we do not have any such elegant expressions, in a general non-manifold situation, and hence we make use of the manifold equalities. We make use of the manifold equality that each edge has two neighboring faces. For the remaining six adjacent relationships, as described in Section 5.2, Woo and Wolter [1984] proved that on an average the number of adjacent elements (denoted by N, in Table 8) ranges from 3 to 6. We adopt the midpoint of the region i.e. 4.5 in our rating scheme. Armed with these expressions it is a straightforward matter to work out an overall accessibility rating

as shown in the last column of Table 8.

11.2 Summary and Conclusions

This work is a guided tour through the fascinating world of non-manifold boundary representations, nearly all the non-manifold boundary representations to date have been covered and critically analyzed.

This work should be of value to beginners, developers and researchers in Geometric Modeling. However a word of caution: we had to resort to several simplifications to obtain overall storage and access performance ratings, and as such should form a rough guide only, for launching a finer analysis based on each user's requirements.

Chapter 12

Validity of Boundary models

In Part 1, we identified several criterion for an assessment of data structures. Several of them have been used in the previous chapter for analysis of an assortment of non-manifold data structures. In this Chapter we are concerned with the meaningfulness of data structures.

12.1 Invalid boundary models

One of the main drawbacks of the B-Rep data structures is how to ensure that the final model is physically valid. CSG does not need any special mechanism to guarantee validity. However in B-Reps, vital amount of work may be lost if the final model is invalid i.e. not manufacturable.

The problem is somewhat analogous to the data input and correct data transmission on networks. Systems designers use data input validation techniques. An additional check digit guarantees that any error in the number is detected. In modeling, Euler's formula serves the role of the check digit. Again, as with the data base

and networks, the types of errors can be due to the user's ignorance or due to mistyping. We paraphrase, below, the errors and the tests for their prevention. Below a transaction denotes a sequence of manipulation tools which may be due to a single boolean operation or a sequence of primitive operators (e.g. KEF operator to kill an edge and a face, which is referred to as an Euler operator).

- Ensuring validity of the transaction
  - *Sequence test* to ensure the proper sequencing operation. For example, KEF can only be applied if such an edge and face exist and only if the edge is shared by two faces, as discussed by Mantyla [1988].
  - *Completeness test* The sequence of operations should be terminated with the proper operators for meaningful results.
- Ensuring validity of the geometric data

  By assigning incorrect geometric data to a topologically valid model, it is perfectly possible to create physically invalid objects [Mantyla 1988].
  - *Existence test* Check if the user specified geometry parameter exists (e.g. free form surfaces may not be supported).
  - *Limit or range check* Often the range is known and the input value may be checked against this and reported if outside the range.
  - *Combination test* Geometric data should be checked for combination (e.g. a description of circle involves not only its centre but also its radius).
  - *Robustness tests* Tests for unusual surface intersections, singularities and coincidencies which are often caused by the limited precision of the computing machinery

- Ensuring validity of the topological data, similar to geometric validity above
  - *Existence test*
  - *Combination test*
- Guarding against inadvertent errors
  - *Undo operator* can recover from errors.

12.2 Techniques for model validation

12.2.1 Techniques for topological validity

The Euler's formula is an invariant on the number of topological entities. Every solid object must fulfill this invariant condition. In the case of non-manifold objects physical realizability is no longer applicable. Since Euler formula applies to solid objects only and non-manifold includes wire frame and surface models, we do not have an automatic invariant condition on which to base our model manipulation tools. Euler's operators can guarantee only one (topological validity) of the four classes of errors, listed in the previous section.

Euler's formula for manifold solids

The advantages of the manifold euler operators are that a finite number of them are adequate for modeling any arbitrary object (but Mantyla [1988] does use some other operators such as new, addlist and dellist, because Euler operator invocation incurs an extra overhead such as a function call and certain operations such as splitting and boolean operations create more than one solid and the Euler operators required

for all such cases form a large number). They can also form the basis of inversion algorithms.

Mantyla [1988] uses two layers of Euler operators: low level, in which, the parameters are pointers and high level in which the parameters are generally vertex and face identifiers. The high level operators do scanning to retrieve the pointers of the identifiers and then call the low level operators. For example, the low level mev's (make edge and vertex) arguments are two half-edge pointers and a vertex identifier, whereas the corresponding high level operator's arguments consist of identifiers for one solid, two faces, and four vertices. Note that in Mantyla [1988] scheme , edges and half-edges have no such identifiers and are identified by their vertices (for example, a half edge is identified by its face and its two vertex identifiers).

Euler's formula for non-manifold objects

Because the Euler's formula applies to manifold solids only, several authors have attempted to generalize it. Two main directions in the extension: generalization to non-manifold R-sets, where wire and sheet conditions are disallowed and an euclidean cell based approach to allow such degenerate solids also.

Extension to all non-manifold objects Weiler [1986] defined only the data structure but provided no mathematical basis for the basic modeling operations. Masuda et al. [1989] proposed a mathematical theory of modeling, based on Euclidean cell complexes. A cell complex stands for a collection of wire frame, surface and solid models.

A cell can have a dimension ranging from 0 to 3, for a vertex, edge, face and volume respectively. The cells do not include their boundaries. For example, an edge

cell includes all the points up to but not the end points. The term closure qualifies cells with their boundary included. A model (termed the cell complex) is an union of all the cells such that each cell's boundary is also a part of the model and no two cells share any 3-D space. Objects with through holes and cavities need preprocessing: divison into their constituent volumes, each of which do not contain through holes and cavities.

The authors gave an extended Euler-Poincare formula applicable to the above regime.

$$v - e + (f - r) - (V - V_h + V_c) = C - C_h + C_c$$

v, e, f stand for number of vertices, edges, and faces. r, V, V_h, V_c, C, C_h, C_c stand for the number of holes in faces, volumes, holes through volumes, cavities in volumes, cells, holes through complexes, cavities in complexes respectively.

As discussed in a previous chapter, they represent both an empty and a full cube, the former is a collection of 2-D cells while the later also includes a 3-D cell (i.e. a volume). When a space is closed with a face, a hollow model (cell complex - not a solid volume) is created. An additional euler operation is required to fill the hollow space and extinguish the cavity. The data structure is identical to Radial edge except that the two topmost level entities model and region are replaced by complex and volume. In this scheme a cube with a through hole and a cube with another cube of different material embedded in it (such as in composite materials) can both be represented, the latter has value of 0 instead of 1 for C_h and 2 instead of 1 for V (the other 8 values being identical). The extended Euler-Poincare formula has 10 variables and thus forms a 10 dimensional space which can be spanned by 9 independent vectors. Accordingly 9 Euler operations (each with its own inverse operator, totalling 18), are

sufficient for any arbitrary modeling task.

The extended Euler-Poincare formula is independent of a data structure for encoding the information. Note that holes and cavities in complexes and volumes can also be represented by a single shell representing their disconnected boundaries. Thus the data structure needs only 7 topological entities. Accordingly the minimum number of Euler operations can be reduced.

Extension to non-manifold solids For extension to R-sets different authors have proposed similar formulae. For example Murabata and Higashi [1990], proved that

$$(V - V_r) - (E - E_r) + (F - F_r) = 2(S - H)$$

where the subscript r denotes multiply connected entities, S and H denote the number of shells (maximal connected simplices) and number of handles (holes or genera). In the non-manifold object, H can assume a negative value for conformity with the equation. Although theoretically the number of sufficient Euler operators is 5, various authors preferred redundancy on the grounds of efficiency and natural manipulation. This increased the number of entities in the Euler-poincare equation dramatically. For example, Luo and Lukacs [1991] bases their Euler operators on a 22 dimensional space (21 topological entities), thus naturality achieved by sacrificing simplicity.

12.2.2 A general validation method based on visual feedback

As mentioned previously, Euler operators can at best ensure topological validity and hence we need additional mechanism for guarding against other errors. An interactive design is only possible by providing the user with fast visual displays of the object

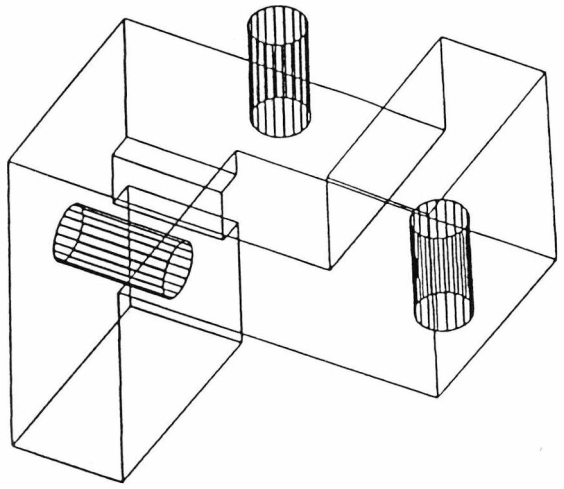

Figure 29: Widget

at crucial steps of the modeling process. Some gain in speed is possible by the avoidance of the usage of the Euler operators and their associated function call overhead. For example Stroud [1990] chose to do away with the Euler operators in the wire frame manipulation, which is predominant in the intial design stages. A wire frame display being itself ambiguous (e.g. see Figure 29), does not guard from invalid models. Hence we need a hidden line removal algorithm or hidden surface algorithm. The latter depends on the hardware and is relatively expensive to run. For fast response which is a must in interactive modeling and for device independence we prefer a hidden line removal algorithm (incidentally with hidden line removal it is possible to measure and dimension, a desirable characteristic especially for complex assemblies). Most of the commercial modelers use some form of visualization ([Hewlett-Packard

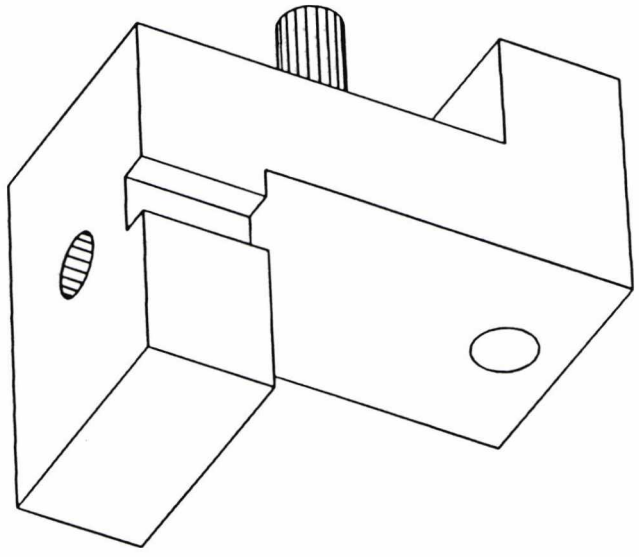

Figure 30: Widget without hidden lines but polygonal generators

1992] solid modeler based on ACIS kernel uses hidden surface, while [Datavision 1992] uses hidden line removal). A display algorithm enables the user to check the validity of the object and to take corrective action (such as an undo) when the model is not according to his wishes.

Although there exist a great variety of hidden line removal algorithms, none of them cater for non-manifold objects. For example Magrabhi and Griffiths [1989], McKenna [1987] and Kripac [1985] permit edges to be shared by at most 2 faces (i.e. manifold objects). Similarly no two solids can intersect, and there can be no dangling or isolated planes or lines in [Ottomann, Widmayer and Wood 1985].

Also most of the hidden line algorithms display the polyhedral generator edges of the curved objects (see Figure 30). For improved visualization it is desirable to avoid

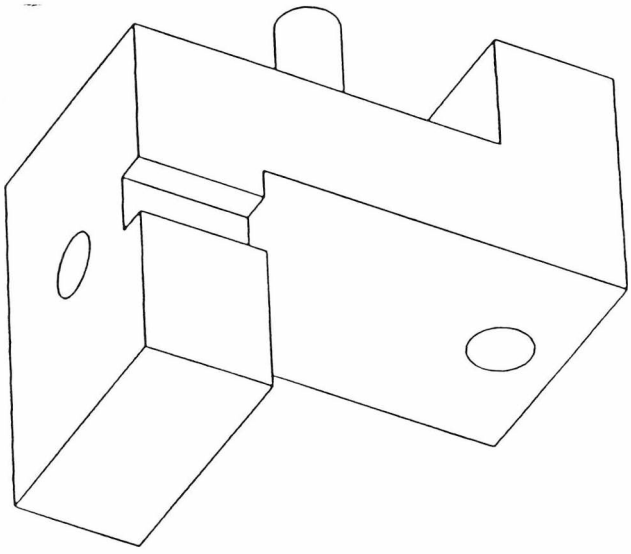

Figure 31: Widget without hidden lines and polygonal generators

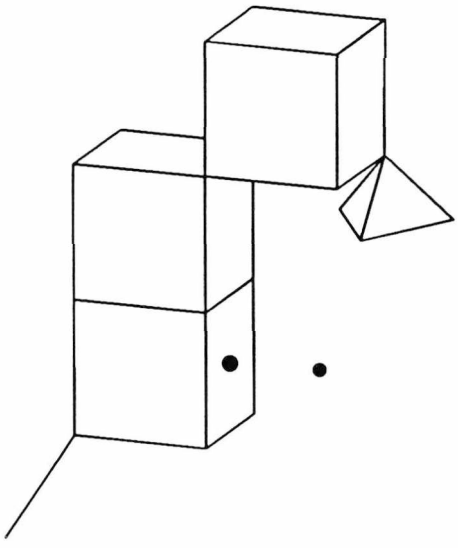

Figure 32: Hidden line removal on a Non-manifold object

their display (see Figure 31). An approximate display of curved objects is possible by ignoring the plot of an edge, whose included angle (i.e. the angle between the two neighboring faces) is less than a preset threshold. This check requires retrieval of adjacent face information for each edge, which is readily available in Δ data structure. Based on Δ data structure a hidden line removal algorithm has been implemented which is completely general (for example see Figure 32) and displays aesthitically curved objects (for example, compare Figures 30 and 31) and is also faster.

12.2.3 Limitations of the proposed method

Johnson [1986] identified two types of reliability problems: impossible objects and latent defects. The above discussion is centred around the former problem. Latent defects, which involve a compromise in the integrity do not show up immediately. Thus visual feedback is not enough to detect latent defects. The modeler must do

extensive geometric tests, for unusual surface intersections, signularities and coincidences to remedy the situation. Some of these problems can be partially mitigated by the adoption of non-manifold topological data structures and robustness techniques developed by Hoffman, Hopcroft and Karasick [1988]. But the bulk of the latent defects place heavy demands on the compute intensive geometric tests. The thesis being predominantly concerned with topology, solutions for latent defects, are beyond its scope.

12.3 Conclusions

Boundary models are susceptible to invalidity and Euler formula is only a partial solution, and can remedy only a few of the causes. While acknowledging that a formal basis is difficult (or impossible!) for an automatic enforcement of validity, we have argued and implemented a pragmatic approach to address the nagging invalidity issue.

Part IV

Efficient algorithms

We lay the foundation for a fast interactive geometric modeler in this part, which also illustrates several of the ideas presented hitherto. This part also details the implementation aspects, deals with interface of the algorithms with the data structures presented in the previous two parts and the efficient conversion of boundary data structures.

Chapter 13

An Introduction to Algorithms

So far we chiefly concerned ourselves with the data structure storage and retrieval. In this Part we study the interface between algorithms and data structures. Algorithms depend on the application at hand and hence it is useful to categorize them, which is the subject of this chapter. As will be seen in the following Sections, different categories place different demands on the data structure. In Part 1, we have seen that there are several uses of CAD data - a part once modeled is utilized by a variety of users. We can categorize algorithms into modeler specific and application specific. The issue we address in this chapter is how to structure data to satisfy both these categories. In Part 1, we argued at a general level that a single CAD data base is better because it ensures integrity of data. In this chapter we delve deeper into the same issue but in the more specific context of the boundary data structures - debate whether to use a single or multiple data structures accross the whole spread of uses. This is illustrated with a comparison of the data structure requirements of data communication, which is a specific illustration of application algorithms, with modeler algorithms. The Chapter is wound up with a discussion on the standard

approaches for handling modeler and application algorithm requirements.

13.1 Modeler algorithms vs. Application algorithms

13.1.1 Classification

Manipulative usage [Kalay 1983] refers to operations which alter (modify, delete) the data as typified by set operations and others involved in model building stage. Non-manipulative or post-modeling use refers to retrieval of information from archival data bases without modification as typified by visualization and robotic applications such as computer vision. The terms manipulative and non-manipulative can also be interchangeably used for modeling and application support, the person involved referred to by end-user and application developer respectively. End-user [Dietrich et al. 1989] typically uses solid modeler for design or analysis.

Baer et.al. An early example from literature, on the modeler vs. application debate is the survey by Baer, Eastman and Henrion [1979] which reported that one relation of the nine is sufficient - which one depends on the application. e.g. Vector graphics: $E \rightarrow V$ or $F \rightarrow V$ or $V \rightarrow V$ (how vertices are joined), Boolean (i.e. shape) operations: $V \rightarrow F$, Euler operations: adjacency amongst faces e.g. $F \rightarrow$F. Out of the 11 systems surveyed, 8 used $F \rightarrow E$, 5 used $E \rightarrow V$, 4 used $F \rightarrow V$, 3 used $E \rightarrow E$, one each $F \rightarrow F$ and $V \rightarrow E$ and none used $V \rightarrow V, V \rightarrow F$.

Most of the modelers are geared to making end-user's jobs efficient. For example, Karasick and Leiber [1991] concedes that his star edge data structure is suitable for modeling but not for visualization. Radial edge of Weiler [1986] stores the vertex, edge and face based adjacencies which can be derived from the other information

(i.e. their *use* fields, which facilitate elegant boolean algorithms). The redundancy is justified on the grounds of insulating the application developer who is at ease with the intuitive Vertex, Edge and Face rather than their incomprehensible *use* concept. TGMS [Dietrich et al. 1989] was aimed at correcting this bias i.e. provision of good support for the application programmer's unanticipated tasks.

13.1.2 A case study: Communication algorithms

To amplify on the gulf between the requirements of applications and modeler algorithms, we study a prime example of applications viz. communication. With interdependence increasing there is an increasing need for exchange of information amongst different systems. Wilson [1988] states that the chief requirement of data structure for exchange are preservation of data, compactness for cheap communication, and "to not impose unduly complex processing requirements on the sending and receiving systems to minimize processing development costs". It is commonly held that brevity and processing complexity have inverse relationship. This was shown to be fallacious in certain topological cases, by Ala [1992]. In virtual memory environments for topology compactness is desirable for fast processing.

Wilson concluded that access and compactness are desirable for a modeler while, for communication compactness is more important than processing efficiencies. Processing refers to conversion algorithms between the host and transfer structures. Hence Wilson [1988] preferred a skeletal version of modified winged edge structure for communication. The edges of a loop are circularly linked and the structure allows for minimum representation for a sphere. Transfer structures do not need back pointers. Sufficiency and generality of Wilson [1988] are similar to sufficiency for polyhedra and

curved objects respectively of Weiler [1985]. Wilson felt that for a modeler sufficiency, generality (absence of restrictions) on graph structure, compactness and algorithmic complexity are all important. Δ being the most compact amongst the constant time data structures is a natural choice for data exchange.

Study by the CAD*I committee Schelechtendahl [1988] has found that processing time

$$T = O(complexity) = O(V + E + F).$$

Interestingly the complexity does not depend on the number of loops although the objects considered had loops varying from F to $2F$ in number. Other performance factors include the size of host and transfer structures.

13.1.3 Approaches for handling applications and modeling

There are two approaches for handling applications and modeling.

A single data structure

We could use the same representation as that of the modeler for applications. The applications are tightly coupled to the modeler.

Application specific (i.e. custom made) data structure

Alternatively, we could use a separate representation for each individual application. But one has to address the problems of representation conversion and their associated large I/O costs and maintaining the data integrity (recall that relational data bases have the virtue of data integrity over the hierarchical and network data bases). An advantage of this approach is that years of modeler work need not be discarded and

at the same time insulate user applications from the intricacies of the modeler. An analogy may be drawn with the machine (i.e. modeler's data structure) independent UNIX operating system (i.e. schemata).

Example implementations Examples of the latter approach are the husks of the Spatial-Technology [1991], schemata of Karasick and Leiber [1991] and the Dietrich et al. [1989] application layer atop the GDP modeler. They filter modeler data into a simpler and uniform representation. Efficiency of the schemata construction depends on both the modeler representation and nature of the application queries (see for example discussion on TGMS).

Some authors have attempted to make use of the modeler routines for applications. For example, Karasick and Leiber [1991] proposed the usage of the vector classification results of the modeler for applications.

Schemata Karasick and Leiber [1991] states that boundary traversal (e.g. containment $E \rightarrow L, R \rightarrow S$) of application algorithms is similar to the vector neighborhood classification i.e. whether a vector points into a S, F or along an E. $2D$ and $3D$ neighborhood classification procedures of the modeler can be used to subdivide or unite modeler objects and other schemata for use with application programs.

TGMS Dietrich et al. [1989] uses the entities: V, E, L,Solid. Since N_{Lf} is variable, faces are not stored. The data representation used is $L \rightarrow E$ and the loop nesting tree (i.e. the parent-child links). TGMS provides methods to get all the topological information between each pair of entities. Order is considered to be significant only in $L \rightarrow E, V$ (i.e. the order of occurrence of edges and vertices when the loops are traversed with the material on the left). The cost of the methods

depends on the modeler's representation e.g. since GDP does not have $V \rightarrow E$, the method is very slow and an impediment to the solid modeler independence. The authors state that access cost can be reduced by caching and preprocessing, which increase the storage, so usage pattern is important to strike a balance between the storage and access.

13.2 Application algorithms vs. Application algorithms

Not only application requirements differ from modelers, but also from application to application e.g. to solve the same object problem Karasik [1989] combined multiple modeler faces on the same plane into maximal connected faces and faces are described by geometrically (i.e. lexicographically) ordered vertices. Note: same object problem uses only $F \rightarrow V$. However visualization applications may require division of modeler faces (e.g. convex decomposition is needed for certain hidden line algorthms).

13.3 Conclusions

Algorithms can be broadly grouped into modeler and application oriented. Each exhibit slightly differing preferences for the choice of storage. An approach, whereby a reasonable performance is guaranteed for the whole spectrum of algorithms (starting from communication which demands a lean data structure to modeler algorithms which demand data rich representations) is possible by the adoption of Δ, instead of tailoring a custom made data structure for each individual application. Δ is an excellent compromise candidate, as it does not incur the overhead of storage intensive data structures such as Weiler [1986]'s radial edge data structure. A custom made data structure implies large I/O overhead for conversion from modeler data structure

and also may lead to inconsistencies amongst different versions of the same data.

The Chapter has highlighted the diversity of algorithms and as such the diversity is too complex to be addressed in a single thesis. In the next two Chapters, we only address algorithms for interactive design and data conversion, which form a miniscule proportion of all possible algorithms.

Chapter 14

Fast Model Manipulation

One of the important goals of this thesis has been the development of fast modeling tools. In an interactive modeling fast response is vital. In the previous Parts we proposed a data structure which is compact and analyzed it for performance in real situations. But we were primarily concerned with the low level interrogation and their optimization. The queries consisted of retrieving the topological neighbors for each of the entities. But in a modeling environment it is common to provide high level tools for the manipulation of models. However such algorithms (notably the boolean intersection which forms the corner stone of the other boolean operations) are very slow. In this chapter we investigate techniques for realization of speed in modeling operations. We extend our analysis of low level representation optimization for achieving fast high level algorithms. In Section 6.4, we employed special purpose design, based on query frequency estimation. We adopt a similar strategy i.e. we prioritize modelling tools based on their usage pattern and then focus on the critical tools. This results in a fast modeler in an overall sense. We start with an investigation into the requirements of an interactive modeler and exploit the commonly

occurring interactions between objects. We then describe an approach which exploits the requirement analysis.

14.1 A fast modeler with special purpose optimization

As discussed in Part 1, interactive design usually involves

- local modification
- visual display
- sweep and
- booleans

By examining their frequency of usage, we can build an optimal interactive modeler. [Johnson 1986]'s study, on the solid modeling usage pattern reported that approximately 30% of the total project time is spent in the design phase. In the design phase 50% of time is spent on the local modification. As argued in Section 12.2.2, in interactive design, visual feedback is very important to the user, so we expect a hidden line [1] removal algorithm execution after every alteration to the model, which assigns visualization a very high frequency of 40%. As the following discussion reveals, the frequency of occurrence of the conventional booleans is very small. Sweep usage frequency has not been estimated, but it can be expected to occur more frequently than booleans. Thus the order of usage of the modeling tools corresponds to their listing above. We discuss below a fast means for each of the above tools. In the next two sections we examine the question of choosing the best data structure

[1] Hidden line is better than hidden surface, since it is possible to dimension, an aid for parametric modeling and complex assemblies

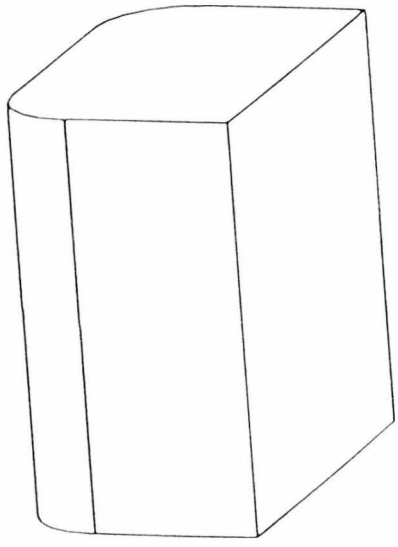

Figure 33: Chamfer on a block

for local modification and sweep. Local and global modifications are exemplified by an analysis of chamfer operation and sweep (booleans being the other kind of global operations) respectively.

14.2 Local modification

Various forms of local modification were introduced in Chapter 2. In chamfer, the face may be planar or quadratic (cylindrical for edge as shown in Figure 33 and spherical for a vertex). The usage of splines is not so widespread as quadratic surfaces. In local operations, it must be possible to retrieve the data of an arbitrary face or vertex or edge and update the appropriate topology and geometry. As an example, for the chamfer on an edge, we need to identify the affected face, edge and vertex identifiers.

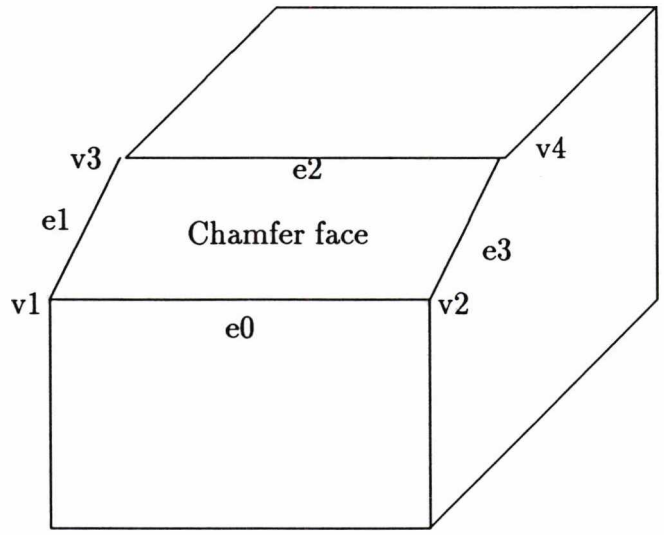

Figure 34: Chamfer example

Thus we need quick retrieval of $e \rightarrow V$, and $e \rightarrow E$ and $v \rightarrow F$. Similarly for a vertex chamfer, we need vertex based information i.e. $v \rightarrow E$, $v \rightarrow F$ and $v \rightarrow V$ to identify the affected edges, faces and vertices. The cost can be broken down into two broad categories: cost of identifying the affected entities and the cost of modification. If the data structure employs other than the three entities, we need to identify them as well (for example in half-edge data structure, we need to identify the affected half-edges as well). The cost of modification is dependent on the data structure, in question. As the discussion below reveals, the modification cost increases with the data structure size, thus dissuading us from large sized exotic data structures.

Analysis of chamfer

To illustrate local modification, we consider chamfering an edge of a block, as shown in Figure 34. The chamfer involves the following operations:

- Creation of three edges ($e1 - e3$ in Figure 34).
- Creation of two vertices ($v3 - v4$ in Figure 34).
- Creation of one face.
- Updation of four edges.
- Updation of four faces.
- Updation of two vertices.

The basic algorithm is as follows. We first need to get the V, E and F identifiers before retrieving them. We need to identify all the faces affected and alter their vertex list. They are the faces adjacent to the two vertices of the original edge e_0. Hence we need $e_0 \rightarrow V$, say $v1$ and $v2$, followed by, $v1 \rightarrow F$ and $v2 \rightarrow F$.

Which of the above operations is actually needed, depends on the data structure in question. Let us consider three alternatives which carry progressively more topological information. Since we want to study the effect of topology only, we assume the same geometry in all the three cases, say the vertex coordinates.

14.2.1 Oriented polygon list of vertices

The only topology update needed is $F \rightarrow V$. We need to identify all the faces affected and alter their vertex list.

With the data structure, we need a linear scan of all the face lists, to find all such faces whose vertex list has both the two vertex identifiers i.e. $v1$ and $v2$. The operation is clearly linear in the number of faces and hence the complexity is $O(E)$.

14.2.2 Δ data structure

As tabulated in Table 3, on an average, $e_0 \rightarrow V$ requires, three record accesses and $v1 \rightarrow F$ and $v2 \rightarrow F$ require four record accesses each (as a byproduct we also know $e_0 \rightarrow E$). Thus a total of 11 record accesses are required to retrieve the identifiers of the vertex, edge and face entities to be updated. Since Δ only maintains one record per entity (e.g. the two faces of an edge) the total number of records to be retrieved which need updation are 10. Hence the total records retrieved for updation are 21. For the entities to be created we know their adjacent entities from the identification step done before. Hence we only need to add the records to be created to get the grand total of record alterations which is 27.

14.2.3 *GDS* data structure

The basic steps are analogous to Δ. The identification cost is only one for the two vertices, two for the faces and two for the edges, a total of five records. However for each of the updated entities, GDS maintains all three entities, and hence each updation is three times that in Δ. Note that it may be possible to maintain, them in one contiguous memory allocation (and hence requiring one record) but it requires special effort since the natural data is very unlikely to be available in that form. Since the total number of updated entities is 10 and each entity has three records, the total records is 30. As before the creation process involves six entities. Hence the grand total is 41, which is nearly one and half times that in Δ.

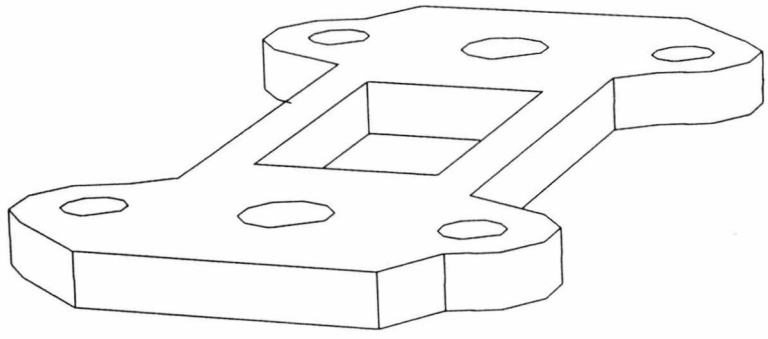

Figure 35: Gizmo

14.3 Sweep

The number of faces F, in translational sweep (e.g. see Figure 35, which was produced by sweeping the top face), is related to the number of vertices, V, of the swept face, $f0$, as follows.

$$F = |f_0 \rightarrow V| + 2 = 2 + |f_0 \rightarrow V|$$

During the interactive design phase, where mainly visual feedback is important, the oriented polygon list data structure is the fastest.

In summary, the total time for an algorithm , T, is given by,

$$T = T_1 + T_2$$

where T_1 = Total time required for a minimal representation with repsect to a stated goal and T_2 = Total time required for updating the redundant information.

For example, with a data rich representation (e.g. Weiler's Radial Edge), the penalty overhead of T_2 for sweep is very large, compared to a lean representation (e.g. polygon list). In other words, polygon list data structure is the minimal representation for sweep and display.

We may conclude the boundary data structure vs. algorithm debate by stating that the size of data structure, in addition, to the anomalies discussed in Chapter 7, also influences the update time. As a rough guideline, the cost of update in a typical algorithm is proportional to the total number of entities, further reinforcing our previous assertion that extra entities, such as half edge and loop, increase the access time. On an intuitive level, larger the data size, greater the time to maintain it. Thus the discussion shows that Δ data structure is an excellent choice for local modification and sweep together.

14.4 Fast algorithm for visualization

A key requirement of interactive modeling is a fast response usually in the from of visual feedback. Most of the hidden line algorithms require polygon oriented information and hence Δ, being a superset, is quite handy. It does not require any preprocessing step as is the case with [Spatial-Technology 1991] and most other data structures. Global coherence has been exploited largely by most algorithms but not local coherence. Again the Δ data structure's vertex adjacency information enables us to exploit local coherence for further speedup. Local coherence saves us from the expensive point-in-polygon test.

As discussed, in the preceding part, hidden line algorithm is indispensable for validity of models. For interactive modeling, we integrated the hidden line algorithm with the other modeling tools and Δ is natural facility for doing this. For curved objects it is desirable to avoid the display of the polygonal approximation generator edges. This requires $E \rightarrow F$, which is readily avaialable in Δ data structure.

14.5 Boolean operations

Boolean operations receive the most attention in geometric modeling literature. Several algorithms exist for them. Most of them form the intersection algorithm as a bed rock for the other booleans. They (e.g. [Hoffman et al. 1988]) exploit the fact that any boolean operation can be reduced to appropriate intersection and complementation between the participant objects, as shown below.

$$A \cup B = (A\prime \cap B\prime)\prime$$

$$A - B = A \cap B\prime$$

Hence it is common in the literature to detail an intersection algorithm with a hairy case analysis. For example, Karasik [1988] devotes half of his thesis to the description of an intersection algorithm. Booleans are notoriously slow. How can we speed them up? The key lies in the usage pattern. Unfortunately very little attention has been devoted in the literature, to whether we need such compute intensive boolean operations. As we shall see later, this approach promises tremendous gains in speed.

We discussed that features and an integrated wire, sheet and solid capabilities are the most importnat in an interactive design environment. In the next two sections we examine whether good support for these warrant the usage of such slow general purpose booleans or could they be substituted by specialiazed forms of fast booleans.

14.5.1 Requirements of modeling operations

Form feature modeling requirements

Section 8.3 provided a brief introduction to features technology. A key observation of features enables us to devise a special purpose boolean which is adequate for

the majority of feature modeling tasks. [Pratt 1990] defined feature as *a region of interest* **on the surface** *of a part* . The key point is that a feature occurs on the surface. The volumes defined by AFV and CFV are often identical and the feature is then termed as a regular feature. Only the irregular features give rise to non-null intersection or difference between the feature (protrusion and depression respectively) and the part. An example: a square depression on a block's top face constitutes a regular feature, if the top face is flat, an irregular feature otherwise (e.g. cylindrical). On the other hand, a popular example in literature [Gursoz et al. 1990, Hoffman et al. 1988, Laidlaw et al. 1986] is the intersection between two cubes identical but differing in rotation by a slight angle (to test the breakdown of the algorithm, for the rotation for which the intersection becomes a single cube). As the above discussion reveals, such interactions are rare in typical feature modeling operations. Although they are important to ensure a full proof robust modeler, Johnson [1986] assigned a lower priority to robustness compared to the speed. The interactions usually result in null intersection or difference between the attached feature and the parent object. This observation forms the main speed improvement. The boolean operations we describe forbid non-null intersections or difference between the participant parts. This restriction might appear to be too severe to limit the flexibility. However, apart from the tremendous gain in the speed, it disciplines users to force into thinking in terms of features, by restricting freedom to intersect objects. Unrestricted use of the intersection operation, may later, make it difficult to obtain a functional feature description of the model. It is interesting to note that although union and difference have analogies in manufacture (e.g. assembly by welding and machining respectively), intersection has none whatsoever. Thus it reflects the natural process of modeling and the subsequent manufacture. Our experience has shown that it can model any

object. It may be possible to obtain a linear algorithm, by sorting on the face normal vector component (a sort on one of the 3 components is enough) in addition to the conventional bounding box sort.

Unified modeling requirements

The requirements that stem from non-manifold modeling, have been well documented in the previous chapters. Briefly the manipulation tools should be capable of dealing with non-manifold interactions between various objects. Two parts may be touching at a single edge or a vertex or a face or there may be free standing edges and sheets and isolated vertices.

Most of the available commercial algorithms and literature (e.g. the well known [Mantyla 1988] algorithm) prohibit occurrence of such non-manifold conditions. The extended Δ (see section 10.5) comes in handy for tackling the non-manifold conditions.

14.5.2 Previous Boolean algorithms

We study the salient faetures of some recent boolean algorithms. They incorporate sophisticated non-manifold representations in their algorithms and are representative of the current trends.

Gursoz et.al.

The authors proposed a novel data structure which was described in great detail in part 3. Gursoz, Choi and Prinz [1991] classify boolean algorithms as top down and bottom up. In contrast to the conventional top down approach where the objects are

tested for possible intersection from objects to vertices in a hierarchical manner, they prefer a reverse order i.e. beginning from the lowest dimensional vertex to the 3-D volumes as follows. Test sequentially for intersection of a pair of

1. $V - V$
2. $V - E$
3. $E - E$ and $V - F$
4. $E - F$
5. $F - F$

A single tolerance value is enough to deal with all the intersections. A vertex has a tolerance zone, which is a sphere of radius equal to the tolerance value, an edge has cylinder of length equal to the edge's length and radius as before, and a face a slab whose thickness is the same as the tolerance and base equal to the face area. Each of the tolerance zone, excludes the tolerance regions of immediately lower level entities. Each of the above intersections is deemed to occur only, if one entity, falls within the tolerance zone of the other participant entity. The operation is repeated for each pair of entities from the two participant sets. There are 10 types of possible intersections between the four possible entities. The combinations with solid are implicit so, the net combinations are six only. Geometric interference test done only, if no topological connectivity exists between the entities. The common operations are deletion from a list, splitting an edge into two, and testing for common vertex between two edges. The authors claim that the advantage of the bottom-up approach is that the boolean operations of non-manifolds are identical to manifold boolean operations and are

robust (their modeler fails for very small angle between two intersecting cubes). It appears that the bottom up approach is incapable of exploiting the classic time saving (reduction from quadratic to near linear complexity) bounding box tests.

Crocker and Reinke

The algorithm is based on the most widley used non-manifold data structure i.e. Radial edge data structure of Weiler [1986] discussed in Part 3. Similar to Masuda et al. [1989], Crocker and Reinke [1991] proposed an editable non-manifold B-Rep, which provides a much wider domain (wire frame, surface and solid) and fast editing . A common boolean operation involves the following four basic steps.

1. Intersection
2. Classification
3. Selection
4. Topology construction

Instead of the conventional ordering (e.g. [Hoffmann 1989]), as above, they order the four basic steps as 1, 4, 2, and 3, in their approach termed as Merge and Select. Their approach further differs, in that, all but the appropriate parts are discarded in the selection step in the conventional, while they also maintain an additional history mechanism in the topology construction step. Their timing results show that the initial creation phase took the same time, but, 17 times faster in editing, and four times faster in total. The traditional approach requires reevaluation of the whole model except for addition (e.g. move, change boolean) and hence is proportional to the model complexity whereas the merge and select approach is local in nature and

hence its complexity is proportional to the number of primitives locally involved in the change. For operations involving global changes, the Merge-and-select approach performs the same as the traditional. The authors report that 50 percent of the time is spent in face-face comparison and only 30 percent in the topological operations. The storage requirements range from 1.5 to 3 times with the traditional booleans. This includes the considearble saving in storage by employing the idea that a surface is stored only once even if more than one face lies on it. Unlike the algorithm discussed in the previous section, this algorithm works on curved objects also. Hence they use surface-surface intersection as the starting point and classify the intersection curves and points, with respect to all faces on each surface. Mantyla [1988] starts with edge and face intersection, because his domain is polyhedra. It should be noted that editing is also possible with non-manifold data structures such as half edge, as illustrated by Mantyla [1988], who computes the union, intersection, difference simultaneously, but he resorts to ad hoc edges to achieve this. Mantyla's algorithm is very fast because it avoids the point-in-polygon/polyhedron test by exploiting a special vertex neighborhood classification, but is limited to manifold polyhedra, while the Crocker and Reinke's algorithm can handle curved as well as non-manifold objects.

Other boolean algorithms

Hoffmann [1989] and Karasik [1988] pioneered robust boolean algorithms. Their data structures have been already covered in Part 3. In addition Karasik gave a complexity analysis of intersection algorithm, using results from computational geometry. He proved that his algorithm requires a time complexity of $O(E \times log(E))$, where E is the product of the half edges in the two solids, involved. He states that the other linear algorithms, in litearture [Laidlaw et al. 1986, Paoluzzi et al. 1989], actually

require $O(E^2)$, because of the preprocessing into simpler faces (convex and triangular respectively). But as stated by Mantyla [1988] and Requicha and Voelcker [1983] the worst case analysis is not very informative and for practical speedup, geometric locality is more effective. A common approach for exploiting geometric locality is to use bounding box tests for edge, face and volumes, before attempting full scale intersection tests. Bounding box tests eliminate a lot of cases and offer excellent speed gains in complex objects.

14.5.3 A fast boolean algorithm

The intersection step of Crocker and Reinke [1991] algorithm proceeds by face with face intersection, followed by wire edges with faces, wire edges and shell vertices, shell vertices and faces with shell vertices. As noted in Chapter 11, several authors (e.g.[Weiler 1986, Gursoz et al. 1990]) opine that the degenerate conditions other than the non-manifold vertex and edge are the exception rather than the rule. The objective of unified modelling is to equip the user with increased flexibility for creation (not exclusion!) of solids, so a frequent operation is to make-regular or make-manifold the whole geometric model. The make-regular operation (e.g. see [Weiler and McLachlan 1991]) first checks for the existence of enclosed regions and then deletes all the lamina faces, wire edges, and shell vertices, in that order. This implies that it is convenient if the degenarate entities are explicitly available to obviate the need to scan the entire list of faces, edges and vertices. Hence the data structure, we designed in the section 10.5, forms a sound basis.

The algorithm handles non-manifold objects and also features. The most important goal is to achieve fast algorithm for interactive operation. The algorithm belongs

to the top down category. Two operations are permitted, either + or −, corresponding to union or difference respectively. The participant volumes can interact in only such a manner that the intersection ($A \cap B$) or difference (at least one of, $A - B$ and $B - A$) is null.

Note that the implementation is elegant and we can use the operator overloading facility of C++ to implement + and −. The operator + stands for vector sum of objects, or faces or edges. The top down approach rapidly eliminates non-intersecting objects before proceeding further down for expensive and innumerable comparisons. To achieve further improvement (near linear time) we may also sort on the face and edge direction vector components by using a $n \times log(n)$ sort algorithm (e.g. quick sort). Then only those edges or faces which have lower direction vectors need be considered.

Like any other boolean algorithm no new face geometry is created. Only topology changes and creation or deletion of faces occurs. The algorithm also does not create new edge or vertex geometry. Note that most of the intersection algorithms create edge and vertex geometries. The algorithm works on topological checks and modification, without geometric tests and thus is similar in complexity and simplicity to sweep algorithms. The algorithm has been implemented only for polyhedral objects. It however includes B-spline space curves and surfaces and all the quadratic surfaces approximated to polyhedra by an arbitrary resolution. A characteristic of the construction industry models is that they can be modeled with the restrictions in which our algorithm works. For rounding suitable cylindrical rounds are provided. Note that any CSG model domain can be built with just one cylinder primitive. Mantyla [1988] rigorously established that a finite sequence of Euler operations forms a sound basis

for the construction of an arbitrary model. In an analogous manner it can be conjectured that union and difference can model any arbitrary object. But this involves atleast some limit on the freedom of the user. An argument against the proposed algorithm, may be that it requires human intervention. But even a full boolean will constantly require human intervention because of the danger of degenerate conditions and non-manifold conditions in some modelers.

14.6 Conclusion: A basis for a fast modeler

We have seen that for the bulk of high priority opearations our Δ data structure performs excellent. This implies that the weighted sum of the execution times of interactive modeling operations (local modifications, visualization, sweeps and a specialized boolean, in that order) is the least with the approach we advocated. We have seen that the algorithm's restrictions on the user are reasonable and meet the bulk of the modeling tasks. This combined with the local modification, visualization and sweep algorithms affords a fast interactive modeler. Yet there will be applications which demand capabilities beyond those of the proposed algorithm. Such occasions are very rare in typical modeling operations, but for completeness, we have implemented the conventional booleans atop the Δ data structure. Their description has been omitted for brevity.

14.6.1 Comments on the implementation

Linked lists and Euler operations are conventionally used in the modeling world. For reasons elucidated in Section 12.2.1, we did not base our algorithms on Euler operations. We already discussed the virtues of the array implementation based on dynamic

allocation of memory, in Section 7.6.2. Further in interactive modeling, solids need to be created and deleted frequently. Allocation and disposal of memory is faster with array implementation. With dynamic allocation facility of the modern programming languages we no longer (as was the case with Fortran) need to, prespecify the array size. Although we argued against loops in Chapter 7, based on our experience with the implementation, we however think that algorithmic complexity would have been simpler with the adoption of loops. Some example outputs are shown in Figures 31 and 35.

Considerable time was spent in the development of a hidden line algorithm and modifying it to suit the aspect graph construction, which will be described in the next chapter. Because of the limitations of the hiden line algorithms for non-manifolds (see section 12.2.1), two fresh new hidden line algorithms were coded. The programs also include extensive suite of routines for converting between the data structures figuring in Chapter 7. It was observed that the commercial graphics and solid modelling packages do not permit an access to the internal data structure. Such an access is vital for efficient matching strategy in vision [Arman and Aggarwal 1990]. This was one of the reasons for embarking on the development of CAD programs rather than buying them off-shelf. But it is felt that development of a full fledged CAD package to match commercial packages with fancy facilities, requires extensive coding which often spans decades of man years, as exemplified by the experience of BUILD solid modeller developed at Cambridge University. So only the bare minimum facilities of the CAD package were implemented.

Coding and debugging has been a taxing job. Most of the algorithms in CAD are simple to conceptualize but implementing[2] them and getting the simple concepts to

[2]An excellent starting point for implementation is [Mantyla 1988]

work is a difficult task. This has been the experience of several others. As an example the sectioning of a solid is trivial to implement physically. Even a butcher knows that all that is required is to slice the solid with a knife and he has a clear mental description of what to expect after sectioning. But when it comes to computer implementation, it is a horrendous task with so many degenerate cases and special cases (e.g. what is the result of sectioning the solid along the top face). The program ought to take all such possibilities into account for robustness. Robustness and handling of the degeneracies has been the subject of active research (e.g. [Hoffmann 1989]). As a result most of the solid modelling operations such as the general purpose union, intersection, are painfully slow to execute.

Chapter 15

Efficient Conversion of B-Reps

In Chapter 4, we considered several CAD data structures. Particularly interesting are the Aspect graph and Spatial data structures. Their major drawbacks are the enoromous size and processing power. With developmennts in parallel computing and falling memory prices, they will become very attractve in the future. Also for an overall perspective, i.e. to cover all the facets of a data structure, we discuss conversion of B-Reps into these two promising representations. We have considred several aspects of data structure including unambiguity and validity. This chapter highlights the fact that such a list of aspects is not comprehensive and there are several additional consideartions to be taken into account.

Convertibility of a data structure is very desirable, since different applications benefit from a different data structure. Also in some cases the input data is avaiable only in certain forms. For example, image processing is very popular for modeling and recognition of objects. Image data needs to be converted into a form suitable for identification. Image data falls into the class of spatial data structures and as discussed before, spatial data is not usable by many applications. We briefly describe

its conversion to B-Rep form, followed by the conversion of B-Rep to Aspect graph.

15.1 Spatial to B-Rep conversion

In general spatial to B-Rep conversion is very difficult and it is much harder with the image data. With the image data we have problems of noise and 2-D to 3-D interpretation. The noise may be due to the dust in the background (e.g. a construction site) or due to the inherent measurment errors in the image intensity quantifier. In either case the problem is coping with spurious and or missing data. Also the image data is a 2-D projection which needs perspective inverson to get a 3-D model. Both these problems: noise and perspective inversion are active research areas and no general solutions exist.

For illustration purposes, we limit ourselves to a specific case of identification of bricks from images. Even this simpler problem needs a great deal of apriori knowledge for tractability. The image data consists of light intensity at each pixel (a pixel is a space subdivison of the extent of the image, usually $512 \times 512 = 262144$ pixels). This is analogous to the volume occupancy represenatation at each cell in the spatial data structure, but in 2-D. The image intensity variation is gradual accross the image except at the boundary of the object, where the change is sudden. The process of detection of the image discontinuities is known as edge detection which basically detects the boundary. This step needs to be followed by a line fitting stage which coalesces the edgels (pixels whose intensity differs greatly from their neighborhood) and approximates them into the boundary of the object. The above description is very sketchy, with many details omitted for brevity. We have proposed an efficient algorithm which combines both the edge and line finder stages, whose details may be

found in Ala, Chamberlain and Ellis [1992a]. An image data of a home brick with the model superimposed is shown in Figure 36.

15.2 B-Rep to Aspect graph conversion

15.2.1 View centred representation

In Part 1, we studied both object and view centered representations. Modelling dictates the matching strategy and its parameters. A viewer-centered approach matches 2-D image features with 2-D projections (stored for each aspect) in contrast to the matching of 2-D features to 3-D model features (e.g. ACRONYM [Brooks 1984]) in the object-centered approach. The viewer-centered approach holds the promise to imitate human beings, since with the future potential of parallel processing, it may be feasible to simultaneously compare the different views with the image. However the known algorithms for computation of aspects of the objects are severely constrained and also prohibitively expensive to compute. For example, the aspect graph algorithm of Gigus and Malik [1990], requires a storage and time of $O(n^8)$, where n is the number of faces of the object. Also the algorithm is limited to orthographic projection only and like many other algorithms has not been implemented. The resulting view data (also known as view potential) is unmanageable and complex.

A characteristic of the B-Rep data structures is that they all require a storage linear in the number of the edges or faces i.e. $O(n)$ (e.g. The Δ data structure proposed by Ala [1992] requires a storage of 6E, E being the number of edges). As mentioned above the viewer centered modelling requires a polynomial (8 and higher degrees) storage and time. In spite of the rich literature in object-centered modelling and availability of efficient data structures no use of them has been made for view

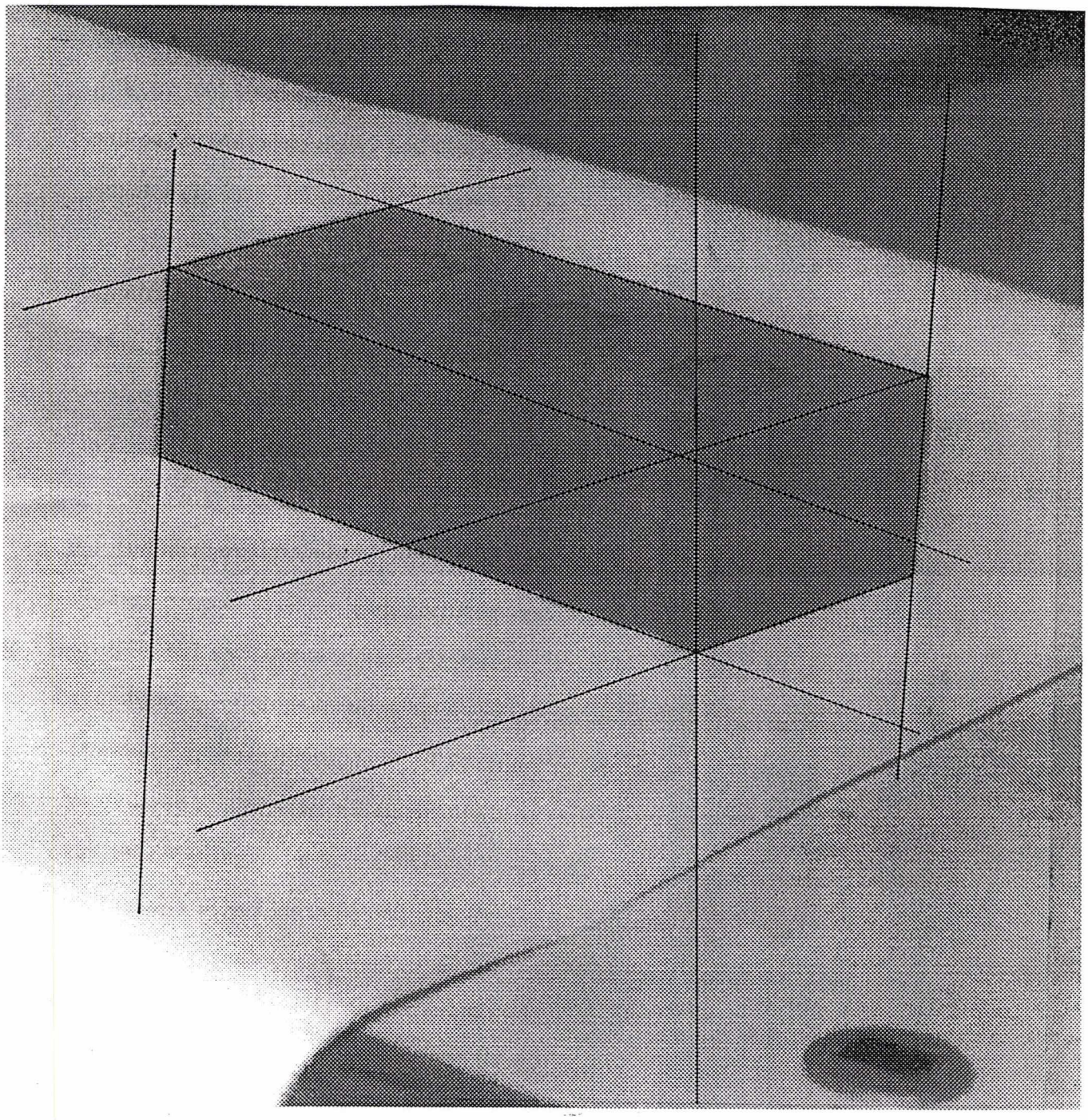

Figure 36: An image of a brick with model lines superimposed

centered modelling. The central theme of this Section is to show how some research results from CAD may be profitably employed in the construction of aspect graphs.

Different types of attributes have been used to define the aspect (by different people). Depending on the application at hand, various topological measures on the projection have been used to define an aspect: the set of visible faces [Stewman and Bowyer 1990, Platinga and Dyer 1986, Ikeuchi 1987], the set of visible edges [Ala, Ellis and Chamberlain 1992b], set of occluding edges [Hebert and Kanade 1985] and graph structure of the line drawing [Gigus and Malik 1990, Platinga 1988]. Since edge extraction is the prevalent method for intensity images, an edge based aspect is preferred to a face based one.

There are two approaches for partitioning the view point space: independent of the object (e.g. predetermined approximation of the view sphere to an icosahedron with 20 cells) and partitioning into maximal regions directly related to the complexity of the object [Gigus and Malik 1990]. The uniform approach typically uses a tesselation of the sphere such as a icosahedron and is easy to compute. The demerit is in selecting the right scale of partition (e.g whether to approximate the view sphere boundary by 100 or a million trianglular facets), the coarse partition may miss out some important views while a fine partition may involve unnecessary cost in computing the millions of views and comparing the views in contiguous regions for merging neighbourhoods with the same aspect. For example, in figure 37, which represents the views of an L-block, from 16 different viewpoints on the tesselated icosahedron, several views are identical. The view points were obtained by subdivison of one (the zeroth, in this case) of the 20 faces of an icosahedron with identifiers ranging from 0 to 11, recursively twice i.e. in the first level four smaller triangles 0-0 to 0-3, each of which give rise to four subdivisions in the second level of recursion, e.g. 0-0 gives rise

to 0-00 to 0-03.

The alternative partitioning approach is complicated and there exists only one general method for all classes of objects under perspective projection by Platinga [1988], while Stewman and Bowyer [1990] restrict to convex polyhedra and Gigus and Malik [1990] restrict to polyhedral objects under orthographic projection. Platinga [1988] computes the visibility volume for each face. For orthographic and perspective the visibilty volume is 4-D (two degrees of freedom in viewing direction i.e. view space, and two degrees of freedom in appearance i.e. image space) and 5-D respectively. Boolean operation of these volumes yields the visibility volume for the whole object. The boundaries are projected onto the view sphere and their intersection points yield the boundaries of the partition.

Platinga [1988] states that the bounds on the size of the aspect graph are also the bounds on the number of aspects. The number of aspects in the convex case is $O(n^2)$ and $O(n^3)$ for orthographic and perspective projections respectively. For a non-convex object they are $O(n^6)$ and $O(n^9)$. For Gigus and Malik [1990] the size of the view data is $O(n^8)$. Thus in both cases: [Platinga 1988] and [Gigus and Malik 1990], the size of data is difficult to handle for real time recognition. The problem must require most of this data to justify the amassing of such a huge data.

15.2.2 An efficient recursive algorithm

Although the formal methods of aspect graph construction date back to the mid-eighties (e.g. [Platinga and Dyer 1986, Stewman and Bowyer 1988, Gigus and Malik 1990]) applications still continue to use the more intutive and simplistic method based on the tesselation of icosahedron (e.g. [Flynn and Jain 1991, Hansen and Henderson

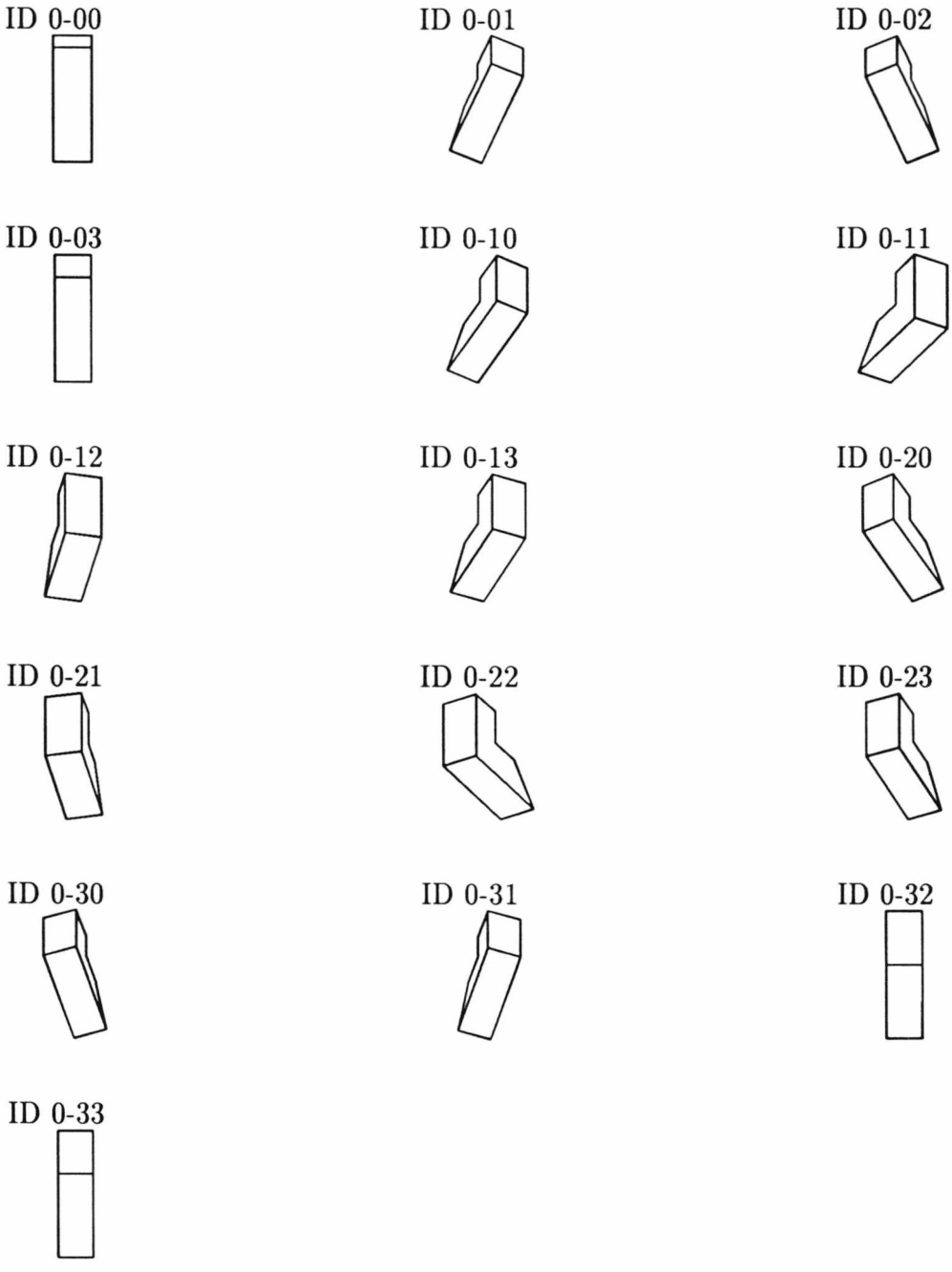

Figure 37: Views of an L-Block on a spherical triangle

1987, Ikeuchi 1987]).

The proposed algorithm seeks cross fertilization of ideas with CAD, specifically the Boundary data structures of the Solid Modeling and Rendering algorithms of graphics. It results in much faster algorithms for aspect graph construction. We propose an efficient method of constructing aspect graphs which requires $O(n)$ space and time for convex objects. Such a simplification has been made possible by the utilization of the B-Rep data. The B-Rep data structure holds incident faces for each of the edges e and vertices v (denoted by $e \rightarrow F$ and $v \rightarrow F$ respectively). From this data it is a straight forward matter to compute the aspect graphs. The list of faces constitute the first level of aspects i.e. where single faces are visible. The same labels, as used for the faces can be used for the aspects in this level. Thus we avoid extra storage for aspect identification. The second level of aspects which have two faces can be labelled by edges, with aspect faces given by their adjacent faces. Similarly for the third level of aspects, the vertex labels and their incident faces suffice. Note that unlike the first two levels, at the third level [1], the number of faces is not constant, being equal to the the number of incident faces e.g. a pyramid with n sides has n each with 3 incident faces and one vertex i.e. the apex, with n faces. Thus we can construct in $O(n)$ time the aspect graph for convex objects under orthographic as well as perspective projection. From the face list of an aspect, the boundaries of the view space can be easily constructed.

Another efficient method for aspect graph construction employs the rendering algorithm (hidden line removal algorithm) and uniformly partitioned sphere for viewpoints. Unlike the naive approach, which merely uses a fixed scale of partitioning,

[1] Higher level aspects have very little probability of occurence for a reasoanble viewing distance and hence are ignored

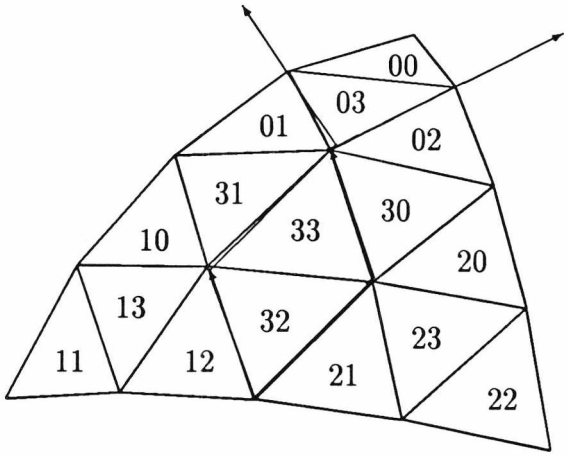

Figure 38: View point partition of a spherical triangle

our approach uses an adaptive partitioning of the tessels. A tessel is recursively subdivided only so long as the views in the partition differ. The recursive divison terminates when the child tessels have identical views. The accuracy acheived is comparable to the exact partitioning. Thus our approach possesses the best of both the worlds: the simplicity of the uniform partioning and the accuracy of the exact aspect graphs. It merges the contiguous regions which have the same aspect, by comparison of the visible edges. It is applicable to any arbitrary polyhedral and curved object and is more efficient in time. The worst case time complexity of a hidden line algorithm is $O(n^2)$, n being the number of edges. Thus the computation of views requires $O(mn^2)$, where m is the number of aspects. Comparisons for coalescing contiguous regions sharing aspects, requires $O(m^2n^2)$ operations. Thus the overall complexity is $O(m^2n^2)$ for the proposed algorithm. The algorithms (e.g. [Gigus, Canny and Seidel 1991, Platinga 1988]) have the worst case time complexity of $O(n^8)$.

The implementation of the above algorithms is near completion and preliminary test runs are very promising. A partitioning of one of the twenty triangles of an

icosahedron into 3 regions of equivalent views is shown in Figure 38. The details of the algorithms may be found elsewhere [Ala et al. 1992b].

15.3 Conclusions

There are several issues in a data structure design and analysis. No thesis can possibly do justice to all of them. We have identified some important issues in this chapter. Thus the chapter aims at highlighing the diversity of considerations in data structure assessment. Although the thesis covered important aspects of B-Rep data structures in depth, there are numerous others which do not even find brief mention. For example, Ikeuchi and Kanade [1988] argues that sensor modelleing is required in conjunction with the object modelling since the appearance of an object is influenced not only by the object properties but also by the sensor characteristics. The two aspects of sensor characteristics are detectability and reliability, which are expressed in configuration space representing the correspondence between the sensor and the object coordinates. A sensor model thus aims to bridge the discrepancy between observed (sensed) data and the actual value from the geometric model.

Part V

Conclusions and Future trends

We summarize the main ideas developed so far, list the main contributions of this work and indicate the main directions in the future and the implementation work left unfinished.

Chapter 16

Conclusions and further work

Section 1, summarizes the thesis and also presents broad conclusions. The next Section, highlights the specific contributions. The last Section, describes possible improvements and puts forward a propasal for a more complete data structure.

16.1 Summary and Conclusions

This thesis concerned with the improvement of efficiency of boundary data structures. Boundary data structures have come a long way from the days when a butterfly shaped data structure (winged edge) was conceived for representation of geometric data for computer vision. Various researchers enhanced the basic set of topological entities from a meager three into at least 11 and the mappings accordingly rose to $11^2 = 121$. This implies that there are 2^{121} ways of structuring the boundary data. This thesis was aimed at filling an important gap in the evolution of boundary data structures i.e. selection methodologies from amongst such a huge number of potential data structures. A related problem of how to estimate the efficiency of existing data structures has also been addressed in great detail.

The thesis began with an exhaustive list of CAD data uses and hypothesized the data structure requirements. We proposed a design methodology for a systematic approach for selection from amongst the multitude of data structures. This methodology, while not only making a description and specification simple to comprehend but also resulted in an efficient data structure which is the most compact amongst all the constant time data structures. We described two approaches for the design of data structures: a separate data structure tailored for each application and a single data structure which is optimal in an overall sense, but may not be optimal for an individual application. Such a global approach, while guaranteeing a constant time for any conceivable application, also ensures the much needed data integrity.

The formal design method has been complemented by an indepth treatment of analysis techniques. We had delved deep into the analysis of boundary data structures. Such an analysis lead to the discovery of startling ideas. More storage does not necessarily imply efficient access. All the previous data structures assumed an unlimited random access memory. Modern computers operate in a virtual memory environment which leads to anomalies. We showed how to estimate the performance in such a virtual memory environment. Incidentally the Δ shaped data structure we proposed performs even better in a virtual memory environment.

The growing trend for more flexibility in manipulating and using CAD data, necessitates usage of non-manifold data structures which were analyzed in depth. The coverage has been extensive and we proposed ways to avoid the overhead associated with a popular data rich non-manifold data structure i.e. Weiler's Radial Edge data structure. A big drawback of the boundary representation is the generation of invalid models i.e. ones that are not manufacturable or physically realizable. In the context of non-manifold modeling, validity considerations assume new dimension, which hitherto

have received little attention. We proposed a practical approach for aiding in the generation of valid models. The visualization algorithm is completely general, it handles all non-manifold conditions and is very efficient since it exploits topological information for local coherence.

Although the main emphasis in this thesis has been on the data structures, often big improvements result from an analysis of algorithms. We considered some important model manipulation algorithms, and based on the observation that the majority of operations involve null intersection or difference between two participant objects, proposed and implemented an algorithm which is many magnitudes faster than the conventional boolean algorithms. The Δ data structure has been implemented in boolean and hidden line algorithms and has been proved to be more efficient in practice. Finally, we degressed to the conversion of boundary data structures, the main idea being to point out the multifaceted nature of data structures.

16.2 Contributions

The following is the list of specific contributions of this thesis.

1. Quantitative methods for evaluation of boundary data structures

2. Optimality of boundary data structures and their associated algorithms

3. A systematic methodology for the design of boundary data structures

4. Performance anomolies of boundary data structures

5. Introdction of a reduced access and memory efficient data structure

16.3 Future Work

16.3.1 Issues outstanding

There is a great scope for major improvements in the thesis. The following is a brief description of major issues which deserve further study.

Extension to curved geometry

The performance anomolies studies in this thesis mainly investigate topology. Informal estimates of geometry size range from 2 to 3 times topology size. Also many of the curved surface intersections are compute intensive. For example, Crocker and Reinke [1991] report that topological evaluations account for usually less than 30 percent whereas the face/face intersections account for more than 50 percent of the of the total evaluation time. It will be interesting to test for performance in a full non-linear domain including free form surfaces. The algorithms have been implemented only for polyhedral models. Currently free form surfaces and quadratic surfaces are being used but with polygonal approximation. An extension for curved surface domain is underway.

Data structure design assisstant

An expert system to incorporate and capture the wealth of knowledge in data structures could be very useful as a data structure design assistant.

Extension to higher dimensional problems

The design methodology is limited to the 3-D world only. There have been proposals for extension of B-Reps for n-Dimensional problems by Rossignac and Requicha [1991]

and Ferrucci and Paoluzzi [1991] and for inclusion of non-geometric information such as the laws of physics by Terzopoulous [1991] and Greenberg [1991]. We need a suitable extension for the design methodology.

Statistical analysis of the usage pattern of modeling tools

In the design, an accurate estimate of the query frequencies requires extensive data gathering and statistical analysis over a wide variety of user applications. To a large extent the thesis relied on the an informal estimate of the query and modeling algorithm usage pattern. A more systematic requirements analysis than the one described in Chapter 2 will go a long way in making the special purpose optimization approach attractive.

Further experimentation into virtual memory and database studies

The data base empirical studies reported in chapter 7 were carried out in basically non-CAD data bases. Even then there seems to be no convergence of results as to the presence of locality or sequentiality in data base reference behavior. All that can be said is that clearly discernible patterns are absent in them and it is in contrast to strong locality exhibited by program memory references. Further research is required to know the precise behavior of CAD data-base references. Also whether the nature of the nine fundamental topological queries. and the data structure reveal the presence of sequentiality or locality is to be ascertained.

16.3.2 A proposal for a more general data structure

Need for a general data structure

So far we have only two categories of CAD data: topological data which we addressed at length in this thesis and the geometric data from which it is theoretically possible to deduce the whole of topological data. The topological data is limited to the adjacencies and ordering of the primitive topological elements. Topology is not enough for solving a great variety of problems (e.g. computation of aspect graphs, organization into perceptually significant groups). If we need such additional information we need to manipulate the geometric data which is often cumbersome and error prone. Several applications frequently need access to data other than the primitive topological data.

Applications

There exist several methods for model based recognition: to detect the position and orientation (three degrees of freedom, each of translation and rotation in the three cartesian axes) we need to match the salient features from the image with the CAD model features. A naive application of this method can be very slow: we can use perceptual grouping techniques [Lowe 1985] to reduce the search space of the possible view points. Hence it is common to devise matching strategies based on perceptual groups such as parallel lines, collinear lines, concentric circles. Also there is a recent trend in the geometric modeling to provide constraint based parametric modeling [Shimada, Numao, Masuda and Kawabe 1989] capabilities based on such perceptual groups.

Proposal for augumentation of B-Reps

So we propose an additional organization of geometry, similar to binary space partitions of Naylor [1990] and gaussian sphere of Ikeuchi [1987]. This data structure is more general than the B-Rep (referred to G-Rep, from now on) and could be useful in feature extraction, hidden line etc. Because the explicit encoding records collinearity, concurrency, and parallelism between the topological entities, problems due to degeneracy and floating point arithmetic could be avoided and robustness is easy to guarantee. The benefits accruing from such an intermediate grouping between the conventional geometry and topology levels, could easily outweigh the cost of maintaining the additional data. We could also use additional groupings based on *above, below, to the left or right* at a higher level organization. A related model proposed by Biederman [1987] has such 58 such relations between two objects (known as geons) including verticality, relative size, centering and surface size at join surface. B-Rep is an evaluated model of CSG: for the G-Rep further evaluation is necessary. Similar to the fundamental topological queries, we must work out a minimum set of fundamental set of arrangements of higher level entities.

Example computations

An example is the extraction of parallel lines by maintaining a single copy of a particular geometry (direction cosines in the case of straight edges) and indexing all parallel edges to the same geometric record.

The literature is also replete with examples. Faux (see [Pratt 1990]) preferred surface based descriptions rather than face based, since faces on a single surface are likely to be functionally related. When a new face is created its geometry is not

duplicated if the geometry already exists i.e. redundancy of geometry is avoided. Crocker and Reinke [1991] employ this approach in their boolean algorithm. The B-Rep provides too detailed information. So feature sub structuring in terms of higher groupings was advocated by Faux (see [Pratt 1990]): e.g. instead of 6 faces, 3 sets of parallel faces of 2 each are used to describe a cube.

Another example is the deduction of relative size information based on the bounding box associated with each object. Also the transformations associated with each object could be used to deduce some locational order between the objects. Note that the computation of G-Rep would be done off-line, so its derivation step is not critical.

Issues worth pursuing

The G-Rep once computed could be utilized in several algorithms to cut down their execution time and enhance reliability. A new approach to algorithmic design is then possible by expressing into such fundamental queries. Abstraction in the formulation and modularization of algorithms is facilitated. For example, consider the now familiar non-manifold vertex information. We can either use a manifold data structure and compute the non-manifold vertex incidence through geometry or use an explicit non-manifold data structure which has pre-recorded incidence information. Similarly the G-Rep has a symbolic information at a higher level. What constitutes a minimum G-Rep and the fundamental spatial relationships is an issue worthy of pursuit. Other issues: e.g. if the G-Rep has only parallel or perpendicular lines, how do we compute all lines at some angle, say 60 degrees, from a given line are also to be investigated. It is thus essential to investigate the commonly occurring groups of geometry and spatial relationships. In a way the augmented data structure also addresses the problem of algorithmic complexity (e.g. usage of boxing information reduces the hidden line

computation complexity from quadratic to linear).

Bibliography

Ala, S. R.: 1991, Design methodology of boundary data structures, *International Journal of Computational Geometry - Special Issue on Solid Modeling* **1**(3), 207–226. Preliminary version In the Proceedings of ACM Symposium on Solid Modeling and CAD/CAM Applications,1991.

Ala, S. R.: 1992, Performance anomolies in boundary data structures, *IEEE Computer Graphics and Applications* **12**(2), 49–58.

Ala, S. R. and Chamberlain, D.: 1991, An efficient boundary data structure for cad/cam, robotics and factories of future, *6 th International Conference on CAD/CAM, Robotics and Factories of Future.*

Ala, S. R., Chamberlain, D. and Ellis, T. J.: 1992a, Real time inspection of masonry units, *4th International Conference on Image Processing and its applications*, Maastricht, The Netherlands.

Ala, S. R., Ellis, T. J. and Chamberlain, D.: 1992b, An efficient recursive algorithm for aspect graph construction, Manuscript under preparation.

Arman, F. and Aggarwal, J. K.: 1990, Object recognition in dense range images using a cad system as a model base, *NATO proceedings*, pp. 1858–1863.

Baer, A., Eastman, C. and Henrion, M.: 1979, Geometric modeling: a survey, *Computer-aided design* **11**(5), 253–272.

Baumgart, B. G.: 1975, A polyhedron representation for computer vision, *AFIPS National Computer Conference*, pp. 589–596.

Bentley, J. L.: 1986, *Programming Pearls*, Addison Wesley.

Besl, P. J. and Jain, R. C.: 1985, Three dimensional object recognition, *computer* **17**, 75–145.

Bhanu, B. and Ho, C. C.: 1987, Cad based 3d object representation for robot vision, *Computer* **20**(8), 19–36.

Biederman, I.: 1987, Recognition-by-components: A theory of human image interpretation, *Psychological Review.*

Braid, I. C.: 1980, Notes on a geometric modeler, *Technical Report CAD group Document no. 101*, University of Cambridge, UK.

Brooks, R. A.: 1984, *Model based computer vision*, UMI Research Press, Michigan.

Bruzzone, E., Defloriani, L. D. and Puppo, E.: 1989, Manipulating 3-d triangulations, *in* W. Litwin and H. J. Schek (eds), *Foundations of data organization and algorithms - 3rd international Conf. FODO 1989 Proceedings*, Springer Verlag, pp. 339–353.

BYG: 1992, *Grasp User's and Reference manuals.*

Cameron, S. J.: 1984, *Modelling Solids in Motion*, PhD thesis, University of Edinburgh.

Chamberlain, D., Ala, S. R., Watson, J., Reilly, J. and Speare, P. S.: 1992, Masonry construction by an experimental robot, *9 th International Symposium on Automation and Robotics in Construction*, Japan.

Chamberlain, D., Speare, P. S. and Ala, S. R.: 1991, Progress in a masonry tasking robot, *8 th International Symposium on Automation and Robotics in Construction*, Stuttugart, Germany.

Choi, Y.: 1989, *Vertex-based boundary representation of non-manifold geometric models*, PhD thesis, Carnegie-Mellon University, Pittusburgh.

Coffmann, E. G. and Varian, L. C.: 1968, Further experimental data on the behaviour of programs in a paging environment, *Communications of the ACM* **11**(7), 471–474.

Crocker, G. A. and Reinke, W. F.: 1991, An editable non-manifold boundary representation, *IEEE Computer Graphics and Applications* **11**, 39–51.

Datavision, M.: 1992, Euclid solid modeller product brochures, Published by Matra Datavision.

De Floriani, L. and Falcidieno, B.: 1988, A hierarchical boundary model for solid object representation, *ACM Trans. on Graphics.*

Dietrich, W. C., Nackman, L. R., Sundaresan, C. J. and Gracer, F.: 1989, Tgms: An object-oriented system for programming geometry, *Software Practice and experience* **19**(10), 979–1013.

Dobkin, P. D. and Laszlo, M. J.: 1987, Primitives for the manipulation of three-dimensional subdivisions, *Proceedings of the third Symposium on Computational Geometry.*

Eastman, C. M.: 1982, *Introduction to Computer aided design - Course Notes*, Carnegie Mellon University.

Falcidieno, B. and Giannini, F.: 1989, Automatic recognition and representation of shape-based features in a geometric modeling system, *Computer Vision Graphics and Image Processing* **48**, 93–123.

Ferrucci, V. and Paoluzzi, A.: 1991, Extrusion and boundary evaluation for mulitdimensional polyhedra, *Computer-aided design* **23**(1), 40–50.

Flynn, P. J. and Jain, A. K.: 1991, Cad-based computer vision: From cad models to relational graphs, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(2), 114–132.

Gigus, Z. and Malik, J.: 1990, Computing the aspect graph for line drawings of polyhedral objects, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(2), 113–122.

Gigus, Z., Canny, J. and Seidel, R.: 1991, Efficiently computing and representing aspect graph of polyhedral objects, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(6), 542–551.

Greenberg, D. P.: 1991, More acuurate simulations at faster rates, *IEEE Computer Graphics and Applications* pp. 23–29.

Guibas, L. and Stolfi, J.: 1985, Primitives for the manipulation of general subdivisions and the computation of vornoi diagrams, *ACM Transactions on Graphics* **4**(2), 74–123.

Gursoz, E. L., Choi, Y. and Prinz, F. B.: 1990, Vertex-based representation of non-manifold boundaries, *in* M. J. Wozny, J. U. Turner and K. Preiss (eds), *Geometric Modeling for Product Engineering*, North-Holland, pp. 107–130.

Gursoz, E. L., Choi, Y. and Prinz, F. B.: 1991, Boolean set operations on non-manifold boundary representation objects, *Computer-aided design* **23**(1), 33–39.

Hanrahan, P. M.: 1985, *Topological shape models*, PhD thesis, University of Wisconsin-Madison, U. S. A.

Hansen, C. and Henderson, T.: 1987, Cagd-based computer vision, *IEEE workshop on Computer Vision*, pp. 100–105.

Harary, F.: 1972, *Graph Theory*, Addison Wesley.

Hatfield, D. J.: 1972, Experiments on page size, program access patterns and virtual memory performance, *IBM J. Res. and Develop.*

Hebert, M. and Kanade, T.: 1985, The 3-d profile method for object recognition, *Proc. IEEE Comput. Soc. Conf. Computer Vision and Pattern Recognition*, pp. 458–463.

Hewlett-Packard: 1992, Hp solid modeller product brochures, Published by Hewlett-packard.

Hoffman, C. M., Hopcroft, J. E. and Karasick, M. S.: 1988, Towards implementing robust geometric computations, *ACM Sym. Computational Geometry.*

Hoffmann, C. M.: 1989, *Geometric and Solid Modeling: An Introduction*, Morgan Kaufmann.

Ikeuchi, K.: 1987, Generating an interpretation tree from a cad model for 3d-object recognition in bin-picking tasks, *International Journal of Computer Vision* **1**, 145–167.

Ikeuchi, K. and Kanade, T.: 1988, Modelling sensors and applying sensor model to automatic generation of object recognition program, *DARPA vision workshop*, pp. 697–710.

Johnson, R. H.: 1986, *Solid Modeling: A state-of-the-art report*, second edn, CAD/CIM Alert, Management Roundtable, Inc.

Kalay, Y. E.: 1983, A relational database for nonmanipulative representation of solid objects, *Computer-aided design* **15**(5), 271–276.

Kalay, Y. E.: 1989, The hybrid edge: a topological data structure for vertically integrated geometric modeling, *Computer-aided design* **21**(3), 130–140.

Karasick, M. and Leiber, D.: 1991, Schemata for interrogating solid boundaries, *Proceedings of ACM Symposium on Solid Modeling and CAD/CAM Applications*, pp. 25–34.

Karasik, M.: 1988, *On the representation and manipulation of rigid solids*, PhD thesis, McGill University, Canada.

Karasik, M.: 1989, The same object problem for polyhedral solids, *Computer Vision, Graphics and Image Processing* **46**, 22–36.

Kawabe, S., Shimada, K. and Masuda, H.: 1989, A framework for 3d modeling: Constraint-based description and non-manifold geometric modeling, *in* T. Sata (ed.), *Organization of engineering knowledge for product modeling in computer integrated manufacturing in Procs. of 2nd Toyota Conf.*, Elsevier, pp. 325–357.

Kearns, J. P. and DeFazio, S.: 1989, Diversity in database reference behavior, *Performance Evaluation Review* **15**(1), 11–19.

Koenderink, J. J. and van Doorn, A. J.: 1979, The internal representation of solid shape with respect to vision, *Biol. Cybernet.* **32**, 211–216.

Kripac, J.: 1985, Classification of edges and its applications in determining visibility, *Computer-aided design* **17**(1), 30–36.

Laidlaw, D. H., Trumbore, W. B. and Hughes, J. F.: 1986, Constructive solid geometry for polyhedral objects, *SIGGRAPH '86*, Dallas, pp. 161–170.

Lowe, D. G.: 1985, *Perceptual organization and visual recognition*, Kluwer Academic publishers.

Luo, Y. and Lukacs, G.: 1991, A boundary representation for form features and non-manifold solid objects, *Proceedings of ACM Symposium on Solid Modeling and CAD/CAM Applications*, pp. 35–44.

Magrabhi, S. M. and Griffiths, J. G.: 1989, Removal of hidden lines by recursive subdivision, *Computer-aided design* pp. 570–576.

Mantyla, M.: 1988, *An Introduction to Solid Modeling*, Computer Science press.

Marr, D.: 1982, *Vision*, W. H. Freeman.

Masuda, H., Shimada, K., Numao, M. and Kawabe, S.: 1989, A mathematical theory and applications of non-manifold geometric modeling, *in* F. L. Krause and H. Jansen (eds), *Advanced Geometric Modeling for Engineering Applications*, North-Holland, pp. 78–92.

McKenna, M.: 1987, Worst-case optimal hidden-surface removal, *ACM Transactions on Graphics* pp. 19–28.

Murabata, S. and Higashi, M.: 1990, Non-manifold geometric modeling for set operations and surface operations, *IFIP/RPI Geometric Modeling Conference*, Rensserlaerville, N. Y.

Naylor, B.: 1990, Binary space partitioning trees as an alternative representation of polytopes, *Computer-aided design* **22**(4), 250–252.

Nurre, J., Hall, E. L. and Ronig, J. J.: 1988, Acquring simple patterns for surface inspection, *DARPA Proceedings*, pp. 586–591.

Ottomann, T., Widmayer, P. and Wood, D.: 1985, A worst case efficient algorithm for hidden-line elimination, *International Journal of Computer Mathematics* **18**(2), 93–119.

Paoluzzi, A., Ramella, M. and Santarelli, A.: 1989, Boolean algebra over linear polyhedra, *Computer-aided design* **21**(8), 474–484.

Platinga, H.: 1988, *The asp: A continuous viewer centered representation for computer vision*, PhD thesis, Computer Science Department, University of Wisconsin-madison.

Platinga, W. H. and Dyer, C. R.: 1986, An algorithm for constructing the aspect graph, *Proc. 27th Symp. Foundations of Computer Science*, pp. 123–131.

Porter, S.: 1991, Winds of change, *Computer Graphics World* pp. 34–40.

Potter, C. D.: 1992, Solid modeling: Kernel-style, *Computer Graphics World* pp. 73–79.

Pratt, M. J.: 1987, Conceptual design of feature-oriented solid modeler, *Technical Report Draft Document 3B*, GE Corporation R&D.

Pratt, M. J.: 1988, Synthesis of an optimal approach to form feature modeling, *ASME International Conference on Computers*, San Fransico.

Pratt, M. J.: 1990, Aspects of form feature modelling, *in* D. Roller (ed.), *Geomtetric Modelling: Methods and Applications*, Springer-Verlag.

Pratt, M. J. and Wilson, P. R.: 1985, Requirements for the support of form features in a solid modeling system, *Technical Report R-85-ASPP-01*, CAM-I Inc., Arlington, Texas.

Preparata, F. P. and Shamos, M. I.: 1985, *Computational geometry: an Introduction*, Springer Verlag.

Reingold, E. M., Nievergelt, J. and Deo, N.: 1977, *Combinatorial algorithms: Theory and practice*, Prentice-Hall.

Requicha, A. A. G. and Voelcker, H. B.: 1983, Boolean operations in solid modelling: Boundary evaluation and merging algorithms, *IEEE Computer Graphics and Applications* **3**(7), 30–44.

Rodriguez-Rossel, J.: 1976, Empirical data reference behavior in data base systems, *Computer* **9**(11), 3–13.

Rosenfeld, A.: 1987, Recognising unexpected objects: A proposed approach, *Int. J. Pattern Recongition and Artificial Intelligence* **1**(1), 71–74.

Rosenthal, D.: 1989, More haste-less speed, *Proceedings of EUUG Spring'89*, pp. 123–130.

Rossignac, J. R. and Requicha, A. A. G.: 1991, Constructive non-regularized geometry, *Computer-aided design* **23**(1), 21–32.

Schelechtendahl, E. G. (ed.): 1988, *Neutral File for CAD Geometry, Version 3. 3*, Springer-Verlag.

Shah, J. J.: 1991, Assessment of features technology, *Computer-aided design* **23**(5), 331–343.

Shimada, K., Numao, M., Masuda, H. and Kawabe, S.: 1989, Constraint-based object description for product modeling, *in* F. Kimura and et.al. (eds), *Computer applications in production engineering - IFIP 89*, North Holland, pp. 95–105.

Spatial-Technology: 1991, *ACIS Geometric Modeler: Technical Overview.*

Staudhammer, J.: 1991, 10th anniversary issue guest editor's introduction: Computer graphics - toward the next millennium, *IEEE Computer Graphics and Applications* **11**, 21–23.

Stewman, J. H. and Bowyer, K. W.: 1988, Creating the perspective projection aspect graph of polyhedral objects, *Procs. of the 2nd International Conference on Computer Vision*, pp. 494–500.

Stewman, J. H. and Bowyer, K. W.: 1990, Direct construction of the perspecive projection aspect graph of convex polyhedra, *Computer Vision, Graphics, Image Processing* **51**, 20–37.

Stobart, R. K. and Dailly, C.: 1985, The use of simulation in the off-line programming of robots, *in* J. Billingsley (ed.), *IEE Control Engg. Series: Robots and automated manufacture*, IEE.

Stroud, I.: 1990, Modeling with degenerate objects, *Computer-aided design* **22**(6), 344–351.

Terzopoulous, D.: 1991, Visual modeling, *BMVC91*, Springer-Verlag.

Wagner, H. M.: 1975, *Principles of Operations research*, Prentice Hall.

Weiler, K. and McLachlan, D.: 1991, Selection sets and filters in geomteric modeling and their application in a non-manifold environment, *in* J. Turner, J.penga and M. J. Wozny (eds), *Product Modeling for Computer-Aided design and Manufacture*, North-Holland, pp. 117–139.

Weiler, K. J.: 1985, Edge-based data structures for solid modeling in curved-surface environments, *IEEE Computer Graphics and Applications* **5**(1), 21–40.

Weiler, K. J.: 1986, *Topological Structures for Geometric Modeling*, PhD thesis, Rensserelaer Polytechnic Institute, N. Y.

Weiler, K. J.: 1988, The radial edge structure: A topological representation for non-manifold geometric boundary modeling, *in* M. J. Wozny, H. W. Laughlin and J. L. Encarnacao (eds), *Geometric Modeling for CAD Applications*, North-Holland, pp. 3–36.

Wilson, P. R.: 1988, Data transfer and solid modeling, *in* M. J. Wozny, H. W. McLaughlin and J. L. Encarnacao (eds), *Geometric Modeling for CAD Applications*, Elsevier Science, pp. 217–249.

Woo, T. C.: 1985, A combinatorial analysis of boundary data structure schemata, *IEEE Computer Graphics and Applications* **5**(3), 19–27.

Woo, T. C. and Wolter, J. D.: 1984, A constant expected time, linear storage data structure for representing three-dimensional objects, *IEEE Transactions on Systems, Man, and Cybernetics* **14**(3), 510–515.

Wu, S. T.: 1989, Towards a unified data scheme for geometrical representation, *in* F. Kimura and et.al. (eds), *Computer applications in production engineering - IFIP 89*, North Holland, pp. 259–266.

Yamaguchi, F. and Tokieda, T.: 1985, Bridge edge and triangulation approach in solid modeling, *in* T. L. Kunii (ed.), *Frontiers in computer graphics*, Springer Verlag.

Yamaguchi, Y., Kobayashi, K. and Kimura, F.: 1991, Geometric modeling with generalized topology and geometry for product engineering, *in* J. Turner, J. Penga and M. Wozny (eds), *Product Modeling for Computer-Aided Design and Manufacturing*, North-Holland, pp. 97–115.