THE UNIVERSITY OF CALGARY

Polygon Mesh Modelling for Computer Graphics

BY

Jeffrey B. Allan

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

September, 1988

ISBN 0-315-46553-0

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled, "Polygon Mesh Modelling for Computer Graphics" submitted by Jeffrey B. Allan in partial fulfillment of the requirements for the degree of Master of Science.

Brian Wyvill.

Supervisor, Dr. Brian Wyvill
Department of Computer Science

Ian H Witten

Dr. Ian Witten
Department of Computer Science

B MacDonald.

Dr. Bruce MacDonald
Department of Computer Science

Dr. Doug Norrie
Department of Mechanical Engineering

Date 27/Sept./1988

Abstract

Many different geometric modelling techniques are used for computer graphics. They are difficult to evaluate and compare due to their number and variety. The polygon mesh is a widely used geometric modelling representation, often used to represent digitised data and approximations to curved surfaces. In this thesis polygon mesh techniques are explored and evaluated in comparison with other commonly used modelling methods in computer graphics. In order to make such an evaluation, a set of criteria is first proposed to take account of the design requirements for these methods.

Current research indicates that manipulation of polygon meshes as a modelling technique in its own right is not common. A new methodology for directly manipulating polygon meshes, called polygon mesh modelling, is described, followed by an implementation of the methodology and resulting models which are used as examples. The methodology is evaluated based on the criteria developed, and compared to the other modelling techniques surveyed. Polygon mesh modelling is found to be a viable modelling technique, based both on the analysis, and on the examples.

Acknowledgements

I would like to acknowledge my supervisor, Brian Wyvill, for his support and guidance over the course of this work.

Thanks to all those in the Graphicsland Research Group and the Software Research and Development Group for their technical and moral support.

The careful editorial feedback from Anja Haman, Dan Freedman, and Mike Bonham on early drafts was a great help.

To my girlfriend, fiance, and now wife, Theresa Voigt, who gave me the confidence and strength needed to get over many of the obstacles of this degree, I extend my deepest love and gratitude.

This work is dedicated to the glory of God.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Computer Graphics

Computer graphics is a broad field covering all forms of image generation by computers. Computer generated images vary in style and complexity from simple, two dimensional (2D) pie charts to highly detailed and realistic three dimensional (3D) images and animations. Applications of computer graphics include the visualisation of scientific, economic, and other data, advertising, simulation (eg. flight), and entertainment. As a research field, 3D computer graphics can be divided into three major areas: modelling, animation, and rendering.

Modelling is the process of building geometric descriptions of 3D objects from which images can be generated. Mathematical descriptions are used, including numbers, equations, and the relationships among them. Modelling embodies the tasks of creating, modifying, and integrating these mathematical models. A model can be as simple as a sphere or as complex as a human figure or a detailed city. It may represent real or imaginary objects or even non-physical ideas or abstractions. The modelling process is a necessary prerequisite for animation and rendering.

Animation is the specification of changes to one or more models or to the point of view over time [Thalmann *et al* 85]. The simplest form of animation is to change the point of view while the models being viewed remain static. It is more difficult

to change the location, size, or orientation of models. It is most difficult to change the shape of models, or other attributes, such as colour. Modelling requirements for animation are discussed in Section 2.2.9.

Rendering is the process of generating an image from a set of models [Thalmann *et al* 87]. There are several algorithms used for rendering, varying in complexity and realism from wire frame drawings to algorithms which simulate natural phenomena (eg. clouds [Gardner 85]), or inter-object light interaction (eg. radiocity [Greenberg *et al* 86]). Modelling requirements for rendering are discussed in Section 2.2.10.

This thesis examines some of the most commonly used 3D modelling techniques and presents a methodology for polygon mesh modelling. Animation and rendering will be addressed only to the extent that they are related to modelling.

1.2 Terminology

A *modelling technique* consists of three components: the *description* supplied by the designer (also called the *representation*), a number of *manipulation methods* for altering the description, and the resulting *model* produced from the description. An *object* is the actual (physical or imaginary) entity being modelled. So, a description specifies a model which represents an object.

The term *modelling technique* should be distinguished from *modelling system* which is an implementation of one or more modelling techniques, together with other support systems such as data structures and a user interface. A modelling system may embody several modelling techniques.

The term *designer* is used to refer to the person building the model. This avoids the confusion over the term *modeller* which could refer either to the designer or to the modelling system.

All other special terms used will be set in italics when they first occur. They will be defined at that first occurrence or shortly thereafter.

1.3 Computer Graphics Modelling versus CAD

Geometric modelling is a broad field encompassing Computer Aided Design (CAD) [Gardan/Lucas 84] and Computer Graphics (CG). Modelling for computer graphics (*CG modelling*) is closely related to CAD. Both processes involve the communication of design ideas from the designer to the modelling system. Both create and manipulate mathematical models of objects or ideas. There is an important distinction, however. For CAD, the final product is the physical realisation of the model, or the detailed plans for producing it. For computer graphics the final product is the rendered image of the model.

This distinction is reflected in the methodology for CG modelling. The model is only required to be accurate in appearance; the underlying form is unimportant. So, for example, a house may not require any internal walls if it is only to be viewed from outside; surface details can be simulated at rendering time as a texture, rather than being modelled explicitly. Furthermore, since CG modelling is only concerned with images, the model may represent a physically impossible object or an idea.

There are some areas of overlap between CAD and CG modelling. Some newer CAD systems include higher realism rendering capabilities (eg. [Prime 86]). CG

models are beginning to require more CAD style information as computer animation moves toward physical simulation [Forsey/Wilhelms 88].

1.4 Requirements of CG Modelling

The general requirements for CG modelling are:

1. That any desired model can be *created and modified.*
2. That creating and modifying models is sufficiently *easy.*
3. That the resulting models are *useful.*

Creation and modification of models requires the ability to create, combine, alter, delete, duplicate, and substitute components of models and their various attributes (eg. colour). Unlike CAD, labels and dimensions do not need to be applied to the model.

The *ease* with which the model is created and modified depends on three elements: the modelling system, the designer, and the modelling technique. A modelling system comprises a user (designer) interface, underlying data structures, interaction techniques, and graphics hardware. Each designer has different levels of expertise, familiarity, experience, dexterity, memory, and cognitive skills. Each modelling technique has different descriptions, manipulation methods, and properties.

The *uses* to which the resulting models can be put include: combining them with other models, animating them, rendering images of them, storing them for later use, and extracting geometric and other information from them.

1.5 Thesis Organisation

Chapter 2 proposes a set of criteria for evaluating CG modelling techniques. These criteria are based on previous characterisation work carried out in the field of CAD.

Chapter 3 classifies the modelling techniques commonly used for computer graphics into a taxonomy and surveys a sample of them. The techniques are each evaluated using the criteria presented in Chapter 2.

Chapter 4 presents a new modelling methodology for polygon mesh modelling. Whereas polygon meshes have been used mainly for digitised data or as an intermediate representation, this methodology formalises the process of directly manipulating an object represented by a polygon mesh.

Chapter 5 gives the details of an implementation of a polygon mesh modelling system called DELTA which was developed as part of the research for this thesis.

Chapter 6 presents the results of this research in four parts. First, the polygon mesh modelling technique is analysed according to the criteria presented in Chapter 2. Second, the analysis of each modelling technique surveyed in Chapter 3, and of polygon mesh modelling are summarised and discussed. Third, several examples of objects modelled with DELTA are presented. Finally, polygon mesh modelling is compared to the other techniques.

Chapter 7 presents the conclusions of this research. Suggestions for further research are offered.

Chapter 2

EVALUATING CG MODELLING TECHNIQUES

In order to examine CG modelling techniques objectively, some metric for evaluation and comparison is required. A standard set of criteria could be used for this purpose. Both [Newman/Sproull 79] and [Foley/van Dam 83] review a small selection of modelling techniques, but neither offers a systematic evaluation or comparison. Section 2.2 proposes a set of criteria for CG modelling techniques based on the work summarised below.

2.1 Related Work

Solid modelling is a form of geometric modelling which is used in both CAD and computer graphics. Classification and evaluation work done in the field of solid modelling will be used as a basis for this work. [Mantyla 88, p.50] presents a list of properties for solid model representations which are summarised below.

Expressive Power This is the class of objects which can be modelled using a representation scheme.

Validity A solid model is a connected set of points in space. It is considered valid if it is bounded (finite), non-deforming (rigid), and contains no isolated points, lines, or faces. A representation scheme is *syntactically valid* if every representation designates a valid solid.

Unambiguity and Uniqueness A solid modelling system maps representations into solids. *Unambiguity* means that there are no one-to-many mappings from

the space of representations to the space of solids. *Uniqueness* means that there are no many-to-one mappings. Unambiguity is necessary for a representation scheme to be deterministic. Uniqueness is necessary to compare two solids by comparing their representations.

Description Language Solid modelling systems use description languages to communicate representations to other systems (eg. finite element analysis systems). This property refers to whether a representation is self-contained or requires conversion to another representation to be used.

Conciseness The amount of computer storage required by a model depends on the representation scheme. This property is especially important for practically interesting solids.

Closure of Operations Models are built through a process of operations performed on them. A representation is *closed* relative to a particular operation if validity is preserved when that operation is performed.

Computational Ease To be useful, solid models must allow certain computations to be performed on them. These include rendering, determination of physical properties (eg. mass, surface area), interference relations with other models, and generation of manufacturing instructions.

Applicability The kinds of applications to which a representation scheme are applicable depend largely on the computations to which it is well suited.

These solid modelling properties form the basis for the CG modelling criteria presented next. They differ to the extent that CG modelling has different requirements from other areas of geometric modelling in which solid modelling is used, especially CAD (see Section 1.3).

2.2 Criteria for CG Modelling Techniques

The criteria for CG modelling techniques proposed in this section will be used to evaluate the various modelling techniques detailed in Chapter 3. Where they differ significantly from Mantyla's properties for solid model representations, the differences

are explained. Table 2.1 lists Mantyla's properties for solid model representations and the corresponding criteria for CG modelling techniques.

Properties for Solid Model Representations	Criteria for Computer Graphics Modelling Techniques
Expressive Power	1. Expressive Power 2. Precision and Resolution
none	3. Representational Fidelity
Validity Closure of Operations	4. Geometric Consistency
Unambiguity and Uniqueness	none (see Section 2.2.11)
Description Language	5. Convertability
Conciseness	6. Space Efficiency
Computational Ease	7. Computational Efficiency 8. Manipulation Speed
Applicability	9. Applicability to Animation 10. Applicability to Rendering

Table 2.1: Properties for Solid Model Representations versus Criteria for CG Modelling Techniques

2.2.1 Expressive Power

The class of models which can be described using a given technique is referred to as its *expressive power*. There are several dimensions along which to classify models. The following classifications are relevant to CG modelling:

1. Solid or Surface
2. Topological Class
3. Continuity

Solid or Surface. The first classification is between solid models and surface models. Solid models represent contiguous regions of space. Surface models, on the

other hand, only contain information about their boundaries. They may represent closed surfaces which are either self-intersecting or bounding. A *bounding surface* is one which encloses a finite volume of space.

Topological Class. Objects can be classified by the number of holes they have through them. For bounding surfaces and solids, the number of holes through an object is called its *genus.* For open surfaces, a hole is called a *break.*

Continuity. The continuity of an object is a measure of how smoothly its surface changes. The terminology of mathematical functions is used to describe continuity. A mathematical function with no breaks in it is said to have continuity of position (C^0). A function has continuity of slope (C^1) if its derivative is continuous, and it has continuity of curvature (C^2) if its second derivative is continuous. A C^1 discontinuity in a 3D surface represents an edge or corner. A C^2 discontinuity represents a change in the rate of curvature. Higher order continuity is usually not considered for computer graphics.

2.2.2 Precision and Resolution

Precision is a measure of how precisely the designer can specify geometric properties for a model, such as dimensions and angles. Note that only the modelling technique itself is of concern, not the modelling system since a modelling system can provide facilities for precision if and only if the underlying modelling technique supports it.

Resolution refers to the ability to work at both coarse and fine levels of detail. The designer would like to work at as fine a level of detail as he requires without having to contend with too much detail when he works at a coarse level. The designer might want different parts of a model to have different resolution (eg. a highly detailed

portion of an otherwise simple object), or he might want to change the resolution of a model over time (eg. increase resolution to improve rendering quality).

2.2.3 Representational Fidelity

Mantyla's properties contain no measure of how easily a modelling technique can be used. Section 1.4 lists three elements which influence ease of use: the modelling system, the designer, and the modelling technique. Only the nature of the technique itself is considered since the other two fall outside the scope of this thesis.

Representational fidelity is the degree to which the description for a modelling technique resembles the resulting model. If there is a close correlation, then the designer does relatively little interpreting between the two. If the description does not closely resemble the model produced, then the designer must learn to interpret between them. This puts an extra cognitive demand on the designer which distracts him from the task of designing his model [Hill 84].

As a simple example, presume that a designer wishes to model an ellipse. The analytic description of an ellipse has the form $Ax^2 + Bx + Cxy + Dy^2 + Ey = F$. The parametric description of an ellipse has the parameters: centre, major and minor axis lengths, angle of rotation. The analytic description has a lower degree of representational fidelity than the parametric description, since the six analytic coefficients have little intuitive relationship to the position, shape, and orientation of the ellipse.

A modelling system can compensate for the low representational fidelity of a modelling technique by presenting an interface to the user which hides the discrepancy between the description and the model. However, it is still important to recognise

that this discrepancy exists, even if the designer is unaware of it.

2.2.4 Geometric Consistency

A model is *geometrically consistent* if it defines a physically possible object. In the case of solids, a geometrically consistent model must meet the definition of validity given for solid models in Section 2.1. A geometrically consistent surface model represents an object which does not intersect itself. It may also be required that a surface model have no breaks.

2.2.5 Convertibility

Models are converted to different representations for several reasons:

- Some modelling systems support more than one modelling technique and need to combine components created with each (eg. [Prime 86]).
- Designers may wish to share models among different modelling systems. This is commonly done in the CAD industry using standards such as the Initial Graphics Exchange Specification [Liewald/Kennicott 82].
- A modelling representation may be inappropriate for certain applications such as rendering.

Accuracy and reversability are important when converting models from one representation to another.

2.2.6 Space Efficiency

There are two types of space requirements for modelling: the computer memory required by a given technique embedded in a modelling system, and the secondary storage space required to store the description of the model.

It is common for models in computer graphics to comprise many thousands of components. It is essential that such models be stored as efficiently as possible. The usual approach to reducing storage space requirements is to store the highest-level abstraction possible. This has the disadvantage that the data must be processed each time it is read to reproduce the detailed description. There is a trade-off between storage space and processing time as is generally the case with data processing.

The second space consideration is the amount of computer memory required to run a modelling system which implements a given technique. Here again, the trade-off between storage space and processing time is important since even more information will likely be embedded in the data structure in memory than is required to store it on disc. This extra information might be a simple set of cross-references which speed up data traversal. Some modelling systems maintain a second complete geometric description of the boundary of the model to accelerate display [Pratt 88].

2.2.7 Computational Efficiency

Computational efficiency is a measure of how much computational effort is required to derive geometric properties which are not explicitly specified in the model. Some properties required for animation and rendering are:

- extents (bounding boxes),
- intersections with a 3D line,

- surface normal at a point on the surface,
- point membership (whether a point is inside, outside, or on the boundary)[1],
- surface area.

Some properties can be approximated for some techniques in a reasonable amount of time, but take much longer to compute exactly. Therefore, accuracy will be taken into consideration when evaluating the computational efficiency of a modelling technique.

2.2.8 Manipulation Speed

Manipulation speed is a measure of the amount of delay between the designer's action and the system's reaction. Although this is largely dependent on the modelling system, it is primarily dependent on how efficiently the underlying model can be modified, transformed, and displayed.

Modifications can generally be made to the description of a model very quickly. However, different techniques require different amounts of computational effort to convert the description into a form which can be transformed and displayed.

Transformation of 3D models includes translation, rotation, scaling, and projection. The use of four element homogeneous coordinates and 4×4 transformation matrices is standard in computer graphics [Newman/Sproull 79]. Points can be transformed conveniently and efficiently through concatenation of transformation matrices. Special purpose graphics hardware has been developed to accelerate the processing of homogeneous algebra [Clark 82]. Certain modelling techniques are better suited to homogeneous transformation than others.

[1]Point membership is only relevant for solids and bounding surfaces

Generally, the fastest *display* method for 3D objects is as a *wire frame*. A model is drawn as a set of line segments representing its edges or a grid of lines approximating its surface. Some modelling techniques require conversion to another representation since this edge or grid information is not stored explicitly.

2.2.9 Applicability to Animation

Two applications will be considered for computer graphics modelling techniques: animation and rendering. The requirements which these have of models are outlined in this section and the next.

Most modelling techniques produce models which can be easily translated, rotated, and scaled over time. However, they do not necessarily allow the models to change shape in more complex ways over time in a controlled manner. Generally, modelling techniques suffer from two problems in this regard: either their parameters do not correspond to the changes they produce in an intuitive way (low control) or they tend to lose geometric consistency when modified over time (low flexibility).

2.2.10 Applicability to Rendering

Rendering encompasses all that is necessary to create a desired image of a model. Rendering algorithms require the types of geometric information listed under Computational Efficiency, as well as surface attributes like colour and transparency. Only geometric considerations (as opposed to surface attributes) are considered for surface rendering.

Extents are used to reduce the size of the problem and to establish occlusion order [Sutherland *et al* 74]. Intersections with 3D lines are used for ray-tracing

[Whitted 80]. Surface normals are used for ray-tracing and for shading calculations. Point membership is used to determine the visible surface for certain modelling techniques. Surface area is used for radiocity calculations [Greenberg *et al* 86].

Surface rendering is a computationally intensive process for which many optimisation techniques have been developed, including pre-sorting and scan-line coherence [Sutherland *et al* 74], converting solid models to surface models [Wyvill *et al* 86b], and building space sub-division representations such as octrees [Yamaguchi *et al* 84].

2.2.11 Omissions from the Solid Model Properties

Two properties for solid model representations, unambiguity and uniqueness, have been omitted from the criteria for CG modelling techniques. Amibiguity does not arise with any of the techniques surveyed. Uniqueness is not generally important to computer graphics since models are not generally compared automatically by CG modelling systems. Animation and rendering techniques may be developed in the future which require these properties.

2.3 Summary of Criteria for CG Modelling Techniques

Table 2.2 summarises the criteria for modelling techniques and gives the terminology to be used throughout the thesis to evaluate modelling techniques.

CRITERION	EVALUATION TERMINOLOGY
Expressive Power	Solid or Surface Surfaces – open or closed Holes and breaks – possible, avoidable Continuity – degree allowed or enforced
Precision and Resolution	Precision – high, moderate, low Resolution – variable, fixed, infinite
Representational Fidelity	Degree – high, moderate, low
Geometric Consistency	Surfaces – self intersecting Solids – validity enforced
Convertability	To/from which representations Accuracy – high, moderate, low
Space Efficiency	Memory, Disc – high, moderate, low
Computational Efficiency	Speed, Accuracy – high, moderate, low
Manipulation Speed	Modification, Transformation, Display – high, moderate, low
Applicability to Animation	Flexibility, Control – high, moderate, low
Applicability to Rendering	Availability of required information – high, moderate, low

Table 2.2: Terminology for Evaluation of Modelling Techniques

Chapter 3

SURVEY of MODELLING TECHNIQUES for COMPUTER GRAPHICS

This chapter first classifies the commonly used CG modelling techniques according to a taxonomical scheme, then presents the most useful of them in detail. The techniques are analysed according to the criteria (Section 2.2) of CG modelling techniques. The analyses are summarised in Chapter 6.

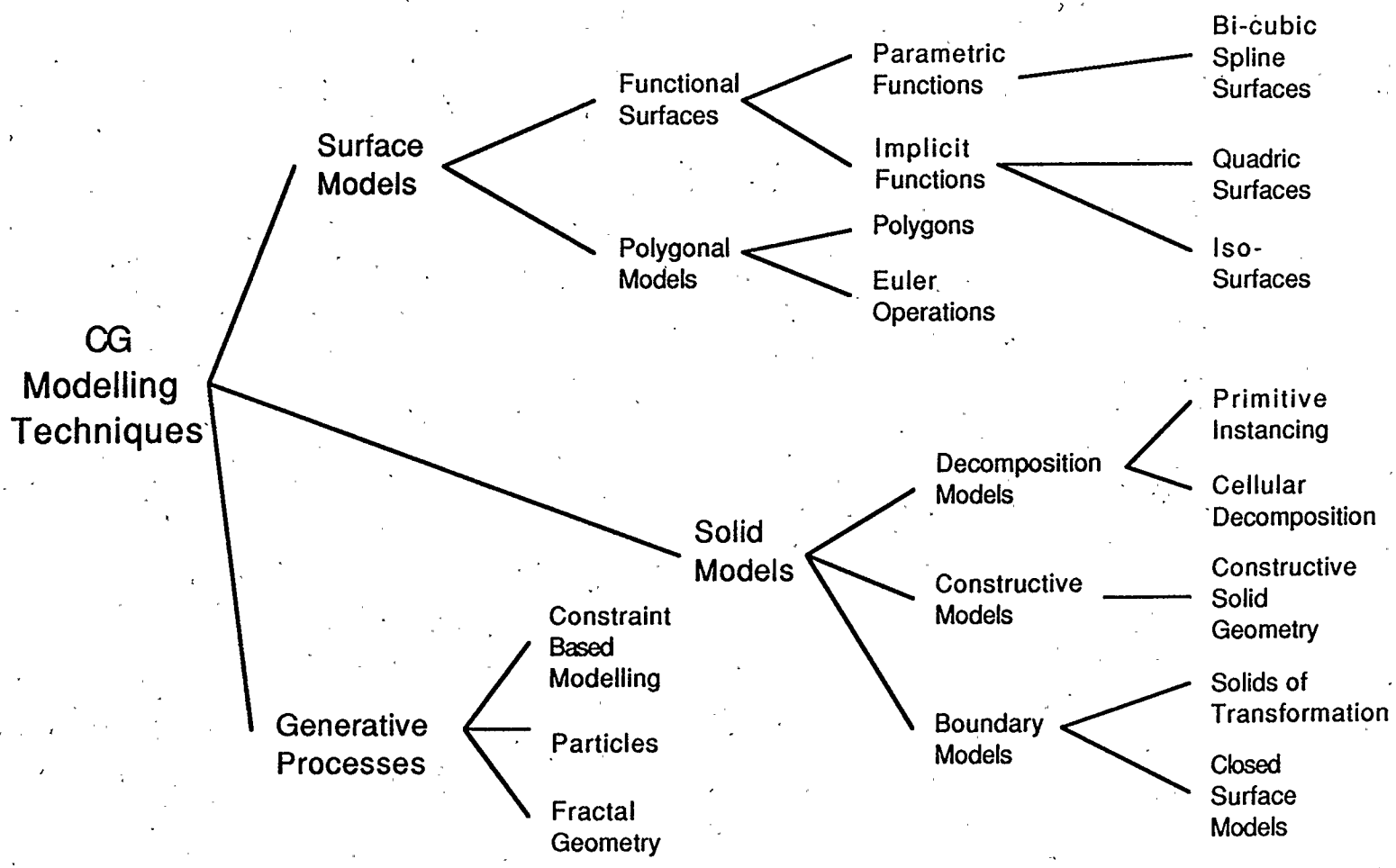
3.1 Taxonomy of CG Modelling Techniques

A taxonomy of CG modelling techniques is illustrated in Figure 3.1. At the top level they are classified by whether they represent objects as surfaces, as solids, or with generative processes. Surface techniques describe infinitely thin 3D shells. Solid techniques define contiguous regions of space. Generative processes can describe surfaces, solids, and other model types, such as fire and cloud.

Surface Models Surface modelling techniques are divided into polygon techniques and functional techniques. Polygon techniques represent surfaces as collections of discrete, planar polygons. Individual polygons can be grouped together [Wyvill *et al* 84] or collections of polygons can be represented by a polygon mesh [Chiyokura/Kimura 83].

Functional techniques describe continuous surfaces with mathematical functions.

Figure 3.1: A TAXONOMY OF CG MODELLING TECHNIQUES



Some functional techniques use implicit functions of the form $f(x,y,z) = 0$ (quadric surfaces and iso-surfaces), others use parametric functions of the form $f(t) = (x(t), y(t), z(t))$ (bi-cubic spline surfaces).

Solid Models Mantyla classifies solid models as: decomposition models, constructive models, and boundary models [Mantyla 88]. *Decomposition models* are made by juxtaposing primitive solids. A simple example is a three dimensional array of cubic cells. More sophisticated decomposition models are binary space partition trees and octrees. *Constructive models* combine primitive solids using set-theoretic binary operators. *Boundary models* represent solids with closed surfaces. Any bounding surface model can represent a solid (eg. volumes of rotation/translation [Prime 86]).

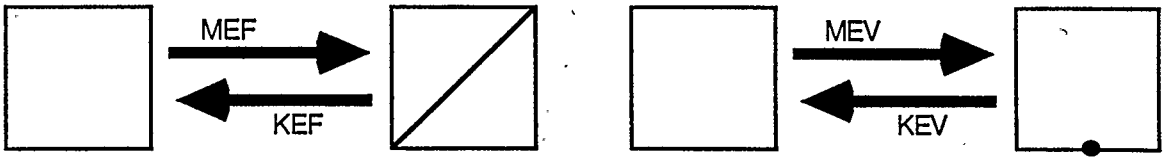
Generative Processes Generative processes produce models from very high level descriptions using generally more complex algorithms than either surface or solid models. Common generative modelling techniques include fractal geometry [Fournier *et al* 82], particles [Reeves 83], and constraint based modelling [Prusinkiewicz/Streibel 86].

The remainder of this chapter describes five common CG modelling techniques: Euler operations on polyhedra, bi-cubic spline surfaces, quadric surfaces, iso-surfaces, and constructive solid geometry.

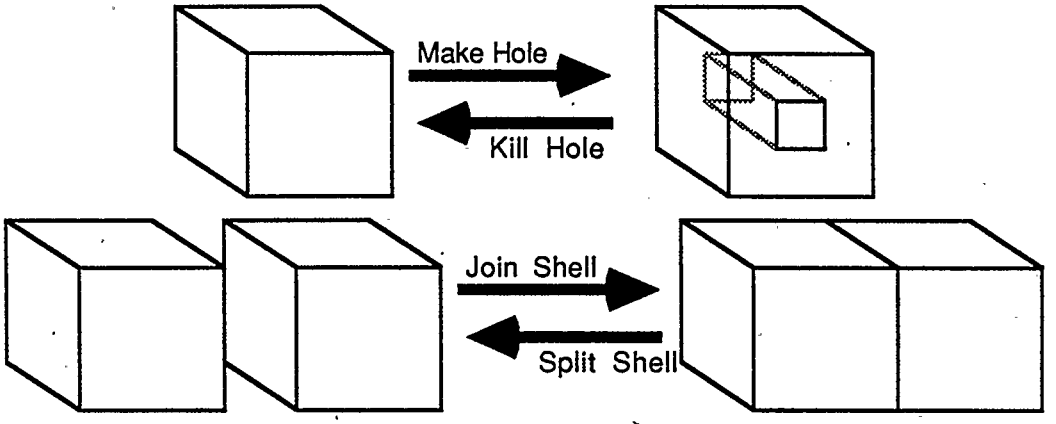
3.2 Euler Operations on Polyhedra

A polyhedron is a collection of planar polygons which form a bounding surface. An Euler operation is any modification to the topology of a polyhedral model which pre-

serves topological consistency [Mantyla 88]. There are two types of Euler operation: local and global (Figure 3.1).



a) Local Euler Operations (preserve V - E + F)



b) Global Euler Operations (preserve V - E + F - 2(S - H))

Figure 3.1: Euler Operations on Polyhedra

Local Euler operations preserve the Euler characteristic ($\chi = V - E + F$) of the polyhedron. For example, splitting a square into two triangles by connecting two diagonally opposite vertices increases both E and F by one leaving χ unchanged. This operation is called MEF (make edge, face). The MEV operation (make edge, vertex) splits an edge into two by introducing a new vertex on it. The KEF (kill edge, face) and KEV (kill edge, vertex) operations perform the inverse of MEF and MEV.

Global Euler operations are used to make and kill holes and to join and split

polyhedra. They preserve the more general Euler-Poincare formula:

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

where S is the number of separate polyhedral shells and H is the total number of holes.

To use Euler operations as a CG modelling technique it is necessary to be able to modify the geometry of a polyhedron as well as its topology. MODIF [Chiyokura/Kimura 83] is a modelling system which uses the local Euler operations Make and Kill together with two geometric operations, Move and Lift, to modify polyhedra (Figure 3.2). Each of these operations can be applied to vertices, edges, or faces. Move extends the edges connected to the element being moved. Lift adds new faces and edges between the old and new positions of the element moved.

3.2.1 Analysis of Euler Operations

Expressive Power Clearly, polyhedra are surface models which admit holes but do not admit breaks. They have only continuity of position (C^0) since slopes change abruptly between polygons.

It can be shown that every bounding surface is topologically equivalent either to the sphere, or the connected sum of n tori [Mantyla 88, Theorem 9.4, p.143]. Therefore, using local and global Euler operations together with some geometric operations like Move and Lift it is possible to create bounding surfaces with any topology.

Precision and Resolution All geometric properties of polyhedra can be specified exactly. The resolution of the mesh is altered with local operations as needed.

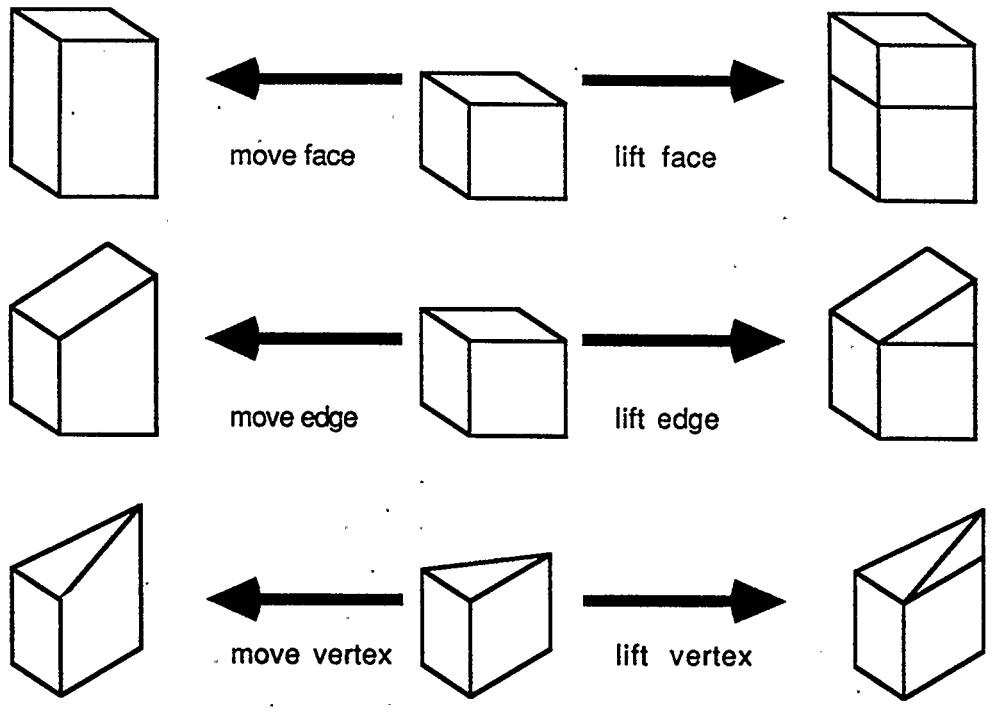


Figure 3.2: Move and Lift Operations

Representational Fidelity Polyhedra are exact representations in that their description is identical to the model they represent.

Geometric Consistency Self-intersecting surfaces and non-planar polygons are easily created, therefore inconsistent objects may be described.

Convertability Most other modelling representations can be converted into polygonal approximations which can be stored as polygon meshes. Polyhedral properties, such as topological consistency, are not guaranteed.

Space Efficiency Data structures for polyhedra tend to be large since all topological and geometric information must be stored explicitly, both in memory and on disc. Polygon mesh data structures are used to represent polyhedra (Section 4.1).

Computational Efficiency Extents, intersections, surface normals, and surface area are very fast to compute. Point membership is more complex since a point must be compared with every face to determine membership [Preparata/Shamos 85].

Manipulation Speed Modification of the data structure is moderately difficult because of its complexity. Transformation is slow for large polyhedra because each vertex must be transformed separately. However, wire frame display is very fast since all edge information is stored explicitly.

Applicability to Animation Polyhedra offer limited flexibility without compromising geometric consistency. Animating them by moving their components (vertices, edges, faces) offers a high degree of low level control, but inconsistencies can

easily be created, such as non-planar polygons which are unacceptable to most rendering algorithms.

Applicability to Rendering Geometric information is readily available from polygonal models. Scan line coherence is easily utilised due to the planar property of polygons. Sorting is useful because of the discrete nature of polygons.

3.3 Quadric Surfaces

Quadric surfaces are represented by second order polynomials in four variables of the form:

$$Ax^2+2Bxy+2Cxz+2Dxw+Ey^2+2Fyz+2Gyw+Hz^2+2Jzw+Kw^2=0 \quad (3.1)$$

where 3D points are represented by four element homogeneous coordinate vectors $[x\ y\ z\ w]$. Regular 3D points $[X\ Y\ Z]$ are obtained by dividing each of x, y, and z by w.

The locus of points which satisfy (3.1) is called a quadric surface. Quadric surfaces include cylinders, cones, ellipsoids, paraboloids, and hyperboloids of one and two sheets (Figure 3.3). Degenerate forms, such as points and lines, are also possible. An implementation of quadric surfaces is presented in [Goldstein/Nagel 71].

Equation (3.1) can also be expressed in matrix form:

$$[x\ y\ z\ w]\begin{bmatrix} A & B & C & D \\ B & E & F & G \\ C & F & H & I \\ D & G & I & J \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = 0 \quad (3.2)$$

This form is particularly useful for performing the following calculations. Most of the following is presented in more detail in [Blinn 86].

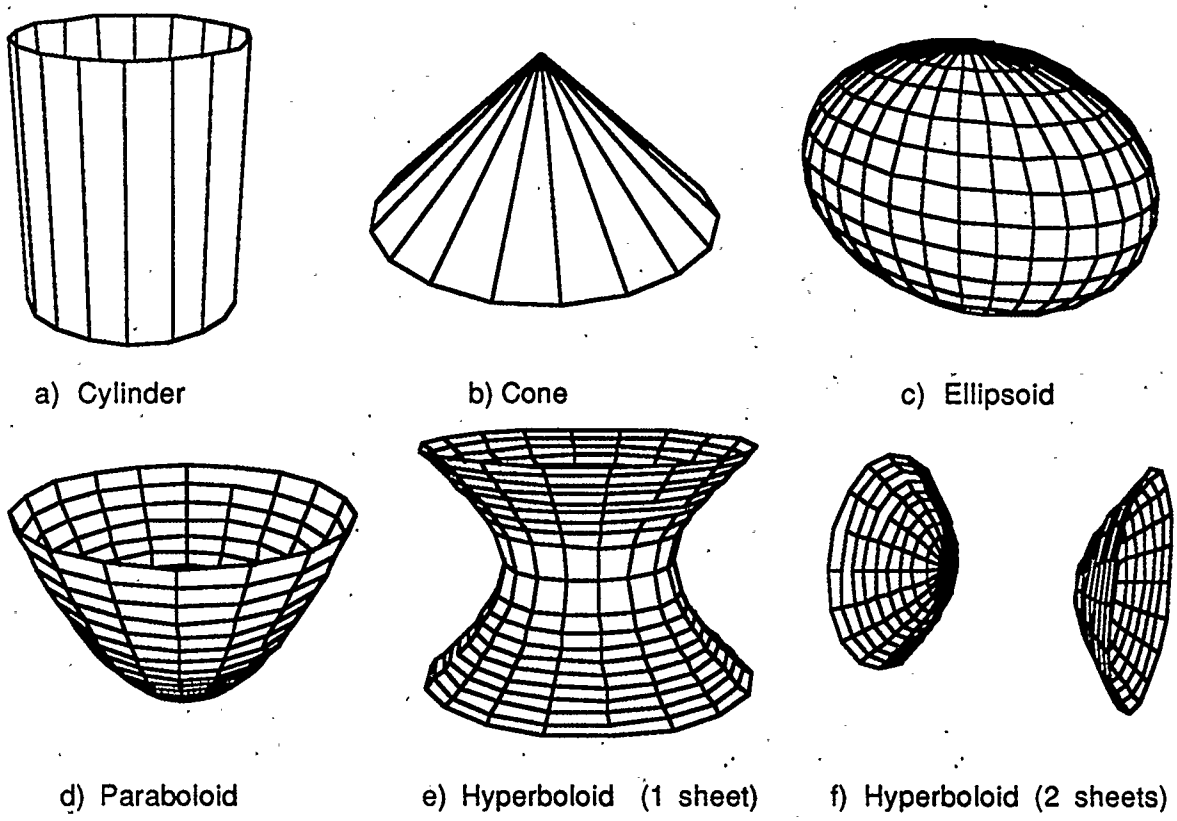


Figure 3.3: Quadric Surfaces — note that these are polygonal approximations, most of which have been truncated

Transformations A quadric surface can be transformed by manipulating the 4×4 matrix in (3.2), called Q. The matrix Q is transformed by any 4×4 homogeneous transformation matrix T, to form Q' as follows:

$$Q' = T^* Q T^{*t} \tag{3.3}$$

where T^* is the adjoint of T (transpose of the cofactor matrix), and T^{*t} is the transpose of T^*.

Intersections To intersect a quadric surface represented by matrix Q with a plane represented by the coefficient vector $P = [a\ b\ c\ d]$, solve PQ^*P^t, where Q^* is the adjoint of Q. If the result is zero, the plane is tangent to the quadric surface; if greater than zero, they intersect in a conic section; if less than zero, they are disjoint.

Similarly, the intersections of a line represented by the coefficient vector $L = [a\ b\ c\ 1]$, are found by solving $LQ^*L^t = 0$.

Extents The x extents of a quadric surface are found by evaluating $PQ^*P^t = 0$ for the plane $P = [-1\ 0\ 0\ x_0]$ which reduces to a quadratic in x_0. The solutions to the quadratic are the x extents of the surface. The analogous calculations are used for y and z extents.

Surface Normals The normal vector N for the surface point $[x\ y\ z\ w]$, is:

$$\mathrm{N} = 2(\mathrm{x\ y\ z\ w})\mathrm{Q}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Rendering A quadric surface can be rendered by transforming it into screen space and intersecting it with the line $[x_s\ y_s\ z\ 1]$ for each screen pixel (x_s, y_s). The surface

normal calculation is then applied to each intersection point to determine its colour intensity.

This is extremely inefficient and can be improved as follows. The screen space limits of a quadric can be found by projecting its silhouette into a 2D quadric curve on the screen. From the 2D quadric, the Y extents can be found, and for each scan line in this range the X extents can be found. Forward differences [Foley/van Dam 83, p.533] can then be used to evaluate the equations for z and surface normal efficiently across each scan line.

3.3.1 Analysis of Quadric Surfaces

Expressive Power The set of non-degenerate quadric surfaces is limited. They do not admit holes. Continuity of curvature is guaranteed.

The expressive power of quadrics has been extended by [Barr 81] with *super-quadrics* (formed by allowing the exponents in (3.1) to be real valued) and *angle preserving transformations* (weighted sums of quadrics).

Precision and Resolution Quadric surfaces provide a low degree of modelling precision since dimensions and angles cannot be specified directly. Being continuous mathematical surfaces, their resolution is infinite.

Representational Fidelity There is a very low intuitive correspondence between the ten coefficients in (3.1) and the resulting models.

Geometric Consistency Non-degenerate quadrics form valid solids in every respect except that most are unbounded (all except ellipsoids).

Convertability Quadric surfaces can be approximated with polygon meshes. Other representations cannot generally be converted to quadric surfaces because of their limited expressive power.

Space Efficiency Space requirements for quadrics are extremely low, both in memory and on disc, making them very space efficient.

Computational Efficiency The algorithms for all of the properties listed in Section 2.2.7 except surface area are outlined above. They are analytical solutions, but each requires at least the multiplication of a vector by a 4×4 matrix. Surface area can theoretically be found by integrating (3.1). Over all, quadrics have only moderate computational efficiency.

Manipulation Speed Modification and transformation are very fast. Wire frame display requires the creation of grid information which approximates the surface.

Applicability to Animation Quadric surfaces offer low animation control due to their low representational fidelity. They also have low flexibility since degenerate forms are easily created which violate geometric consistency.

Applicability to Rendering All geometric information required to render quadrics can be derived directly from the matrix Q with a moderate amount of effort. Scan line coherence can be utilised.

3.4 Iso-surfaces

An implicit surface is defined by evaluating a 3D implicit function for a constant value $v = f(x, y, z)$ (eg. quadrics). An iso-surface is an implicit surface defined by the weighted sum of several field functions, each based on the distance from a geometric element:

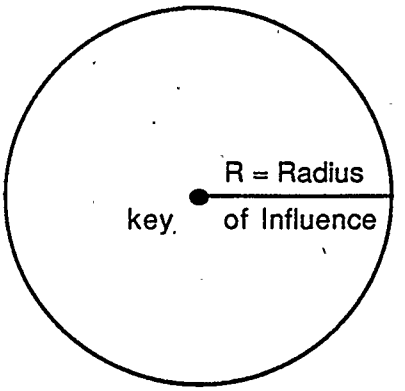
$$v = \sum_{k=1}^{n} f_k(x, y, z) \tag{3.4}$$

where f_k is the field function for the k^{th} geometric element.

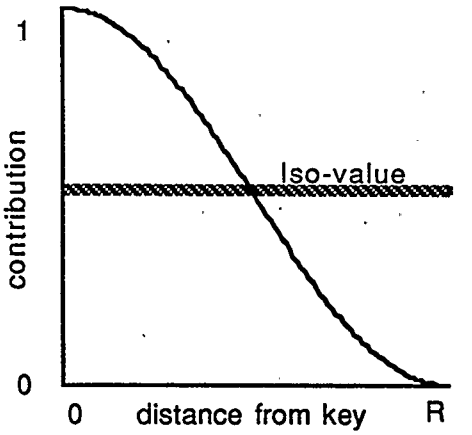
Several variations of this technique have been developed independently: *blobby molecules* [Blinn 82], *soft objects* [Wyvill *et al* 86b], *meta-balls* [Nishimura *et al* 85], and *implicit surfaces* [Bloomenthal 87]. In each of these variations, the function defines a continuously varying scalar field based on distances from geometric elements. [Blinn 82] and [Wyvill *et al* 86b] use points as the defining elements. [Bloomenthal 87] also experiments with 3D parametric curves and planar polygons. Soft objects are outlined below.

Soft Objects Soft objects consist of a set of control points (called *keys*) each of which contributes to a 3D scalar field according to some *field function.* Each key contributes to every point within a *radius of influence R* (Figure 3.4.a) according to a monotonically decreasing cubic function (Figure 3.4.b). Field functions which influence elliptical and super-elliptical regions of space have also been developed [Wyvill 88].

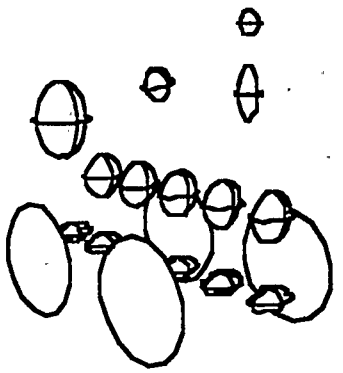
When two or more keys have overlapping regions of influence, the overall field value of a point in the overlapping region is the sum of the contributions of each key. The field value chosen to represent an iso-surface is called an *iso-value.* The set of



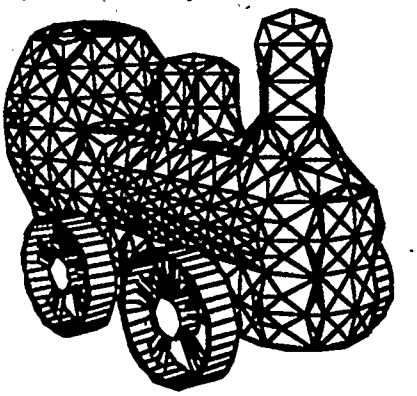
a) Key and Radius of Influence



b) Cubic decay function



c) Skeleton for Chubby -
Each ellipsoid represents a key
(except for the wheels)



d) Chubby - an iso-surface train

Figure 3.4: Iso-surfaces

all points with this field value is guaranteed to be one or more bounding surfaces.

To use this technique, a designer positions the keys, assigns field functions to each, and chooses an iso-value. A collection of keys with ellipsoidal field functions is shown in Figure 3.4.c and the corresponding iso-surface in Figure 3.4.d. A figure like Figure 3.4.c is called a *skeleton* and is a useful approximation of a soft object for quick calculations.

3.4.1 Analysis of Iso-surfaces

Expressive Power Iso-surfaces always form closed surfaces. Holes are possible either by placing keys so that their fields do not "fill in" or by assigning negatively valued field functions to keys. Iso-surfaces exhibit the same degree of continuity as the field functions used, since continuity is preserved when functions are added. Specifically, soft-objects have C^2 continuity since they are based on cubic field functions.

Precision and Resolution Precision is low since the surface is defined implicitly. Techniques have not yet been developed for specifying iso-surfaces explicitly and having the implicit form generated automatically. The resolution is infinite since the field functions are continuous.

Representational Fidelity Iso-surfaces have a moderate degree of representational fidelity. Although the positions of the keys and their field functions are specified explicitly, the shape of the surface can be difficult to predict where regions of influence overlap.

Geometric Consistency Iso-surfaces are always geometrically consistent bounding surfaces since they are closed and do not intersect themselves.

Convertability Iso-surfaces can be efficiently converted to polygon meshes [Wyvill *et al* 86b] or octrees [Bloomenthal 87] using a 3D digital differential analyser. Conversion to iso-surface representation has not been done.

Space Efficiency Iso-surface descriptions are very compact to store on disc. They are also compact in memory but are often accompanied by an evaluated boundary representation or space sub-division, the size of which depends on the resolution used.

Computational Efficiency Due to their non-analytical form, numerical methods are required to compute most geometric properties of iso-surfaces. For example, the intersection with a line is found by starting with two points on the line, one "in" and the other "out", and converging on the surface. Finding the initial points can be optimised using space sub-division [Wyvill *et al* 86b].

Extents can be approximated quickly from the skeleton of an iso-surface. Exact extents require convergence as do intersections with lines. Point membership can be determined quickly by solving (3.4) and comparing with the iso-value. Surface normals are found by summing the partial derivative of field functions. Surface area can be approximated with numerical methods.

Manipulation Speed The parameters describing an iso-surface can be modified and transformed very quickly. Wire frame display requires the creation of grid information which approximates the surface.

Applicability to Animation Iso-surfaces are particularly well suited to animation due to their geometric consistency and continuity [Wyvill *et al* 86a]. Varying any of the parameters smoothly over time will produce smoothly changing closed surfaces. The number of surfaces and the number of holes change in an expected manner.

Applicability to Rendering Geometric information is moderately difficult to compute even with a space sub-division. Scan line coherence and sorting are not applicable since every key is potentially involved in each calculation.

3.5 Bi-cubic Spline Surfaces

Smooth curved surfaces cannot be modelled exactly with polygon models. Implicit functions such as quadrics do not allow the designer to directly manipulate the surface of the model. Explicit functions of the form $x = f(y, z)$ have orientation dependent numerical instabilities.

Parametric functions of the form $f(t) = (x(t), y(t), z(t))$ avoid numerical instabilities while allowing smooth surfaces to be manipulated more or less directly through *control points.* Complex parametric curves are represented piecewise as series of curve segments. For each segment, four constraints must be enforced, namely, the initial and final positions and slopes. Cubics are the lowest order functions which can satisfy these constraints.

Cubic spline curves are designed by specifying these four constraints for each curve segment using control points (see Figure 3.5). A Bezier curve segment [Bezier 74] is defined by its end points, p_1 and p_4, and two other points, p_2 and

p_3, such that $p_2 - p_1$ and $p_3 - p_4$ define the initial and final slopes (Figure 3.5.a). A B-spline curve segment [Gordon/Riesenfeld 74] does not necessarily pass through any of its four control points but is influenced by each (Figure 3.5.b).

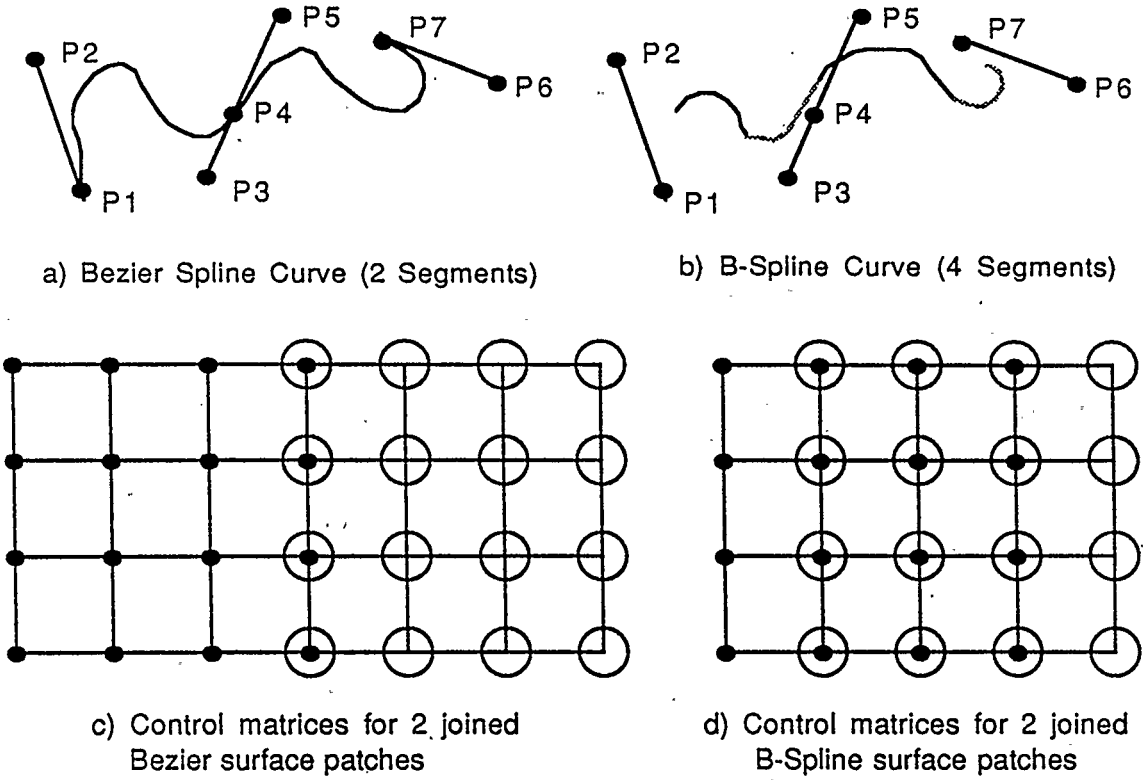


Figure 3.5: Cubic Spline Curves and Surfaces

Curve segments are joined into complex curves by sharing control points. Adjacent Bezier segments share a common end point, p_4. The resulting curve will have continuity of slope (C^1) if p_3, p_4, and p_5 (ie. p_2 of the next segment) are colinear (see Figure 3.5.a). Adjacent B-spline segments share three common control vertices. Continuity of slope and curvature (C^2) are guaranteed across B-spline segment boundaries.

Just as spline curves are made by joining cubic segments, spline surfaces are made by joining bi-cubic patches, each defined by 16 control points. As in the case of curves, patches are joined together by sharing control vertices with adjacent patches to form surfaces. Bezier patches share four control points along each edge (Figure 3.5.c), B-spline patches share 12 (Figure 3.5.d). Surfaces have the same continuity characteristics as the curves from which they are derived.

Bezier splines are an example of *interpolating splines* which pass through some of the control points. B-splines are an example of *approximating splines* which do not necessarily pass through any control points. For a comprehensive coverage of splines in computer graphics, see [Bartels *et al* 87].

Spline formulations with more control mechanisms have been developed. Beta-splines [Barsky/Beatty 83] are a generalisation of B-splines, with two added parameters, "bias" and "tension", which can be specified locally at each control point or continuously between them. [Kochanek/Bartels 84] present an interpolating spline formulation with three parameters, "bias", "tension", and "continuity", at each control point.

3.5.1 Analysis of Bi-cubic Spline Surfaces

Expressive Power Spline surfaces are clearly surface models. The topology of the surface depends on that of the lattice of control points (the *control matrix*). Closed surfaces with holes are possible, but easily avoidable, as are open surfaces with breaks. Interpolating splines only guarantee C^1 continuity between patches whereas approximating splines guarantee C^2.

Precision and Resolution . Interpolating splines have greater precision since their end points are defined. Both types of splines have the property that the surfaces lie inside the convex hull of their control points. The resolution of the surface itself is infinite, however the "control resolution" depends on the resolution of the control matrix.

Representational Fidelity Splines have a moderately high degree of representational fidelity since the coefficients of the parametric functions are derived from geometric control points. Interpolating splines are somewhat higher than approximating splines.

Geometric Consistency Spline surfaces can easily intersect themselves, thereby creating inconsistent objects.

Convertability Spline surfaces convert quickly to polygon mesh representations by parametric interpolation. Spline patches can be used to approximate other smooth surfaces such as quadrics.

Space Efficiency Spline surfaces have moderately high space requirements. A simple vase or sphere may require on the order of tens of patches.

Computational Efficiency Extents can be quickly approximated by the extents of the control matrix or their convex hull. Intersections with lines and planes require numerical methods due to the parametric form of spline surfaces. Surface normals can be found analytically from the parametric position of a point. Surface area can be found by integrating over parameter space.

Manipulation Speed Modifying and transforming the control matrix is fast. Wire frame display is also fast as follows. For a given parametric resolution ϵ in the range $[0 \rightarrow 1]$ a wire frame grid is found by ϵ^2 evaluations of the parametric function. Forward differences can be used, reducing the incremental computations to a few additions.

Applicability to Animation Spline surfaces can be animated very flexibly by moving their control points. Unfortunately, interesting objects can contain an unmanageably large number of control points. The animator must be careful to maintain geometric consistency.

Applicability to Rendering Exact surface rendering requires numerical methods [Lane *et al* 80]. Spline surfaces are commonly converted to polygonal models before being rendered.

3.6 Constructive Solid Geometry

Constructive Solid Geometry (CSG) is based on a set of primitive solids which are combined using binary operators [Requicha/Voelcker 83]. Typically the primitives are simple half space functions which divide 3D space into two regions, "in" and "out" [Kunii/Wyvill 85]. Planar, spherical, conical, and cylindrical half spaces are popular.

Primitives are regarded as being point sets, and are combined using set-theoretic binary operators, typically union ($\cup$), intersection ($\cap$), negation ($\neg$), and difference ($-$). Figure 3.6 illustrates an example CSG model and the binary tree, called a CSG

tree, used to represent it. The left branch of the CSG tree represents a hemisphere created by subtracting a planar half space from a sphere. The right branch represents the union of two cylinders. The right branch is subtracted from the left branch to form the bottom object in Figure 3.6.a.

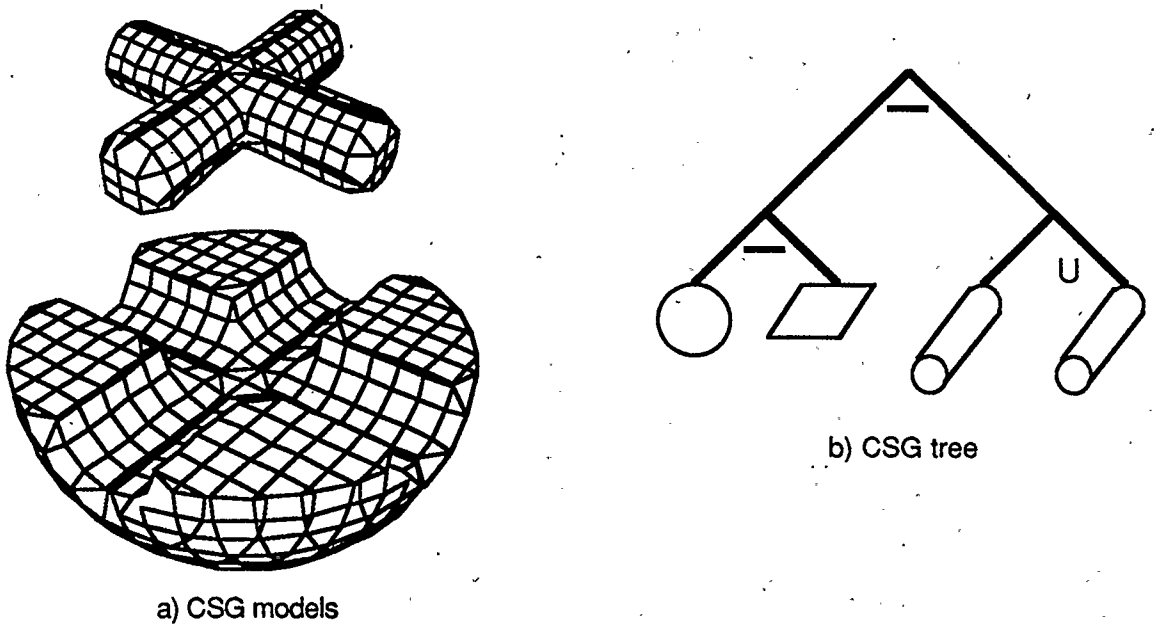


Figure 3.6: Constructive Solid Geometry

CSG Trees Each non-leaf node of a CSG tree is labeled with a binary operator which is applied to its child sub-trees. Each arc of the CSG tree contains spatial transformation information relating its distal child to its proximal parent.

Most calculations for CSG models require a traversal of the CSG tree, visiting each internal node (set operation), arc (transformation), and leaf node (primitive). For example, to intersect a line with a CSG model, the line is transformed into the local coordinate space of each primitive and intersected with it. The intersection

points are sorted along the line and tested for membership to determine which one is correct.

CSG Primitives Most of the half spaces mentioned above define unbounded "in" regions. Bounded primitives like cubes and truncated cones and cylinders can easily be built from them using the union operator.

Any function which divides space into two regions for which the required properties, (point membership, surface normal, and line intersections), can be found can be used as a CSG primitive. Volumes of transformation, iso-surfaces, and quadric surfaces are possible candidates.

3.6.1 Analysis of Constructive Solid Geometry

The analysis for many criteria depends on the solid primitives used. A typical set of simple half spaces (plane, cylinder, cone, sphere) is assumed.

Expressive Power CSG models are solids which may have holes. The degree of continuity depends on the primitives offered. Continuity of curvature is possible (depends on the solid primitives), except at intersections between primitives, where the slope generally changes abruptly.

Precision and Resolution CSG is as precise as the primitives used. In the case of simple half spaces, exact geometric properties can be specified exactly. CSG models have infinite resolution since the primitives are continuous point sets.

Representational Fidelity CSG has a moderately high degree of representational fidelity since both CSG trees and primitives are specified explicitly.

Geometric Consistency There are two cases where invalid solids can be produced. First, unbounded primitives can produce unbounded models. This is easily avoided by taking the intersection of the model with a bounded, enclosing solid (eg. bounding cube). Second, set difference or intersection of primitives with aligned boundaries can produce invalid fragments such as lines or planes. Algorithms exist to prevent this from happening [Mantyla 88, p.83].

Convertability CSG models can be converted to polygon mesh or spline patch models the accuracy of which depends on the resolution used. Clearly, any representation which can be used as a CSG primitive can be converted to a CSG representation using set union.

Space Efficiency CSG models are entirely represented by the CSG tree since the primitives are built into the modelling system. CSG trees are very compact to store on disc. They are also compact in memory but are often accompanied by an evaluated boundary representation or space sub-division the size of which depends on the resolution used.

Computational Efficiency Most algorithms require a traversal of the CSG tree. Extents are determined by converging on the boundary numerically. Intersections require a complete traversal of the CSG tree. The normal of a surface point is found directly from the primitive to which it belongs. Point membership does not always require a complete traversal. Exact surface area can be evaluated analytically or approximated from a boundary representation. Both are computationally expensive operations. However, one of the reasons for the popularity of CSG in CAD is the

fact that volumes can be calculated very efficiently.

Manipulation Speed The CSG tree can be modified and transformed very efficiently. Displaying a CSG model as a wire frame requires conversion to a boundary representation and therefore is comparatively slow.

Applicability to Animation CSG models can be animated in two ways, either by modifying the primitives, or by modifying the CSG tree. The degree of flexibility and control of the primitives depends on which primitives are used. The spatial transformation information in the CSG tree can be animated over time in a controlled manner. The resulting solid or solids will maintain validity.

Applicability to Rendering Geometric information is moderately expensive to calculate. Some form of space subdivision is commonly used. Scan line coherence cannot generally be utilised.

3.7 Taxonomy of CG Modelling Techniques Revisited

The taxonomy presented in Section 3.1 classifies CG modelling techniques by the characteristics of the model produced (eg. surface versus solid), and by the description used to produce it (eg. implicit versus parametric functions). Clearly, this is not the only way to classify modelling techniques. They could, for example, be classified by any of the criteria (eg. manipulation speed or applicability to animation). Another scheme, presented next, decomposes modelling techniques into their software components.

Primitives, Data Structures, and Processes

Some techniques describe models as self-contained primitives, others as specialised data structures, and still others as processes.

For example, polygons are simple *primitives*, which can be organised into a directed graph *data structure*, or generated by some *process* such as fractal geometry. Table 3.1 divides the techniques in the taxonomy into these three classifications.

PRIMITIVES	DATA STRUCTURES	PROCESSES
Points	Display Lists	Euler Operations
Lines	Polygon Meshes	Polygon Mesh
Polygons	Directed Graphs	— Modelling
Solid Primitives	Surfaces/Volumes	Fractal Geometry
Quadric Surfaces	— of Transformation	Set Operators
Iso-Surface Keys	CSG Trees	Constraint Systems
	3D Arrays	Particle Systems
	BSP Trees	
	Octrees	
	Spline Surface	
	— Control Matrices	
	Iso-Surfaces	

Table 3.1: Primitives, Data Structures, and Processes

This classification could be used by the designer of a CG modelling system to decide which techniques to include. For the researcher in CG modelling, new matches between items in different lists could inspire new modelling techniques.

Chapter 4

POLYGON MESH MODELLING

Polygon meshes are usually produced by generative processes (eg. fractals), by digitising physical objects, or by converting models from other representations (such as those presented in Sections 3.3–3.6). Euler operations on polyhedra (Section 3.2) are the only common modelling technique which directly manipulates polygon meshes. Existing Euler operation systems (eg. [Chiyokura/Kimura 83]) have limited ability to manipulate the geometry of a polygon mesh, since only one vertex, edge, or face can be moved at a time.

Polygon mesh modelling is a modelling technique in which the geometry of a polygon mesh is manipulated in a general and powerful way. This chapter presents a methodology for polygon mesh modelling. The next chapter covers the implementation details of DELTA, a graphically interactive polygon mesh modelling system.

4.1 Data Structures

4.1.1 Polyhedron Model

A *polygon mesh* (or simply a *mesh*) is a collection of connected, planar polygons which share common edges and vertices. The data structures used to represent meshes are derived from the mathematical concept of a polyhedron. A polyhedron is a collection of planar polygons which form a bounding surface. A polygon mesh is more general than a polyhedron in two ways: the surface may be open (ie. there

may be breaks)[1], and there may be interpenetrating polygons.

4.1.2 Geometric versus Topological Information

Two types of information must be represented by the mesh data structure: geometric and topological. Geometric information has to do with the positions of vertices, edges, and polygons, as well as such relationships among these components as angles and distances. Topological information has to do with the connectivity among components; which components belong to which, and which are adjacent to one another.

The minimum geometric information required to represent a polygon mesh is the location of each vertex. Additional geometric information such as the normal vector to each polygon, or the mid-point of each edge may be included. Such additional information can be derived from the minimum information if it is not stored explicitly.

The minimum topological information required is the set of edges and vertices belonging to each polygon, and the adjacency relationships among polygons. Additional topological information may include the set of polygons sharing each vertex or edge, or the topological class of the object (plane, sphere, torus). Again, this additional information can be derived from the minimum information if it is not stored explicitly.

Various other types of information (neither geometric nor topological) may be included in the data structure. Modelling for computer graphics, for example, may require surface attributes (like colour and transparency) to be stored for each polygon.

[1] A polyhedron may contain holes (eg. a torus), but its surface has no breaks.

Topological consistency is an important concept when dealing with polygon meshes [Foley/van Dam 83]. A topologically consistent mesh is one with no open polygons and no unattached vertices or edges. Depending on the application, there may be maximum numbers of connect elements. For example, edges might not be allowed to have more that two incident polygons or polygons might not be allowed to have more that four edges.

There are several possible data structures for representing polygon meshes. Two common representations are presented below: doubly connected edge lists and polygon lists.

4.1.3 Doubly Connected Edge List

A *Doubly Connected Edge List* (*DCEL*) [Preparata/Shamos 85] represents a polygon mesh as a set of connected edges. Figure 4.1 illustrates a DCEL structure for a simple mesh with two polygons. The DCEL consists mainly of a list of edge records, one per edge in the mesh. Each edge record has references to two vertices, to two other edges, and possibly to two polygons. The two vertex references represent the two end points of the edge. They can be explicit vertex coordinates, but are usually references to a separate vertex list. The two other edges referenced are the next edge joining each end vertex in order clockwise about that vertex, as seen from "outside" the object. The two polygon references (if present) identify the polygons which share this edge.

A separate polygon list is used to access the edges and vertices of each polygon. It contains a pointer to one of its edges and an orientation flag indicating on which side of the edge the polygon lies.

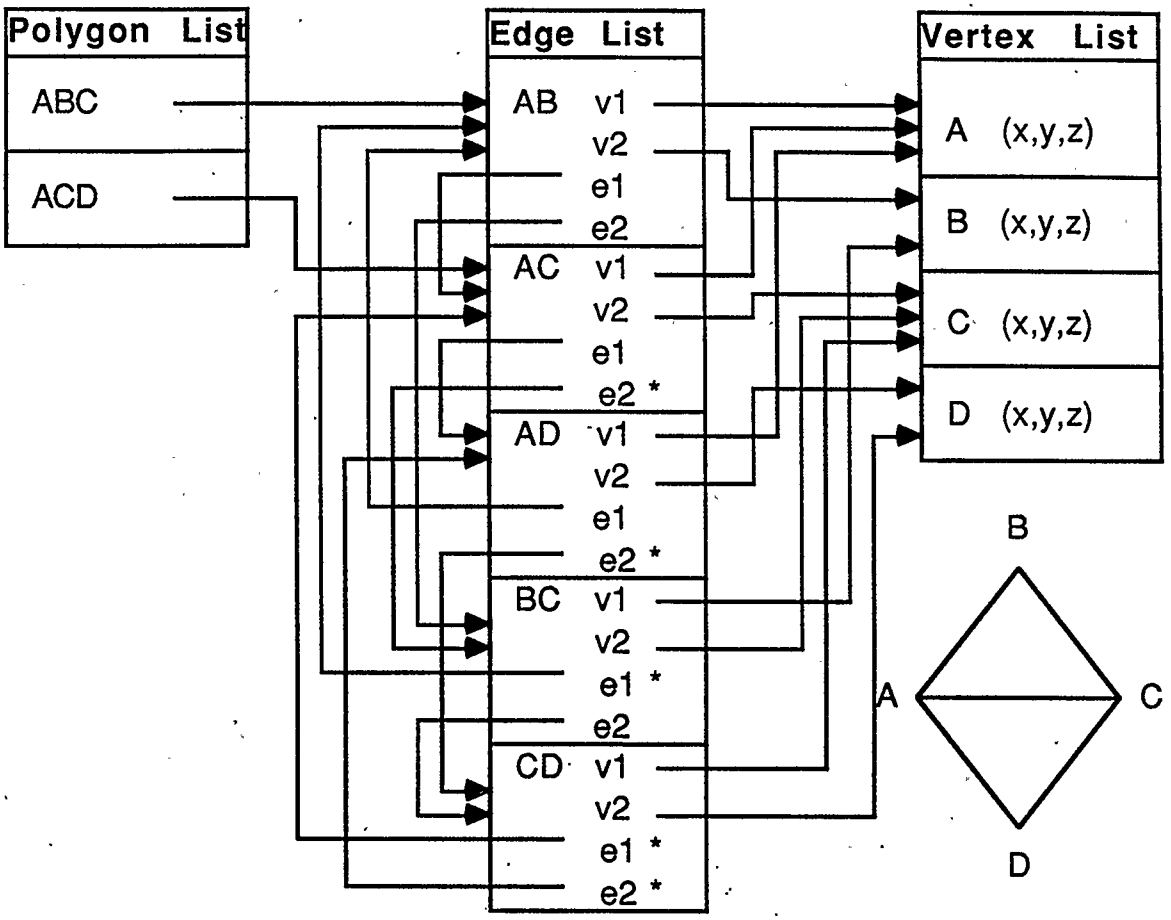


Figure 4.1: Doubly Connected Edge List Data Structure

It is possible to derive all of the topological information about the mesh from the DCEL. For example, the set of edges which meet at a given vertex can be extracted by first finding one such edge then traversing the next edge clockwise until the original edge is found again. The traversal routine must keep track of whether the given vertex is v_1 or v_2 for each edge record visited. A star (*) beside an edge pointer in Figure 4.1 denotes that the given vertex is v_2 in the structure to which it points.

The polygon references are not required in order to describe the topology of the mesh as they could be derived from the edge list algorithmically.

A similar structure is described in [Baumgart 75] called the *winged edge data structure*. The difference between it and the DCEL is that it stores the next edge in both clockwise and anti-clockwise directions for each end vertex.

4.1.4 Polygon List Structure

Another data structure for polygon meshes is the *polygon list* structure [Foley/van Dam 83]. Figure 4.2 illustrates the polygon list structure for a simple mesh with two polygons. Topological relationships are explicitly stored in three lists of records: a polygon list, an edge list, and a vertex list. Each polygon record contains references to each of its edges. Since there is a variable number of edges per polygon, the edge references are organised in a linked list or variable length array. The ordering of the edge references is important for rendering. Each edge record contains references to its two end vertices. The edge records are simpler than those in the DCEL because the extra information is distributed throughout the structure. Each vertex record contains (at least) its 3D coordinates.

A common simplification of this structure is to eliminate the edge list. In this case

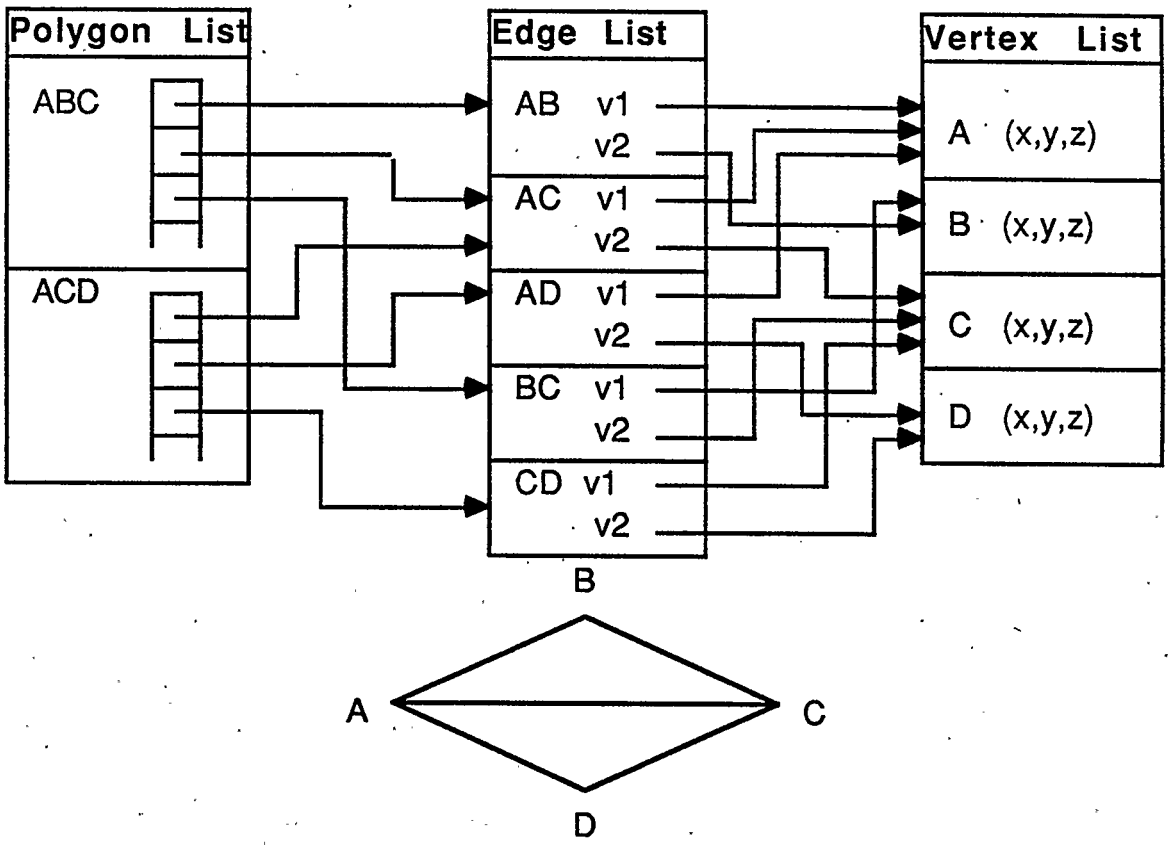


Figure 4.2: Polygon List Data Structure

the polygon list refers directly to the vertex list and the edges are implied between vertices.

4.2 File Formats

There are several formats in which a polygon mesh can be stored in a data file. This section discusses two common formats, polygon files and polygon mesh files, and compares them in terms of file storage size and file reading time.

4.2.1 Polygon Files

The simpler of the two formats is the *polygon file*. All of the information describing each polygon is stored in a contiguous block, as an ordered list of vertices. An edge is implied between each vertex and the next in the list. It is assumed that the polygon is closed, so there is an edge implied between the first and last vertices. Non-geometric information may be stored with each polygon. Alternatively, if such information is shared among several polygons, it may be specified once before the relevant set of polygons. In this approach, the data file is treated as a stream of information the order of which is important. No topological information relating one polygon to another is stored in a polygon file.

The advantage of polygon files is that each polygon can be read and processed immediately. There are two disadvantages. First, much of the geometric information is represented more than once. For example, for an object which has three polygons sharing each vertex (eg. a cube), the coordinates of each vertex will be stored three times. Second, it is not known which vertices are shared. The coordinates of a

shared vertex may differ slightly in two polygons due to numerical round-off, or two separate vertices may coincidentally have the same coordinates.

4.2.2 Polygon Mesh Files

The second file format is the *polygon mesh file*. Here a great deal of geometric information is shared among polygons. Essentially, the internal data structure, whether it is a variant of the DCEL or of the polygon list (see Section 4.1), is written directly to the file. The lists are written in the order: vertex list, edge list (if there is one), polygon list (if there is one). Each list references the lists preceding it with ordinal indices.

The advantages of the polygon mesh file are that each geometric element is represented exactly once and that some topological relationships among them are represented. The disadvantage of the polygon mesh file is that, generally, no polygon can be processed until the entire file is read.

4.3 Polygon Mesh Modelling Methodology

Editing polygon meshes directly is a difficult task because of the discrete nature of the data structure. Euler operations [Chiyokura/Kimura 83] are the only form of direct polygon mesh manipulation documented. While Euler operations are complete, in that they can be used to model any polyhedral mesh, they are awkward to use. The designer may wish to modify a sub-mesh of many polygons, or a set of disjoint vertices in one operation. He may also wish to move different vertices by different distances or in different directions in a controlled manner.

This section describes a methodology developed to provide a general, powerful, and convenient means of polygon mesh modelling. Although some concepts presented below may have been used previously for CG modelling or CAD, the literature search conducted as part of this research did not reveal a systematic collection of polygon mesh manipulation operations as described here.

4.3.1 Triangles

The planarity of polygons with more than three vertices is difficult to maintain if their edges and vertices are allowed to move independently. Therefore, the methodology requires that polygon meshes consist of triangles only. Any simple polygon[2] with N vertices can be triangulated by adding edges to it in $O(N \log N)$ time [Garey *et al* 78]. Convex polygons can be triangulated in $O(N)$ time by, for example, adding a new edge between the first vertex and each subsequent vertex.

4.3.2 Move Vertex Operation

The basic modelling operation in polygon mesh modelling is the *move vertex* operation, which specifies a new 3D position[3] for a specific vertex called the *current vertex*. All edges which terminate at this vertex are stretched or contracted as necessary in order to remain connected to the moved vertex and to remain straight. In this way, the geometry of a mesh can be arbitrarily rearranged into any form that its topology will admit. Therefore, the move vertex operation is, by itself, completely general.

Despite its generality, or perhaps because of it, the move vertex operation is not

[2] A simple polygon is one with no holes in it (simply connected) and which does not intersect itself (non re-entrant).

[3] Interaction techniques for specifying 3D positions will be discussed in Section 5.5.

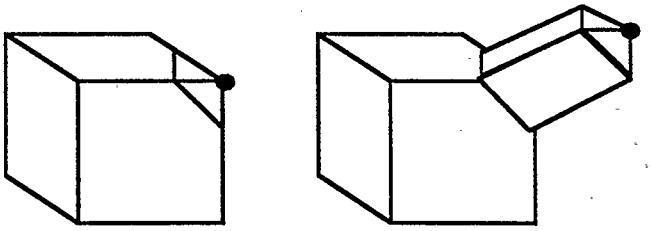
very convenient to use. Modifying a large object with thousands of vertices would be an extremely tedious and inaccurate process for the designer. Several useful extensions to the basic operation are discussed in the following sections.

4.3.3 Range of Influence

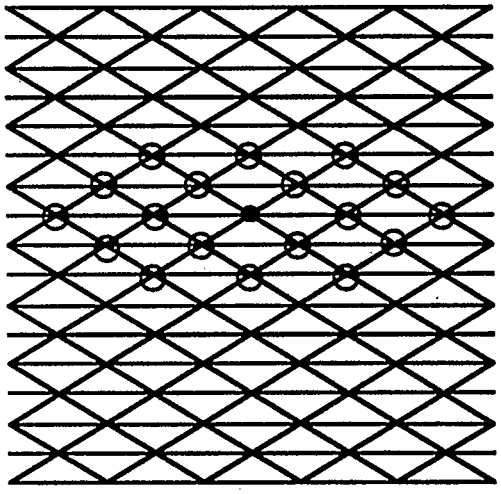
The first extension to the move vertex operation is the notion of a *range of influence* (or simply *range*) around the current vertex. The range is a connected subset of the mesh which includes the current vertex. When the current vertex is moved all vertices in the range, and correspondingly all of their connected edges, are also moved. (Assume, for the moment, that all vertices in the range are moved the same distance in the same direction.) In this way, by specifying one move vertex operation an arbitrarily large portion of the model can be modified (Figure 4.3.a).

The range can be specified by any method which defines a connected sub-mesh which includes the current vertex. Two possibilities are illustrated in Figure 4.3. The first method defines the range as all vertices which can be reached from the current vertex by traversing not more than some maximum number of edges (Figure 4.3.b). The second method defines the range as all vertices which lie within some maximum euclidean distance from the current vertex (Figure 4.3.c).

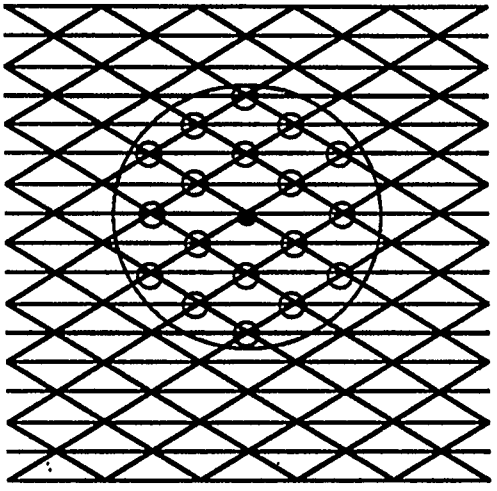
The first method has the advantage of being easy for the designer to understand and to specify. The disadvantage is that the "shape" of the range is unpredictable since it depends on the geometry of the mesh. The second method has the advantage that the "shape" of the range is independent of the geometry of the mesh.



a) Moving a range of vertices



b) Range by number of edges



c) Range by distance - circle indicates maximum distance from current vertex

Figure 4.3: Range of Influence

4.3.4 Decay Function over the Range of Influence

Using the range of influence mechanism, the entire range can be made to move by the same vector as the current vertex, the *movement vector*. This is not always the intention of the designer. For example, if a designer wishes to produce a cone shaped protrusion on a flat portion of the mesh he still must move each vertex individually. The modelling methodology requires the ability to move different vertices in the range by different amounts in a controlled manner.

A *decay function* is used to determine the amount by which each vertex is moved. It maps the position of each vertex in the range to a scale factor. The movement vector is scaled by this factor before being added to the vertex position. Figure 4.4 illustrates several decay functions applied to a flat mesh. Note that the shapes produced will be different if the mesh is not initially flat since the shape of the decay function is added to the original shape of the mesh.

In the examples illustrated in Figures 4.4.b–d, the distance from the current vertex is normalised so that the function will fit in the range. Figure 4.4.f does not normalise the distance, since it is not desirable to have the wavelength of the sine curve depend on the size of the range.

The set of decay functions shown is not exhaustive, but serves to illustrate that any function can be used to map vertex positions into scale factors.

4.3.5 Operations Applied over the Range of Influence

So far, the methodology offers three capabilities: the ability to specify a current vertex and move it in 3D space, the concept of a range of influence around the current vertex, and a way of controlling the distance by which each vertex in the range is

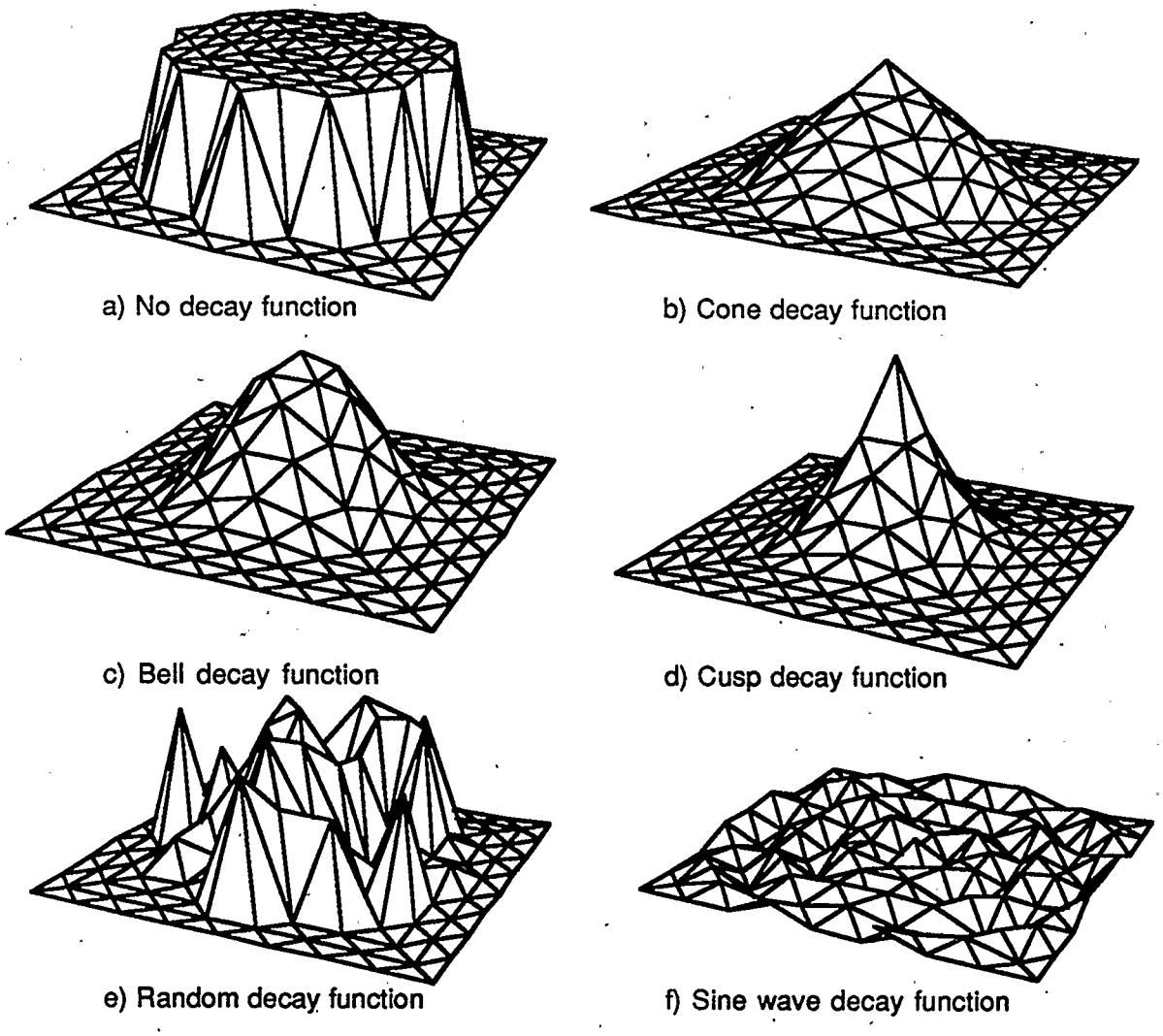


Figure 4.4: Decay Functions

moved. The next generalisation of the move vertex operation is the relaxation of the restriction that all vertices must move in the same direction as the current vertex (ie. along a parallel vector). Under this restriction, the move vertex operation is called the *Parallel Operation.* Some other operations are illustrated in Figure 4.5.

The *Grow Operation* has the effect of causing the object to expand or contract by moving each vertex along its normal vector. The normal vector of a vertex is the average of the normal vectors of its incident faces.

The *Stretch Operation* has the effect of stretching the mesh as though it were made of some elastic material.

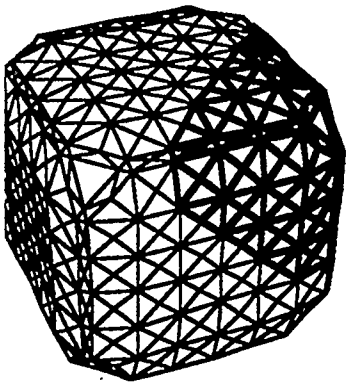
The *Randomise Operation* moves each vertex by a randomly chosen vector. The length of the random vector is restricted by the length of the movement vector. The effect of fractal geometry [Fournier *et al* 82] can be achieved by controlling the randomness, and recursively subdividing the mesh (Section 4.3.8).

The *Smooth Operation* has the effect of smoothing the object by reducing sharp discontinuities of slope.

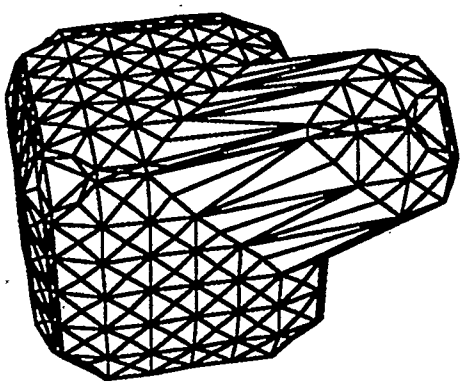
The list of operations above is not exhaustive, but indicates that a large number of effects can be achieved by experimenting further with the operations performed when moving a vertex.

4.3.6 Binding Vertices

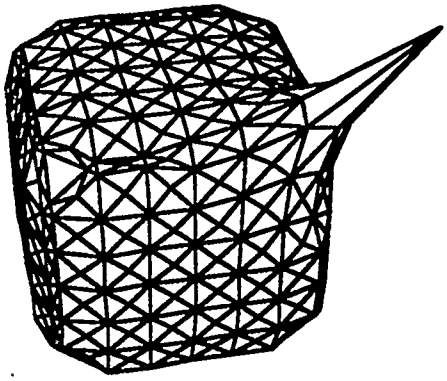
Together, the range of influence, decay function, and move vertex operation offer a powerful set of capabilities. However, they have the limitation that only vertices which lie in a connected sub-mesh can be moved. Often, it is necessary to move individual vertices without affecting any vertices which lie between them. To this



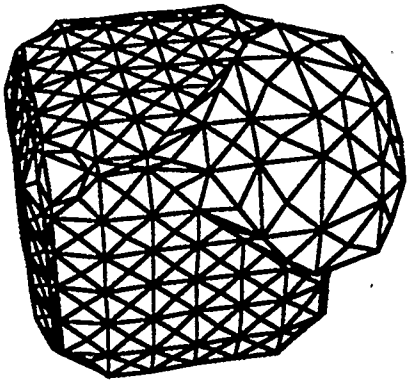
a) Range of influence



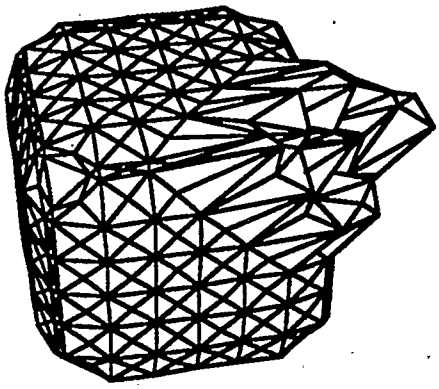
b) Parallel operation



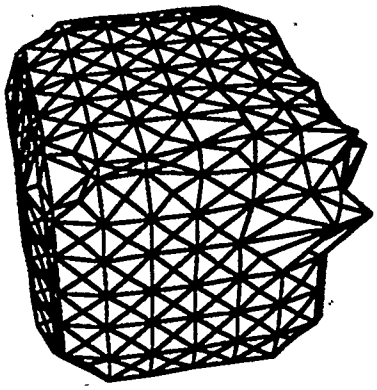
d) Stretch operation



c) Grow operation



e) Randomise operation



f) Smooth operation
Applied once to object in (e).

Figure 4.5: Variations of the Move Vertex Operation

end, individual vertices can be *bound* to the current vertex. Each bound vertex behaves exactly as the current vertex does. When the current vertex is moved, each bound vertex is moved by the same vector.

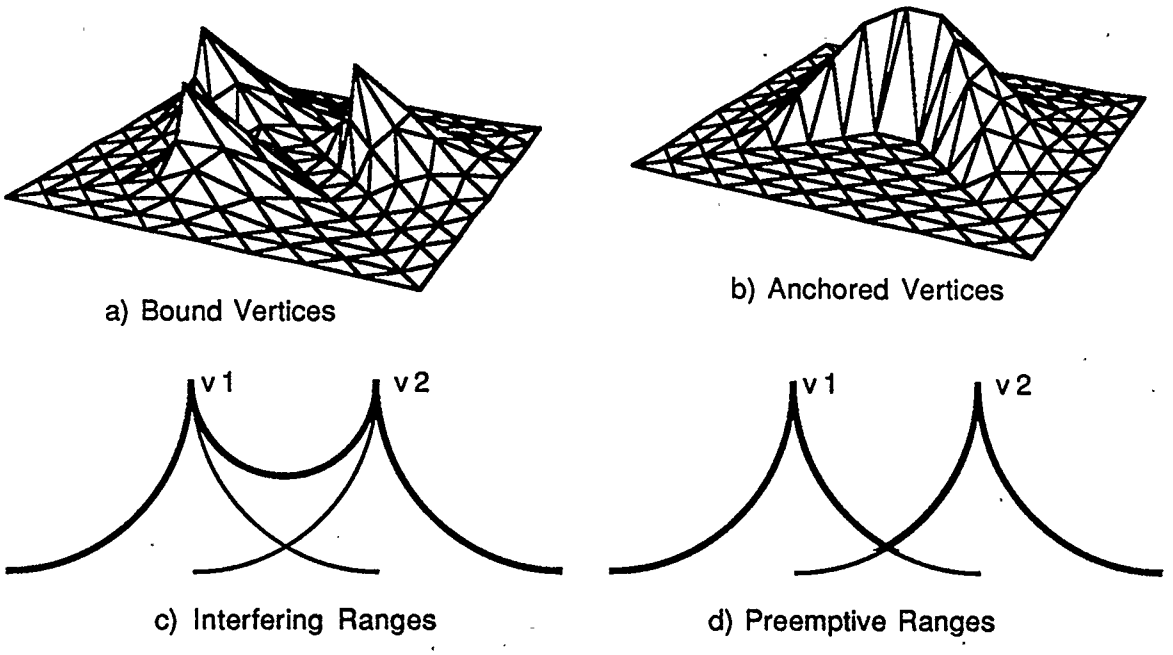


Figure 4.6: Bound and Anchored Vertices

In the case where there is a range of influence defined, the same size range around each bound vertex is also influenced (Figure 4.6.a). Vertices in these ranges behave in exactly the same way as those around the current vertex. When two or more ranges overlap, some vertices are influenced by more than one vertex. This situation can be handled in several ways. For example, the vertex can be moved by each bound or current vertex, producing an interference effect (Figure 4.6.c), or it can be moved by only the nearest one (Figure 4.6.d).

4.3.7 Anchoring Vertices

The next development in the methodology of polygon mesh modelling is the ability to restrict certain vertices from moving. This is done by *anchoring* them to their current 3D positions so that they cannot be moved, even if they lie within the range of a current or bound vertex.

The effect of moving the current vertex could be said to radiate outward from the current vertex toward the limits of the range. The effect stops when it reaches an anchored vertex. By anchoring a connected path of vertices, a designer can effectively create a dyke through which the effect cannot radiate. This can be used to create an abrupt edge on an otherwise smooth shape (Figure 4.6.b).

4.3.8 Topological Changes

All of the editing mechanisms discussed so far have modified the geometry of the object. There are two topological modifications which would be most useful to polygon mesh modelling. The first is the ability to sub-divide a polygon into smaller polygons. The need for this arises when an area in which detailed work is to be done contains too few polygons. The new polygons occupy the same area as the old one and lie in the same plane as it did.

The second useful topological change is the inverse of the first, the ability to combine two or more adjacent, co-planar polygons into one. This ability is more for convenience than necessity. Reducing the number of polygons simplifies wire frame drawings making them easier to understand. Combining polygons also reduces the number of polygons which need to be stored in computer or secondary memory.

These operations correspond to the local Euler operations MEV, MEF, KEV,

and KEF (Section 3.2), and do not alter the Euler characteristic of the mesh.

Chapter 5

DELTA – A POLYGON MESH MODELLING SYSTEM

DELTA is a graphically interactive modelling system for polygon mesh modelling. It has been designed to interact with other components in the Graphicsland computer animation system at the University of Calgary [Wyvill *et al* 86c]. The most recent version of DELTA runs on Iris 3000 series workstations[1]. Previous versions of DELTA ran on Raster Technologies Model I graphics processors[2], Corvus Concept workstations[3], and Sun 3 workstations[4]. The capabilities have evolved with the various versions. Each new version runs on a faster device thereby improving system response time.

This chapter first outlines the capabilities of DELTA, then describes the interesting implementation details.

Some segments of pseudo-code have been included for illustration. Although DELTA is written entirely in the 'C' programming language [Kernighan/Ritchie 78], the pseudo-code is similar to Pascal. Two 'C' style deviations from Pascal are the use of "`->`" to specify a field of a structure referenced through a pointer (eg. "`pointer->field`") and the `return` statement.

[1]Silicon Graphics Inc.
[2]Raster Technologies Inc.
[3]Corvus Inc.
[4]Sun Microsystems Inc.

5.1 Overview of Capabilities

The user dialogue is based on a set of pop-up menus. Table 5.1 lists the menus of operations available in DELTA (v. 4.4). These are described briefly in this section.

MAIN MENU
Confirm
Cancel
Help
Quit

FILE MENU
Read Object
Write Object
Write Ascii
Read Script
Interpolate
Read Template

SELECT MENU
Current Vertex
Set Range
Divide Range
Bind Vertex
Unbind All Verts
Anchor Vertex
Unanchor All Verts
Anchor Perimeter

DECAY MENU
No Decay
Linear
Cusp
Bell
Sine
Random
Parallel Operation
Grow Operation
Stretch Operation
Smooth Operation

VIEW MENU
Dolly
Pan
Orbit
Centre Object
Look At Vertex
Type View
All Visible
Front Face

EDIT MENU
Undo
Type Move
Repeat Move
Vertical
Vertex Normal
Edge Parallel
Plane Normal

Table 5.1: DELTA (v. 4.4) Menus

Main Menu Confirm and Cancel are used to exit a move vertex operation. They appear on this menu for convenience. Help provides a summary of instructions for using DELTA. Quit exits the system after giving the designer a chance to save any

modifications made since the last time the object was written to a file.

File Menu All file input and output operations appear on the File Menu. DELTA normally reads and writes objects in binary polygon files. Polygon mesh files would be faster to read and write but they are not supported by the Graphicsland animation system. Polygon files can be written in two forms: binary or textual. Binary files are smaller but can only be read by the type of computer which writes them. Textual files can be read by any type of computer or by humans.

DELTA writes a script file of operations automatically each time it is run. Read Script reads a script file and executes it as a program. Script files will be discussed in Section 5.7.

Interpolate produces a series of polygon files which can be used to create an animation of a move vertex operation.

DELTA edits one polygon mesh at a time, but a second mesh can be read from a polygon file and used as a guide or template for editing the first. The template object cannot be edited.

Select Menu The designer can select four kinds of vertices: the current vertex, bound vertices, anchored vertices, and vertices at the perimeter of the range. The Select Menu is used to select and de-select different types of vertices. Edges and polygons can also be selected but only in specific contexts. Generally, there are no special meanings attached to edges or polygons. Divide Range is used to sub-divide all of the triangles in the range of influence.

Decay Menu The decay functions and move vertex operations available in DELTA are listed on the Decay Menu. The decay functions (No Decay, Linear, Cusp, Bell, Sine, Random) apply to the Parallel and Grow operations only. Stretch and Smooth are not affected by the decay function.

View Menu DELTA provides many ways to change the view of the object being edited. The three graphically interactive techniques are Dolly, Pan, and Orbit (Section 5.5.2). The non-graphical techniques available are Centre Object and Look At Vertex, which position the view-to point at the centre of the object or at the current vertex. Type View allows the designer to specify the view-from and view-to coordinates textually.

The object being edited can be displayed with all polygons visible or with back facing polygons removed.

Edit Menu DELTA offers several methods of specifying where to move a vertex. Undo returns each vertex which was moved in the last move vertex operation to its previous position. Type Move allows the offset to the new position to be entered textually. Repeat Move applies the most recent offset vector to the (now) current vertex.

The remaining menu items are used to specify the new position graphically. The technique used is described in Section 5.5.1. In it, the designer specifies a 3D line through the current vertex called the *primary axis* of movement. There are four ways that the primary axis can be specified, corresponding to the remaining items on the Edit Menu.

5.2 Data Structure

5.2.1 Modified Polygon List Structure

The data structure used by DELTA to represent the polygon mesh is based on the polygon list structure presented in Section 4.1.4. There are three important differences. First, the polygon list references the vertex list directly instead of the edge list. This is because the polygon list is only used for back face removal (Section 5.4.1) and for writing the object to a data file. The vertices, not the edges, of a polygon are needed for these operations. Second, there are three vertex pointers, instead of a linked list or array of pointers, since every polygon has exactly three vertices. Third, each vertex record has pointers to each vertex which shares an edge with it (its *adjacent vertices*). These cross references are used to traverse the mesh laterally (see Section 5.6.1).

The face, edge, and vertex lists are implemented as arrays, rather than linked lists, to save on pointer space and reduce the complexity of traversals. It should be noted, however, that the traversal time is virtually the same for arrays and linked lists. For arrays, the traversing pointer is incremented by a constant for each element. For linked lists, it is assigned the value of a field of the record which it references. Since the length of the lists is not known initially, the arrays are extended when they overflow. If the vertex list must be relocated in order to be extended then all vertex pointers in the vertex, edge, and polygon lists must be offset accordingly. This overhead is negligible compared to the time taken to build the data structure, and has not been noticed for objects as large as 10^5 polygons. Offsetting is not necessary for the edge and polygon lists, since there are no references into these lists. Figure 5.1

presents pseudo-code for the DELTA data structure.

Polygon List The polygon list is used to determine which vertices belong to front facing polygons for the purpose of back face removal. For each polygon, the vertex pointers are used to determine if it is front facing. If so, the `visible` flag for each vertex is set to TRUE. The vertex pointers are stored in anti-clockwise order as seen from the front of the polygon, as required by the front-facing calculation (Section 5.4.1).

The polygon record also contains some colour information. Colours are either assigned to entire polygons or to individual vertices. Therefore, the polygon record contains a flag, `colourful`, indicating which scheme is used, and a colour table index, `colour`, for this polygon if its vertices are not coloured individually.

Edge List The edge record is the simplest of the three records, containing references to each of its end vertices. The edge list is traversed when drawing the object. For back face removal, only those edges for which both vertices are marked visible are drawn. If there were no edge list, and the object was instead drawn by traversing xn the polygon list, then edges which belong to two front facing polygons would be drawn twice.

Vertex List The vertex record is the most complex, since it contains most of the specific information needed for polygon mesh modelling. There are several fields for the coordinates of each vertex:

```
polygon = record
    v1, v2, v3:          vertex_ptr;
    colourful:           bool;
    colour:              integer;
end;

edge = record
    v1, v2:              vertex_ptr;
end;

vertex = record
    point, new_pnt, old_pnt: point;
    pixel:               pixel;

    adjacent:            array of vertex_ptr;
    joined:              array of bool;
    n, lim:              integer;

    bound, anchored:     bool;
    visible, inrange:    bool;
    position_value:      real;
    serial:              integer;

    coloured:            bool;
    red, green, blue:    real;
    left, rite:          vertex_ptr;
end;

vertex_ptr  = ^vertex;
```

Figure 5.1: DELTA Polygon Mesh Data Structure

pnt	World coordinates of vertex.
new_pnt	Tentative new location for vertex during move vertex operation (used so the designer can cancel the operation).
old_pnt	Position of vertex before it was last moved (for undo).
pixel	Device coordinates (2D) where vertex was last drawn (for selecting vertices and back face removal).

The vertex list is traversed linearly to transform the object into device coordinates.

Pointers are stored to each adjacent vertex, since polygon mesh modelling requires that the mesh be traversed laterally from one vertex to the next. The pointers are stored in an extendible array since different vertices have different numbers of incident edges. Another extendible array, `joined`, indicates whether there is a polygon joining each consecutive pair of edges. This information is used to determine the boundaries of open meshes and for calculating vertex normals. The integers `n` and `lim` are used to maintain the extendible arrays.

A number of other fields are used specifically for mesh editing besides the coordinate fields already mentioned. The integer field, `serial`, is used as a time stamp (Section 5.6.1), and the boolean fields `bound`, `anchored`, and `inrange` are state flags.

The flag `coloured` is set to true if this vertex has a colour of its own, in which case the `red`, `green`, and `blue` fields will hold its colour value.

The `left` and `right` vertex pointers are used to build a binary tree of vertices when reading a polygon file. This is discussed in the Section 5.3.

5.2.2 Polygon List Structure versus DCEL

This section compares the modified polygon list structure with the DCEL. DELTA has the following requirements for the polygon mesh data structure:

1. Fast traversal of vertex, polygon, and edge lists for transformation, back face removal, and display of the object.
2. Fast lateral traversal from vertex to adjacent vertices for propagation of move vertex operations.
3. Access to all vertices of a polygon for polygon normal calculations.
4. Access to all adjacent vertices for vertex normal calculations.
5. Minimal space usage.

Fast data structure maintenance is not critical, since the data structure is relatively static once a mesh is read. Both data structures satisfy the first two requirements equally well. Accessing all vertices of a given polygon is a more complex process for the DCEL. This could be remedied by replacing the edge reference in the polygon structure with vertex references as was done with DELTA's data structure. Accessing all adjacent vertices of a single vertex is also more complex for the DCEL.

Analysis of Space Usage

The space requirements of the two structures are nearly identical. Vertex coordinates take the same amount of space for both structures, as does the extra information specifically for polygon mesh modelling. Therefore, the significant space requirement

for comparison is that used for pointers. The following lemma is needed to analyse the space usage of the two data structures.

Lemma 1. The Average Number of Edges per Vertex for Triangulated Polyhedra without Holes is Not Greater than 6.

Proof. Euler's formula for polyhedra with no holes is:

$$V - E + F = 2 \tag{5.1}$$

For closed, triangulated polyhedra the number of face–edge pairs is $2E$ (2 faces per edge) and $3F$ (3 edges per face). Therefore:

$$2E = 3F \tag{5.2}$$

Substituting (5.2) into (5.1) we have:

$$V = \frac{F}{2} + 2 \tag{5.3}$$

The average number of edges per vertex is $2E/V$ since there is a total of $2E$ edge–vertex pairs distributed over V vertices. Substituting for E and V:

$$\frac{2E}{V} = \frac{2(3/2F)}{F/2+2} = \frac{6F}{F+4} \tag{5.4}$$

For $F = 1$, (5.4) evaluates to $6/5$, and as F approaches infinity, (5.4) asymtotically approaches 6.

From Lemma 1, it follows that a polyhedron with F triangles will have $3/2F$ edges and $F/2 + 2$ vertices. Table 5.2 shows, for both structures, the number of pointers required to represent a polyhedron as a function of the number of polygons. The DCEL has one pointer in each polygon record and four in each edge record. The polygon list structure has three pointers in each polygon record, two in each edge record, and a variable number of pointers to adjacent vertices in each vertex record.

The number of vertices adjacent to a given vertex is the same as the number of edges it has. Lemma 1 proves that this is no greater than six on average.

The overall difference in space requirements is a small factor (9/7) in favour of the DCEL.

	Relation to F	Pointers in DCEL	Pointers in Polygon List
F	F	1	3
E	3/2 F	4	2
V	F/2 + 2	0	6
Total		F + 6F = 7F	3F + 3F + (3F + 12) = 9F + 12

Table 5.2: Number of pointers in DCEL versus Polygon List Structure

To summarise, the advantages of the DCEL are a small, linear space savings and the fact that all structures are of fixed length. The advantages of the polygon list structure are that it admits simpler (ie. faster) algorithms for some of the required traversals. It also has a conceptually simpler hierarchical structure as opposed to the lateral structure of the DCEL.

5.3 Building the Polygon Mesh Data Structure

DELTA reads polygon files and builds polygon list data structures out of them. All shared vertices and edges will appear in more than one polygon. There is no prescribed order to the polygons in the file. Therefore, as each polygon is read, its vertices and edges must be checked to see if they already exist in the data structure. Pseudo-code for the procedure `read_file()` is shown in Figure 5.2. The efficiency of the algorithm depends most on the function `add_vertex()` and the procedure

add_edge(), which check to see if a vertex or edge already exist in the data structure. Additional efficiency is gained by checking the vertices adjacent to v1 and v2 first when looking for v2 and v3 (function find_neighbour()).

```
procedure read_file ()
var
    p1,p2,p3: point;
    v1,v2,v3: vertex_ptr;
begin
    while (not eof) begin
        read polygon(p1, p2, p3);

        v1 := add_vertex(p1);
        v2 := find_neighbour(p2, v1);
        if (v2 is empty) v2 := add_vertex(p2);
        v3 := find_neighbour(p3, v1);
        if (v3 is empty) v3 := find_neighbour(p3, v2);
        if (v3 is empty) v3 := add_vertex(p3);

        add_edge(v1,v2);
        add_edge(v2,v3);
        add_edge(v3,v1);

        add_face(v1,v2,v3);
    end; {while}
end; {read_file}
```

Figure 5.2: Read_File Procedure

Add Vertex The function add_vertex() dominates the time required to read a polygon file. It checks whether the point passed to it already exists in the data structure. This requires a thorough search of all vertices read so far. For this reason, DELTA organises the vertices into a binary tree as they are read. The ordering in

the tree is determined by comparing the x, y, and z coordinates one at a time.

$$P_1(x_1, y_1, z_1) = P_2(x_2, y_2, z_2) \text{ iff } \begin{cases} x_1 = x_2 \text{ and} \\ y_1 = y_2 \text{ and} \\ z_1 = z_2 \end{cases}$$

$$P_1(x_1, y_1, z_1) < P_2(x_2, y_2, z_2) \text{ iff } \begin{cases} x_1 < x_2 \text{ or} \\ x_1 = x_2 \text{ and } y_1 < y_2 \text{ or} \\ x_1 = x_2 \text{ and } y_1 = y_2 \text{ and } z_1 < z_2 \end{cases}$$

The "greater than" relation is analogous to "less than".

Add Edge The procedure `add_edge()` runs in constant time by first checking to see if `v2` is a neighbour of `v1`. If it is not, then `v1` and `v2` define a new edge. Therefore, the run time depends only on the number of neighbours that `v1` already has. Lemma 1 on page 70 proves that this number is not greater than six on average.

Time Complexity to Build Polygon Mesh Data Structure

The time complexity of the inside of the while loop in the `read_file()` procedure is $O(\log V)$, where V is the number of vertices. For triangulated polyhedra $V = F/2+2$, so this is the same as $O(\log F)$. The while loop is executed F times so the overall run time complexity of the `read_file()` procedure is $O(F \log F)$.

5.4 General Geometry

The geometric algorithms used in DELTA are described in two sections: this section describes the solutions to some general geometric problems which can be applied in many situations. These were derived primarily from [Rogers/Adams 76] and

[Allan/Wyvill 87]. The next section describes the application of these general geometric solutions to the specific problems of 3D interaction that have been solved for DELTA.

Due to the finite precision with which digital computers represent real numbers, all real number comparisons must take a tolerance factor into consideration. When comparing two real numbers, a and b, for equivalence, one must check if $|a - b| < \epsilon$, where ϵ is the tolerance factor.

5.4.1 Cross Product Calculations

The cross product of two vectors is a third vector perpendicular to both:

$$\bar{u} \times \bar{v} = \left(\begin{vmatrix} u_2 & u_3 \\ v_2 & v_3 \end{vmatrix}, \begin{vmatrix} u_3 & u_1 \\ v_3 & v_1 \end{vmatrix}, \begin{vmatrix} u_1 & u_2 \\ v_1 & v_2 \end{vmatrix} \right)$$

DELTA uses cross products to perform three different calculations: polygon normal, front facing, and triangle membership.

The normal of polygon (p_1, p_2, p_3) is the cross product of the two vectors $(p_2 - p_1)$ and $(p_3 - p_1)$.

In 2D, cross product has no definition since three mutually orthogonal vectors imply a vector space of dimension not less than three. However, it is often useful to compute a cross product of 2D vectors assuming 0 to be the third component of each vector. The resultant vector will always be of the form $(0, 0, z)$, where z will be positive iff the first vector lies "to the right of" the second.

Once transformed into device coordinates, the coordinates of the vertices of a triangle can be used to determine (1) if the polygon is facing the viewer, and (2) if another 2D point is inside the projection of the triangle. The polygon is front facing

if the z component of the normal vector is positive. A point q is inside the projection of triangle (p_1, p_2, p_3) if the z components of the cross-products of $(q-p_i)$ and (p_j-p_i) each have positive signs (where p_j is the next vertex after p_i).

5.4.2 Projecting Points onto Lines in 2D

When the designer picks edges on the screen, the coordinates of the locator are projected onto the candidate edges to see which one is nearest. For a line represented as a 2D vector (x_v, y_v) and an offset from the origin (x_o, y_o), the point (x_p, y_p) projects to:

$$[(x_v, y_v) \cdot (x_p - x_o, y_p - y_o)]\ (x_v, y_v) + (x_o, y_o)$$

that is:

$$\begin{aligned} x\prime &= (x_v(x_p - x_o) + y_v(y_p - y_o))x_v + x_o \\ y\prime &= (x_v(x_p - x_o) + y_v(y_p - y_o))y_v + y_o \end{aligned}$$

This generalises to 3D for projection onto lines and planes, however DELTA does not make use of this.

5.4.3 Intersecting 3D Lines with Planes

DELTA uses the following algorithm to determine the position on the target plane indicated by the mouse cursor.

The line and plane are represented as:

$$\begin{aligned} x &= a + tl \\ y &= b + tm \\ z &= c + tn \end{aligned} \tag{5.5}$$

and

$$Ax + By + Cz + D = 0 \tag{5.6}$$

where (a, b, c) is a point on the line and (l, m, n) is its direction vector. Substituting the respective components of (5.5) into (5.6):

$$A(a + tl) + B(b + tm) + C(c + tn) + D = 0. \tag{5.7}$$

Solving for t yields:

$$t = -\left(\frac{Aa + Bb + Cc + D}{Al + Bm + Cn}\right) \tag{5.8}$$

Substituting (5.8) into (5.5) above yields the point of intersection. The only constraint on (5.8) is that the denominator cannot equal zero. Note that the denominator is the dot product of the line's direction vector with the plane's normal. When this dot product is zero, the two vectors are perpendicular indicating that the line and the plane being intersected are parallel.

5.4.4 Finding the Nearest Points on two Lines in 3D

DELTA uses the following algorithm to determine the position on the primary axis indicated by the mouse.

Lines in 3D are represented parametrically:

$$(a, b, c) + s(l, m, n) \text{ and } (d, e, f) + t(p, q, r), \tag{5.9}$$

where (a, b, c) and (d, e, f) are points, s and t are independent scalar parameters, and (l, m, n) and (p, q, r) are direction vectors. The nearest points are found by minimising the distance between the lines:

$$\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \tag{5.10}$$

where:

$$\begin{aligned} \Delta x &= a + sl - (d + tp) \\ \Delta y &= b + sm - (e + tq) \\ \Delta z &= c + sn - (f + tr) \end{aligned} \tag{5.11}$$

The minimum of (5.10) is the same as the minimum of:

$$\Delta x^2 + \Delta y^2 + \Delta z^2 \tag{5.12}$$

Equation (5.12) reduces to a second degree polynomial with two variables which can be expressed as a function of s and t for some constants A, B, C, D, E, and F:

$$f(s,t) = As^2 + Bt^2 + Cst + Ds + Et + F \tag{5.13}$$

The minimum of this function occurs when:

$$\frac{\partial f}{\partial s} = 2As + Ct + D = 0 \quad \text{and} \quad \frac{\partial f}{\partial t} = 2Bt + Cs + E = 0 \tag{5.14}$$

Solving for s and t yields:

$$s = \frac{(2BD - CE)}{(C^2 - 4AB)} \quad \text{and} \quad t = \frac{(2AE - CD)}{(C^2 - 4AB)} \tag{5.15}$$

If the denominator $(C^2 - 4AB)$ equals zero then the two lines are parallel. Substituting s and t back into (5.9) yields the two nearest points on the two given lines.

5.4.5 Viewing and Perspective Transformations

DELTA presents the designer with a perspective projection of the object from an arbitrary point of view. The standard transformation pipeline from world coordinates to device coordinates is performed using the Iris Geometry Engine [Clark 82]. This specialised set of chips accepts 3D lines as input; performs matrix multiplication, 3D

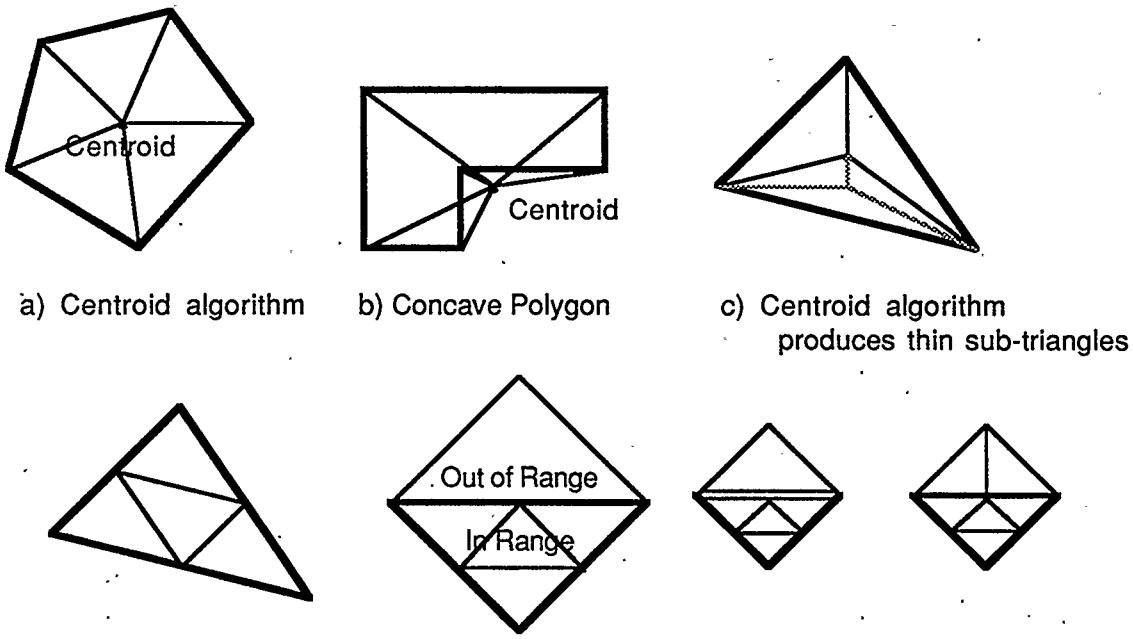
line clipping, perspective or parallel projection, and device coordinate mapping; and draws the resulting 2D vectors on the screen.

The inverse of the viewing transform is maintained by DELTA for mapping viewing coordinates back into world coordinates. It is used to interactively specify changes to the viewing coordinates and to move vertices. Since the viewing transform, V, is constructed from primitive affine transformations it is not necessary to invert V. Instead, V^{-1} is constructed by concantenating the inverse primitive transformation matrices in reverse order.

5.4.6 Polygon Subdivision

The last general geometric procedure to be discussed is the subdivision of polygons into triangles. Since most rendering algorithms used in computer graphics require planar polygons, DELTA triangulates the polygon mesh as it is Read. Triangulation is a large research field itself with much effort going into finding optimal solutions both in terms of time [Garey *et al* 78], and properties of the triangulation produced [Klincsek 80]. DELTA uses two simple triangulation algorithms: one to triangulate polygons as they are read from a polygon file, the other to subdivide triangles to make the mesh finer.

When a polygon with more than three vertices is read, it is assumed to be convex and is triangulated as follows. A new vertex is added at the *centroid* (average of all vertex coordinates) of the polygon. Each original vertex is then connected to the new one with a new edge creating as many triangles as there were vertices in the original polygon (Figure 5.3.a). Figure 5.3.b illustrates how the algorithm will fail for some concave polygons.



a) Centroid algorithm

b) Concave Polygon

c) Centroid algorithm produces thin sub-triangles

d) Mid-point triangle subdivision

e) Sub-dividing the range: Triangles just out of range have 4 vertices

f) Two solutions to problem triangles

Figure 5.3: Triangulating Polygons and Triangles

DELTA uses a different algorithm to subdivide triangles, since the centroid algorithm produces proportionally thinner triangles than it starts with (Figure 5.3.c). This is undesirable since it produces a less uniformly distributed sampling of the surface represented by the mesh. The subdivision algorithm used produces four sub-triangles by adding a new edge between the mid-points of each pair of edges in the original triangle (Figure 5.3.d). The advantages of this algorithm are that the sub-triangles produced are geometrically similar to the original one, and that it can be applied recursively to the sub-triangles. Note that the number of triangles produced increases geometrically with the level of subdivision. That is, subdividing a triangle *level* times produces 4^{level} triangles.

In DELTA, all triangles with three vertices in the range of influence can be divided into sub-triangles with a single operation. Only one level of subdivision is applied, but further levels can be achieved by re-applying the operation to the same range.

Subdividing the range creates a problem along the range boundary. An edge between two polygons, one in range, the other one not, will be divided into two edges by the in-range polygon making the out-of-range polygon four sided (Figure 5.3.e). One solution is to turn the boundary edge into a degenerate triangle with three colinear edges. Such a triangle causes numerical problems if, for example, its area is used as a denominator or its normal is needed. A better solution is to divide the out-of-range triangle into two triangles by connecting the new vertex to the opposite vertex (Figure 5.3.f).

5.5 3D Interaction Geometry

In this section the general geometric solutions presented above are applied to the 3D interaction problems encountered in DELTA; moving vertices in 3D and specifying the viewing parameters. The solutions all use a three button mouse to control a cursor (the *mouse cursor*) on the workstation screen.

5.5.1 Moving Vertices in 3D

Refer to Figure 5.4 for this discussion. The designer specifies a 3D line through the

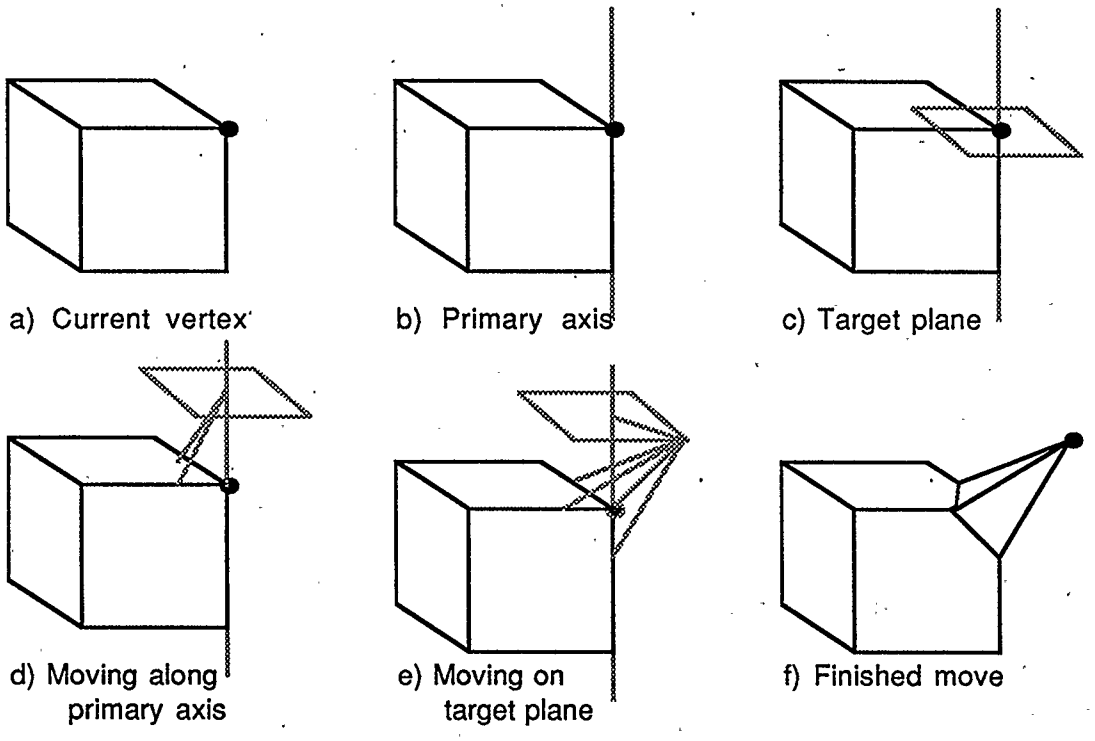


Figure 5.4: Moving a Vertex — Primary Axis and Target Plane

current vertex called the *primary axis* of movement (Figure 5.4.b). There are several methods of choosing the vector of the primary axis listed in Table 5.3.

Vertex Normal	The normal vector passing through the current vertex.
Edge Parallel	The vector parallel to an edge of the current vertex. This edge is selected interactively with the mouse.
Face Normal	The normal vector to a face of the current vertex. This face is selected interactively with the mouse.
Vertical	The vector (0,1,0) which is up in the world coordinate system.
Type Move	The vector typed by the designer in global coordinates.

Table 5.3: Methods of Determining the Primary Axis of Movement

The first three methods define the primary axis relative to the geometry of the object. The last two methods define it relative to the world coordinate system. The primary axis in Figure 5.4.b is vertical. By itself, the primary axis offers the designer one degree of freedom along which to move the current vertex.

Perpendicular to the primary axis is a plane called the *target plane*. Initially, the target plane passes through the current vertex (Figure 5.4.c). The designer can move the current vertex anywhere on the target plane. Thus, the designer has two more degrees of freedom, allowing him to move the current vertex anywhere in 3D space.

One mouse button is used to move the target plane perpendicular to the primary axis. Another is used to locate the new vertex position on the target plane. After choosing the primary axis, the designer can repeatedly manipulate the new location for the current vertex with both mouse buttons. The primary axis and target plane (represented by a 3D square polygon) are drawn as the vertex is moved. The updated

positions of all vertices in the range are shown as well. The move vertex operation can be completed or cancelled at any time by the designer. DELTA enables the designer to change the view, the decay function, operation, or the range without interrupting a move vertex operation.

Implementation of Move Vertex Operation The primary axis and target plane technique for moving vertices uses two of the geometric solutions presented in Section 5.4: finding the point of intersection between a line and a plane, and finding the nearest points on two 3D lines.

Recall that the technique uses one mouse button to move along the primary axis and another to move on the target plane. For each device coordinate generated by the mouse cursor, the line of projection (projector) through that device coordinate is transformed into world coordinates (using the inverse of the viewing transformation matrix). When the designer is specifying movement along the primary axis, the point on the primary axis closest to the transformed projector is found (Figure 5.5.a). The target plane is positioned to pass through this point.

When the designer is specifying movement on the target plane, the projector is intersected with it to find the new vertex position (Figure 5.5.b). The new vertex position is stored as a vector relative to the centre of the target plane (where it intersects the primary axis).

5.5.2 Specifying the View Graphically

DELTA provides three graphically interactive techniques for specifying the view: dolly, pan, and orbit (Figure 5.6). Only the view-from and view-to points are altered.

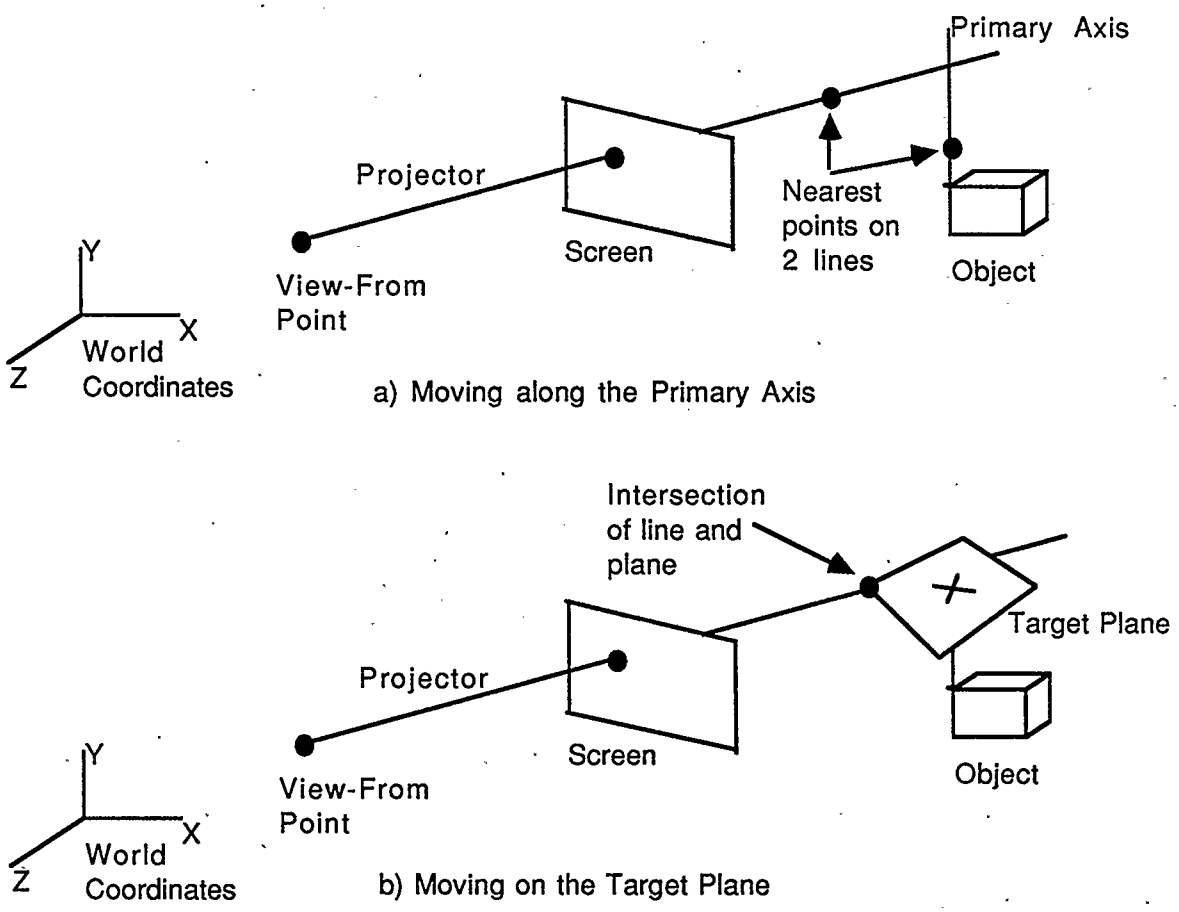


Figure 5.5: Geometry of the Move Vertex Operation — for clarity, the cube is not triangulated

The *view vector* is the vector difference, (view-from – view-to).

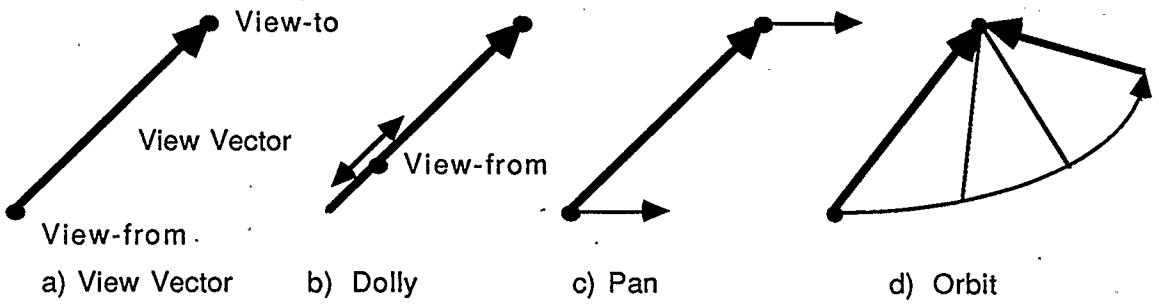


Figure 5.6: Interactive Viewing Techniques

Dolly moves the view-from point toward or away from the view-to point along the view vector. There is only one degree of freedom in this process so the mouse is used as a valuator [Foley/Wallace 74]. Moving the mouse left causes the view-from point to dolly out and moving it right causes the view-from point to dolly in.

Pan moves the view-from and view-to points by the same offset vector perpendicular to the view vector. The offset vector is related to the incremental change in the mouse cursor position. Thus, moving the mouse cursor up and right will have the effect of moving the objects in the viewport up and right. The offset vector is found by transforming the device coordinates for two consecutive mouse cursor positions into world coordinates (using the inverse viewing transformation matrix) and taking the vector difference between them.

Orbit moves the view-from point around the view-to point maintaining the distance between them. Moving the mouse cursor up and right will have the effect of spinning the objects in the viewport up and right about the view-to point. An offset vector in world coordinates is found as for Pan, but it is added to the view-from

point only. The view-from point is then moved toward the view-to point to maintain the length of the view vector.

Mapping Device Space to World Space One problem with graphically interactive view specification techniques is mapping the device space offset into the appropriate world space offset. Due to the perspective projection, the projectors through the previous and current device coordinates are divergent (Figure 5.7). As they get farther from the view-from point they get farther apart. The problem is establishing an appropriate distance from the view-from point to measure the distance between them.

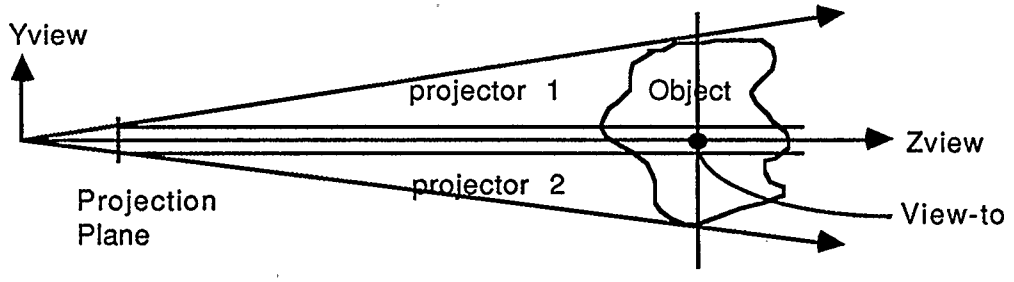


Figure 5.7: Diverging Projectors

The distance between projectors where they intersect the projection plane seems, at first, a reasonable choice. However, as Figure 5.7 shows, the width of the projection window can be insignificant relative to the object being viewed. Moving the mouse the full width of the device would produce a world space offset only a fraction of the width of the object; even though the object entirely fills in the viewport.

The distance at the view-to point is a better choice because the designer can move the view-to point toward or away from the view-from point without affecting

the view. By positioning the view-to point near the object being viewed, the distance between projectors is the distance that the designer intends[5]. The offsets used for Pan, Dolly, and Orbit are scaled by the ratio of the length of the view vector to the distance from the view-from point to the projection plane.

Singularities

Two singularities can occur with the graphically interactive viewing techniques. The view-from and view-to points can converge, or they can become vertically aligned.

Using DOLLY can cause the points to converge by moving the view-from point to the view-to point. When this happens, none of the graphical techniques will work (since they each scale by the length of the view vector). Non-graphical techniques must be used to separate the view-from and view-to points when this happens.

Using ORBIT can cause the view-from and view-to points to become vertically aligned. When this happens the object appears to suddenly spin to one particular view. This is due to the sense of "up" assumed by the viewing transformation [Newman/Sproull 79].

Both of these conditions could be avoided by detecting when they are about to occur and preventing the view-to point from quite converging or aligning. In order to provide a predictable interface, a history of recent views must be kept to determine the direction to repel the view-to point.

[5]This is one reason why DELTA provides the facilities to position the view-from point at the centre of the object or at the position of the current vertex.

5.6 Polygon Mesh Modelling

This section presents the implementation details for the different components of the polygon mesh modelling paradigm presented in Section 4.3.

5.6.1 Range of Influence

Both methods of defining the range of influence, maximum number of edges and maximum distance, were implemented. Their implementation details are similar so only the latter is described.

For efficiency, all vertices in the range are marked as such with the `inrange` flag and their distance from the nearest current or bound vertex is stored in the `position_value` field.[6] The range is marked every time it is changed, whether changed explicitly or by selecting a new current vertex, by (un)binding or (un)anchoring vertices, or by subdividing the range.

The `adjacent` pointers in the vertex list are used to perform a *flood traversal* of the mesh. Assume for now that there are no bound vertices. The flood traversal processes the current vertex, then visits each of the vertices adjacent to it. The traversal floods outward until all vertices in the range have been visited.

The flood traversal lends itself to recursive implementation. Pseudo-code for the procedure is given in Figure 5.8.

Before calling `flood_traverse()` a global counter, `Serial`, is incremented to a new value unique to this traversal. When a vertex is visited it is stamped with the serial number of this traversal so that it will only be processed once.

[6] Actually, the normalised position within the range is stored $(1 - (distance/Range))$.

```
var /* Global variables */
    Serial: integer;
    Range, Range_squared: real;

procedure flood_traverse (vertex,centre: vertex_ptr);
var
    distance_squared, position_value: real;
begin
    vertex->serial := Serial;

    distance_squared := sum_of_squares_of_differences(vertex, centre);
    if (distance_squared > Range_squared) return;

    position_value := 1.0 - sqrt(distance_squared) / Range;
    if (not vertex->inrange) or
       (position_value > vertex->position_value) then begin
          vertex->position_value := position_value;
          vertex->inrange := true;
    end;

     for i := 1 to vertex->n do begin
        next_vertex = vertex->adjacent[i];
        if  (next_vertex->serial <> Serial) and
            (not next_vertex->in_range)     and
            (not next_vertex->anchored)     then
                flood_traverse(vertex->adjacent[i], centre);
    end;
end; {flood_traverse}
```

Figure 5.8: Recursive Flood Traverse Procedure

The traversal routine stops whenever it encounters a vertex which is out of range of the current vertex. For efficiency, the square of the range is compared with the square of the distance. This avoids evaluating the square root for vertices which are out of range.

The `flood_traverse()` procedure calls itself recursively for each adjacent vertex unless any of three conditions is true: (1) the next vertex is already stamped with the serial number of this traversal, or (2) it is already marked as being in range, or (3) it is anchored.

If there are bound vertices, then for each bound vertex, the global `Serial` is set to a new value and `flood_traverse()` is called again. Vertices which are within range of more than one current or bound vertex are processed once for each and retain the lowest position value.

Analysis of Flood Traverse Procedure Marking the vertices in the range as such and storing their position values requires $O(N+B)$ calls to `flood_traverse()`, where N is the number of vertices in the range and B is the number of vertices just beyond the range. Since B is at most the number of vertices in the range times the number of vertices adjacent to each, the upper bound on B is directly proportional to N (recall from Lemma 1 on page 70 that the maximum average number of adjacent vertices per vertex is six). Thus, marking a range of N vertices takes $O(N)$ calls to `flood_traverse()`.

5.6.2 Decay Functions

The decay functions illustrated in Figure 4.4 on page 55 have been implemented in DELTA. With each incremental mouse cursor position in a move vertex operation, the position value for each vertex is mapped into a scale factor by one of the functions listed in Table 5.4. The movement vector is then scaled by this factor, added to the vertex position, and stored in the `new_pnt` field.

Constant	$f(x) = 1.0$	Does not change the shape of the range. This is equivalent to having no decay function.
Cone	$f(x) = x$	The scale factor decreases linearly from the current vertex to the limit of the range.
Cusp	$f(x) = x^2$	The simplest quadratic produces a cusp shaped protrusion from a flat surface.
Bell	$f(x) = sin((1-x)\frac{\pi}{2})$	Maps the input range onto one quarter of a sine wave cycle. This produces a bell shaped curve from a flat surface.
Wave	$f(x) = sin(K x Range)$	The sine of some constant K times the distance from the moving vertex to the current vertex (x $Range$) produces a wave effect.
Random	$f(x) = \text{rand}()$	A random number in the range $[0 \rightarrow 1]$ with equal distribution produces a randomly perturbed surface.

Table 5.4: Decay Functions (x = position value)

5.6.3 Updating the Display

While moving a vertex or specifying the view graphically, the display is updated with each incremental move of the mouse. On the Iris, this is done by redrawing the entire object in a secondary frame buffer (the back buffer) then swapping the front and back buffers during the monitor horizontal retrace. No erasure is done, producing a smoothly changing image without flicker. The object is drawn by traversing the edge list. Thus, every edge is supplied to the Geometry Engine exactly once.

The double frame buffer technique has the disadvantage that large objects cannot be updated quickly enough. (Even with specialised hardware it is always possible to push the limits of speed.) Therefore, the display algorithm checks the position of the mouse cursor periodically (every 1000 vectors or so). If the cursor has been moved more than a threshold distance, then the redraw is aborted and the back buffer is displayed with the partially redrawn object. This produces the effect that the display always keeps pace with the mouse, but if the mouse moves too fast and the object is too large then the display will degrade momentarily.

5.7 DELTA Script Files

DELTA writes a script file of all editing operations performed. These script files contain instructions and parameters in a human readable format. They can be examined and modified directly like a computer program, then re-executed by DELTA on some, possibly different, object. An example script file is shown in Figure 5.9. Complex operations such as changing the view or moving a vertex are represented by tokens which summarise their effect.

```
# TRANSFORM A CUBE INTO A CHAIR

# SQUASH
currentvertex 0.000000 1.000000 0.000000
bindvertex -0.500000 1.000000 0.500000
bindvertex -0.500000 1.000000 -0.500000
bindvertex 0.500000 1.000000 -0.500000
bindvertex 0.500000 1.000000 0.500000
range 1.224745
decayfunction Linear
movevertex 0.000000 -0.468986 0.000000

# DIVIDE RANGE
dividerange
dividerange
unbindall
range 0.000000

# LEGS
currentvertex -0.500000 -0.086061 0.500000
bindvertex -0.250000 -0.086061 0.500000
bindvertex -0.375000 -0.086061 0.375000
bindvertex -0.500000 -0.086061 0.250000
bindvertex -0.500000 -0.086061 -0.500000
bindvertex -0.500000 -0.086061 -0.250000
bindvertex -0.375000 -0.086061 -0.375000
bindvertex -0.250000 -0.086061 -0.500000
bindvertex 0.250000 -0.086061 -0.500000
bindvertex 0.375000 -0.086061 -0.375000
bindvertex 0.500000 -0.086061 -0.250000
bindvertex 0.500000 -0.086061 -0.500000
bindvertex 0.500000 -0.086061 0.500000
bindvertex 0.250000 -0.086061 0.500000
bindvertex 0.375000 -0.086061 0.375000
bindvertex 0.500000 -0.086061 0.250000
movevertex 0.000000 -0.908793 0.000000
unbindall
```

```
# ROUND BACK
currentvertex -0.500000 0.531014 0.000000
range 0.000000
anchorvertex -0.125000 0.473706 -0.500000
anchorvertex -0.125000 0.473706 0.500000
anchorvertex -0.125000 0.531014 -0.375000
anchorvertex -0.125000 0.531014 0.375000
anchorvertex -0.250000 0.531014 -0.250000
anchorvertex -0.250000 0.531014 0.000000
anchorvertex -0.250000 0.531014 0.250000
anchorvertex -0.375000 0.473706 -0.500000
anchorvertex -0.375000 0.473706 0.500000
anchorvertex -0.500000 0.376745 -0.500000
anchorvertex -0.500000 0.376745 0.500000
anchorvertex -0.500000 0.416399 0.000000
anchorvertex -0.500000 0.416399 0.250000
anchorvertex -0.500000 0.473706 -0.125000
anchorvertex -0.500000 0.473706 -0.375000
anchorvertex -0.500000 0.473706 0.125000
anchorvertex -0.500000 0.473706 0.375000
anchorvertex 0.000000 0.531014 -0.500000
anchorvertex 0.000000 0.531014 0.500000
range 1.891632
decayfunction Bell
dividerange
movevertex 0.000000 1.213332 0.000000
```

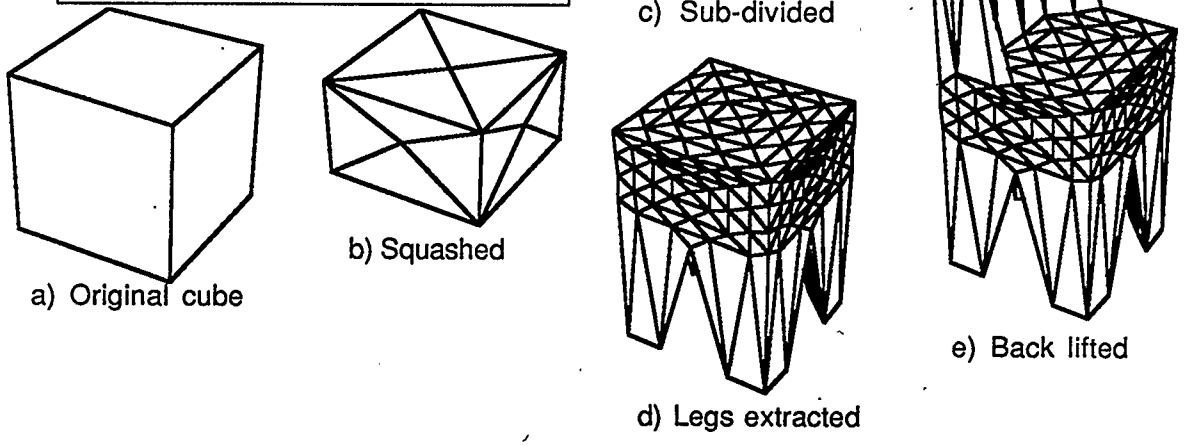


Figure 5.9: Example DELTA Script File

In order to be useful, scripts must be deterministic. There is one case where DELTA scripts are non-deterministic. The random decay function may produce a different effect each time it is executed. This could be corrected by using a seeded random number generator and saving the seed with the script.

Advantages of Script Files

Reduced File Storage Space. A large, complex polygon mesh object can be stored efficiently as a small, simple polygon file and a script file. For example, the script in Figure 5.9 transforms a unit cube into the complex polygon mesh object shown in Figure 5.9.e. This is a special case of *data base amplification*, the practice of storing small amounts of high-level information which can be used to re-create large amounts of low-level information.

Programmability. As mentioned, script files can be written and edited like computer programs. For example, if the four legs of the chair in Figure 5.9.d had been made with four separate move vertex operations, the script file could have been edited to make them the same length.

N-step Undo. DELTA provides a one step undo facility by storing previous values of vertices when they are moved. An N-step undo facility would require saving extensive state information because the move vertex operation is not reversible. The set of vertices in the range of influence changes when the current vertex is moved. Script files provide a crude N-step undo capability by allowing the designer to truncate the script and re-execute it on the original object.

Debugging. Due to the determinism of most editing operations, scripts have been very valuable for correcting bugs during development of DELTA. The ability to edit

and re-execute scripts expedited the isolation of errors.

Starting with Different Meshes. A coarse mesh can be used to develop an editing script quickly, then the script can be re-executed on a finer version of the same object. This is a form of prototyping which can be used to save both time and storage space. The finer mesh must contain a vertex at the same location as each vertex identified in the script or the results will be undefined.

Transforming Polygon Meshes. It may be less expensive to transform a small polygon mesh object and the parameters of a script file before executing it, than it is to transform the object afterward. This is especially true if the script subdivides the polygon mesh a great deal.

Animation. The move vertex operation can be interpolated parametrically by scaling the movement vector by values between 0 and 1. A sequence of object descriptions can then be used to produce an animation of the operation. This capability has been implemented using a simple linear interpolation of the movement vector. More complex animation would be possible by allowing several operations to occur during overlapping time periods. This has not been explored due to time constraints.

Chapter 6

RESULTS

The results of this research are presented in four parts:

1. The polygon mesh modelling technique is analysed according to the criteria presented in Chapter 2.

2. The analysis of each modelling technique surveyed in Chapter 3, and of polygon mesh modelling are summarised and discussed.

3. Several examples of objects modelled with DELTA are presented.

4. Polygon mesh modelling is compared to the other techniques.

6.1 Analysis of Polygon Mesh Modelling

Polygon mesh modelling has been presented in Chapter 4 and an implementation described in Chapter 5. The criteria for CG modelling techniques are now applied to analyse polygon mesh modelling.

Expressive Power Polygon meshes can represent open or closed surfaces. Holes in closed surfaces are possible as are breaks in open surfaces. The topological class of the input cannot be changed. Polygon meshes only have continuity of position (C^0) between polygons.

Geometric Consistency Self-intersections are easily created when moving vertices. Planarity of polygons is not a problem since the polygon mesh is triangulated.

Precision and Resolution All geometric properties can be specified explicitly. However, the results of moves involving complex ranges, decay functions, and operations can be difficult to predict. The resolution is increased locally as needed by dividing the range.

Representational Fidelity Polygon meshes are exact representations in that their description is identical to the model they represent.

Convertability Most other modelling representations can be converted into polygonal approximations which can be stored as polygon meshes. As well, digitiser data can be transformed into polygon meshes. Conversion to other representations is generally not possible.

Space Efficiency Storage requirements for polygon meshes are large since all topological and geometric information must be stored explicitly, both in memory and on disc. Disc storage space can be considerably reduced with the use of script files.

Manipulation Speed Modification of the data structure is relatively fast using the flood traversal. Transformation is slow for large meshes because each vertex must be transformed separately. Wire frame display is very fast since all edge information is stored explicitly.

Computational Efficiency Extents, intersections, surface normals, and surface area are very fast to compute. Point membership, relevant only for bounding sur-

faces, is more complex since a candidate point must be compared with every face to determine membership.

Applicability to Animation The move vertex operation offers a high level of control for manipulating the mesh. However, it is possible to compromise geometric consistency by intersecting a mesh with itself. Objects can be animated by applying moves over time.

Application to Rendering Geometric information is readily available for polygonal models. Scan line coherence can be well utilised due to the planar property of polygons. Sorting can also be used because of the discrete nature of polygons.

6.2 Summary of the Analysis of Modelling Techniques

Tables 6.1 – 6.3 summarise the analysis of each modelling technique surveyed in Chapter 3, and of polygon mesh modelling.

They form an objective basis for comparison among the techniques. Each technique has advantages and disadvantages, and each is best suited to particular modelling tasks. Some general observations can be made about the modelling techniques as a result of the summary.

First, expressive power is probably the strongest determinant in evaluating the suitability of a technique. No desirable feature can make a technique suitable if it cannot represent the required model.

Second, very few conversions are possible among techniques. In fact, no conversions exist that are exact and apply generally to all models of the respective

	EULER OPERATIONS	QUADRICS
Expressive Power	Closed surfaces	Open and closed surfaces
	Holes	No holes or breaks
	C^0	C^2
Precision	High	Low
Resolution	Variable	Infinite
Representational Fidelity	High	Low
Geometric Consistancy	Low	High
Convertability	*from* most others	*to* polygon mesh
Space Efficiency	Low	High
Computational Efficiency	High	Moderate
Manipulation Speed	Moderate	Moderate
Applicability to Animation	Low flexibility	High flexibility
	Low control	Low control
Applicability to Rendering	High	Moderate

Table 6.1: Summary of Analysis of Modelling Techniques (part 1 of 3)

	ISO-SURFACES	SPLINE SURFACES (Interpolating/Approximating)
Expressive Power	Closed Surfaces	Open and closed surface
	Holes	Holes and breaks
	C^2	C^1/C^2
Precision	Low	Moderate/Low
Resolution	Infinite	Infinite
Representational Fidelity	Moderate	Moderate
Geometric Consistancy	High	Low
Convertability	*to* polygon mesh	*to* polygon mesh
		from smooth surfaces (approx.)
Space Efficiency	High	Moderate
Computational Efficiency	Moderate	Low
Manipulation Speed	Moderate	High
Applicability to Animation	High flexibility	Moderate flexibility
	High control	Moderate control
Applicability to Rendering	Moderate	Moderate

Table 6.2: Summary of Analysis of Modelling Techniques (part 2 of 3)

	CSG	Polygon Mesh Modelling
Expressive Power	Solids	Open and closed surfaces
	Holes	Holes and breaks
	C^2	C^0
Precision	High	High
Resolution	Infinite	Variable
Representational Fidelity	High	High
Geometric Consistancy	High	Low
Convertability	*to* polygon mesh	*from* most others
	from solid primitives	
Space Efficiency	High	Low
Computational Efficiency	Moderate	High
Manipulation Speed	Moderate	High
Applicability to Animation	High flexibility	Moderate flexibility
	High control	High control
Applicability to Rendering	Moderate	High

Table 6.3: Summary of Analysis of Modelling Techniques (part 3 of 3)

techniques.

Third, there is a continuum, defined by several of the criteria, along which the techniques lie. At one end are those techniques which are higher in representational fidelity, precision, computational efficiency, and applicability to rendering. At the other end are those which are higher in geometric consistency, degree of continuity, space efficiency, and applicability to animation. Euler operations and polygon mesh modelling lie toward the first end, quadrics and iso-surfaces lie toward the second, and spline surfaces and CSG lie somewhere between.

6.3 Examples of Polygon Mesh Modelling

The models presented in this section are examples of objects which have been modelled using DELTA. They have each been chosen to illustrate some aspect of the polygon mesh modelling methodology or its application to a modelling problem.

6.3.1 Landscape

Figure 6.1 illustrates the landscape designed for the computer animation, "The Great Train Rubbery" [Wyvill *et al* 88]. Preemptive, overlapping ranges and bell decay functions were used to create the hills atop the "hoodoo" landforms and in the valley floor. It also demonstrates the integration of DELTA into the Graphicsland animation system; polygon meshes, iso-surfaces, and explicit polygons were all used for the film.

6.3.2 Animal head

Figure 6.2 illustrates the use of Divide Range to increase the resolution where needed for detail work, without increasing the size of the polygon mesh unnecessarily. The original mesh read by DELTA was a single square polygon. Bi-lateral symmetry was enforced by the system. It could have been accomplished by modelling the changes to one side, then editing the script to reflect the changes on the other side.

6.3.3 Grecian Urn

Figure 6.3 illustrates the use of DELTA to add finishing touches to a model produced with another modelling technique. The urn is a simple surface of revolution to which

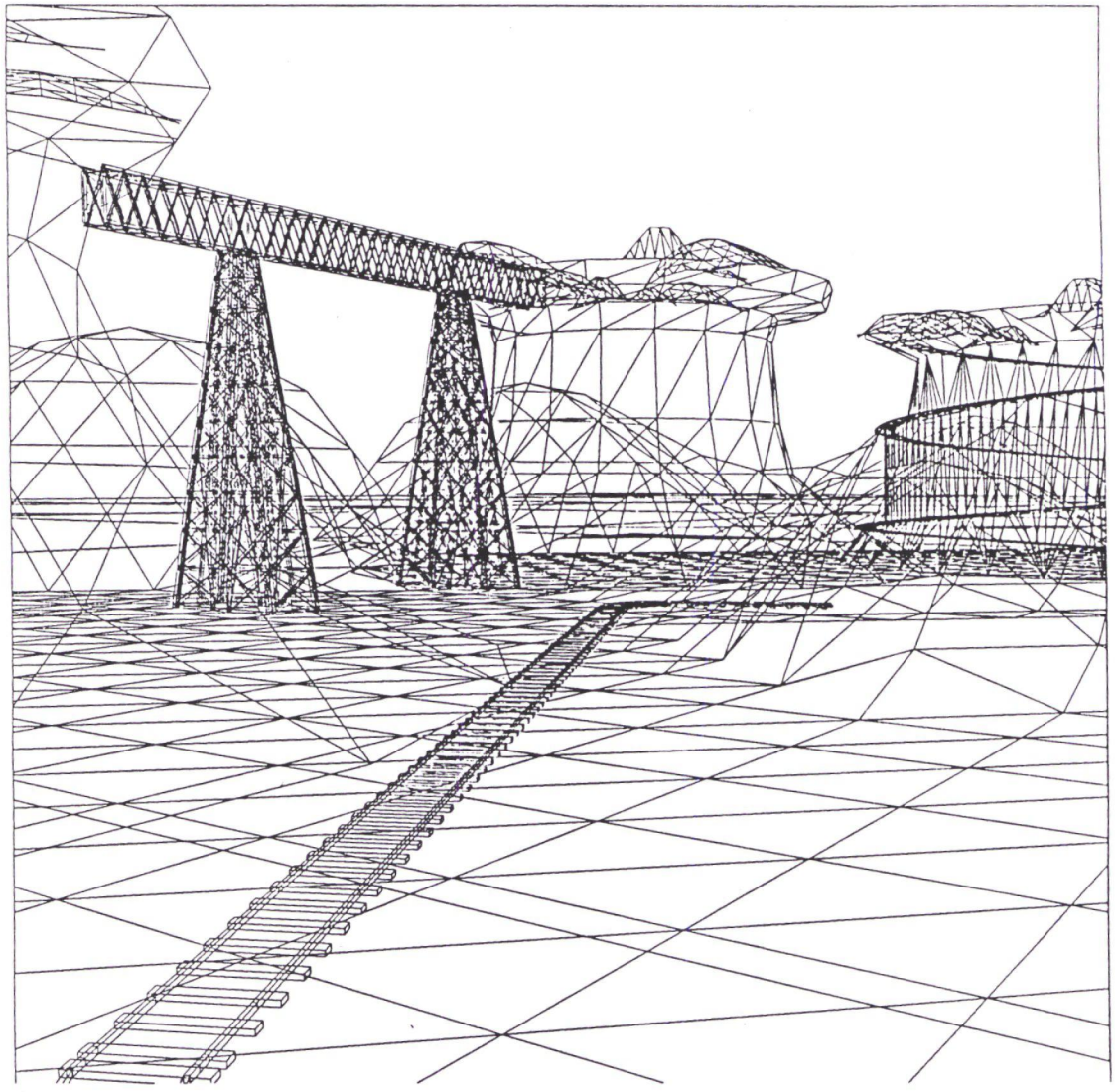


Figure 6.1: Landscape for "The Great Train Rubbery"

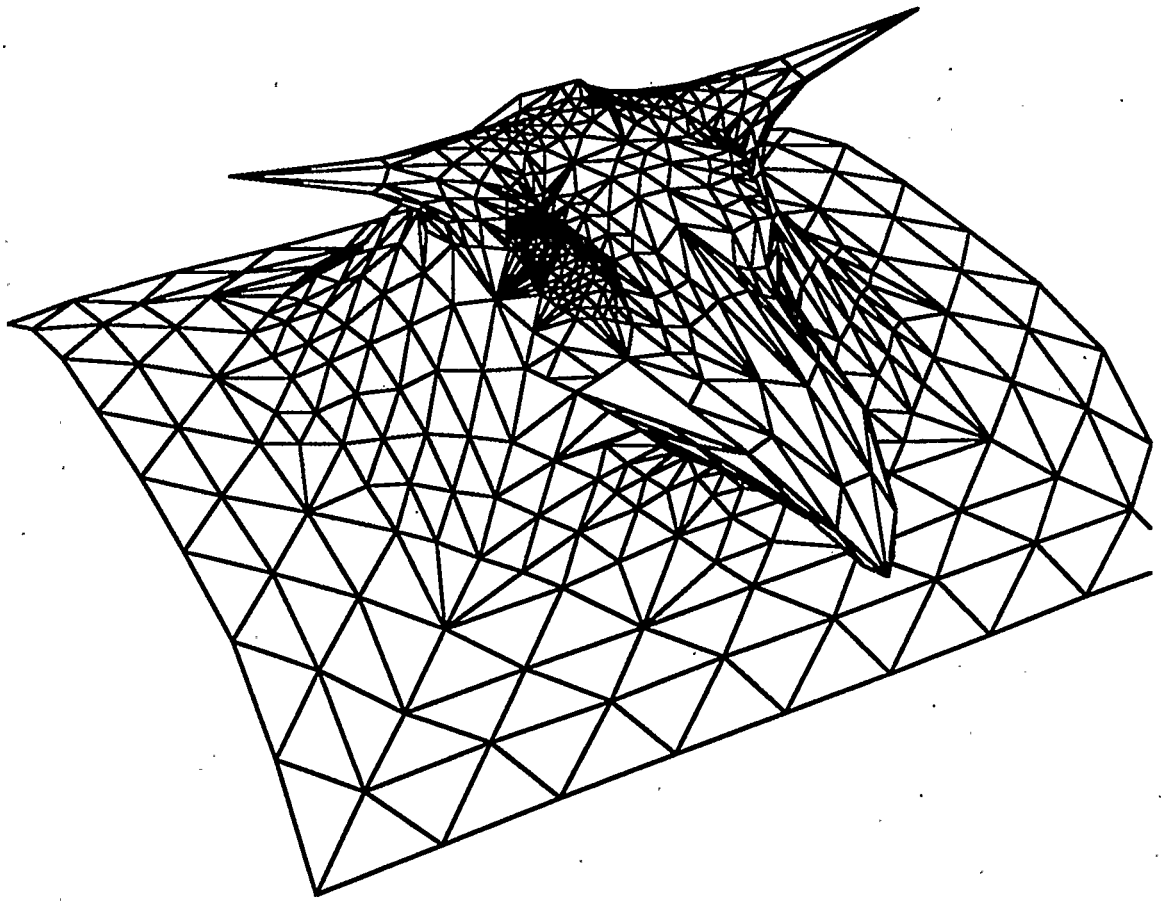


Figure 6.2: Animal head

a lip and handle were added. Both features were made by locally increasing the resolution of the mesh.

6.3.4 Application of DELTA to Prosthetic Medicine

DELTA has been applied to the process of generating prosthetic devices for below the knee amputees. The digitised data of the residual limb of an amputee are converted to a mesh (Figure 6.4.a) and modified interactively by the prosthetist to create the modified positive mold (Figure 6.4.b), which is fabricated with a digital milling machine. The physical positive mold is then used to create the socket of the prosthetic limb for the patient.

Sensitive areas of the limb data must be raised to provide relief and weight bearing areas depressed to provide support. The grow operation and bell decay function are used for these operations.

6.3.5 Menger Sponge

The abstract model shown in Figure 6.5 was created from a recursively self-similar polygonal object known as Menger's sponge [Mandelbrot 83]. It illustrates the application of polygon mesh modelling to models produced by other modelling techniques.

6.4 Comparison of Polygon Mesh Modelling to other Techniques

Polygon mesh modelling can be compared to the techniques surveyed in Chapter 3 by considering the example models presented above.

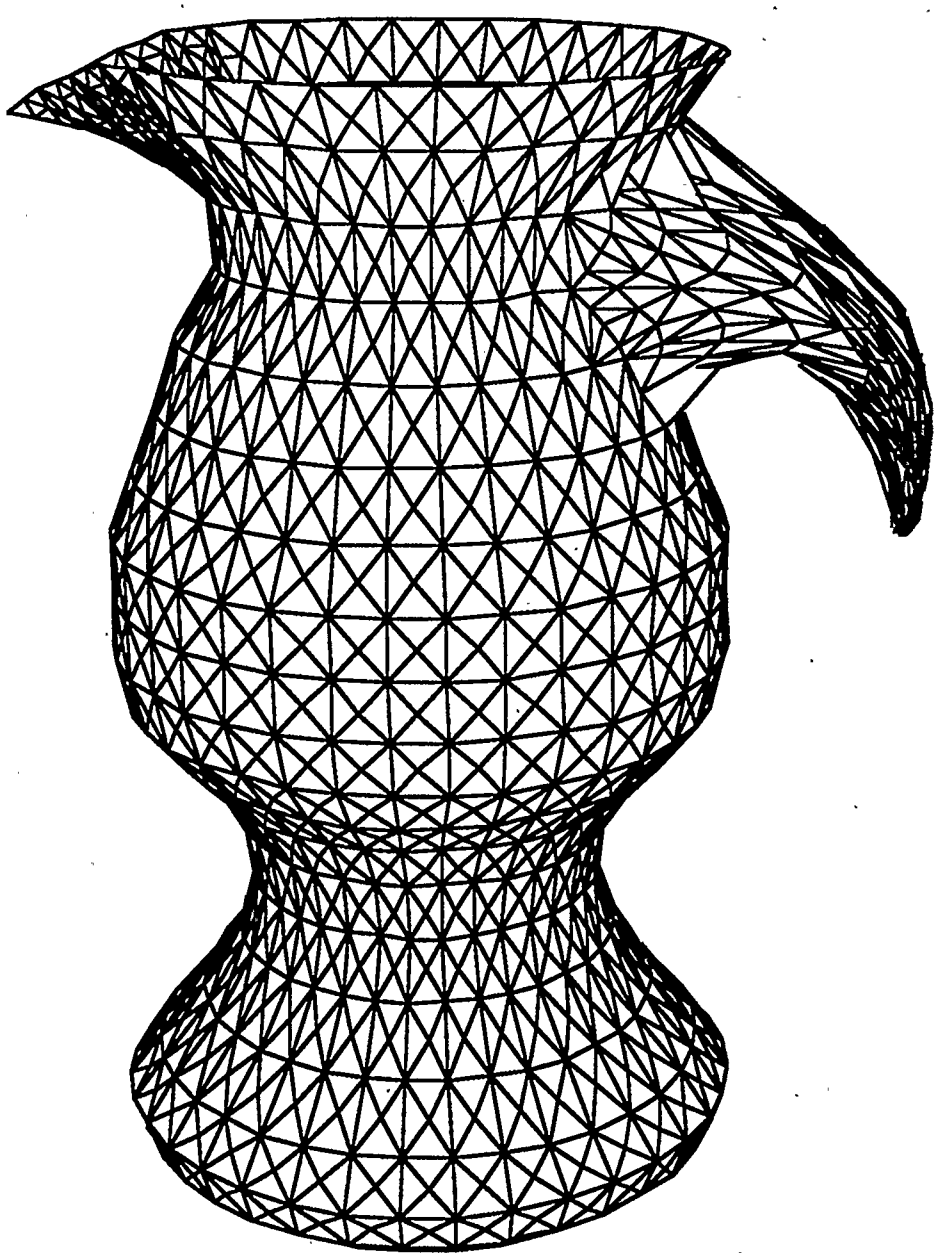
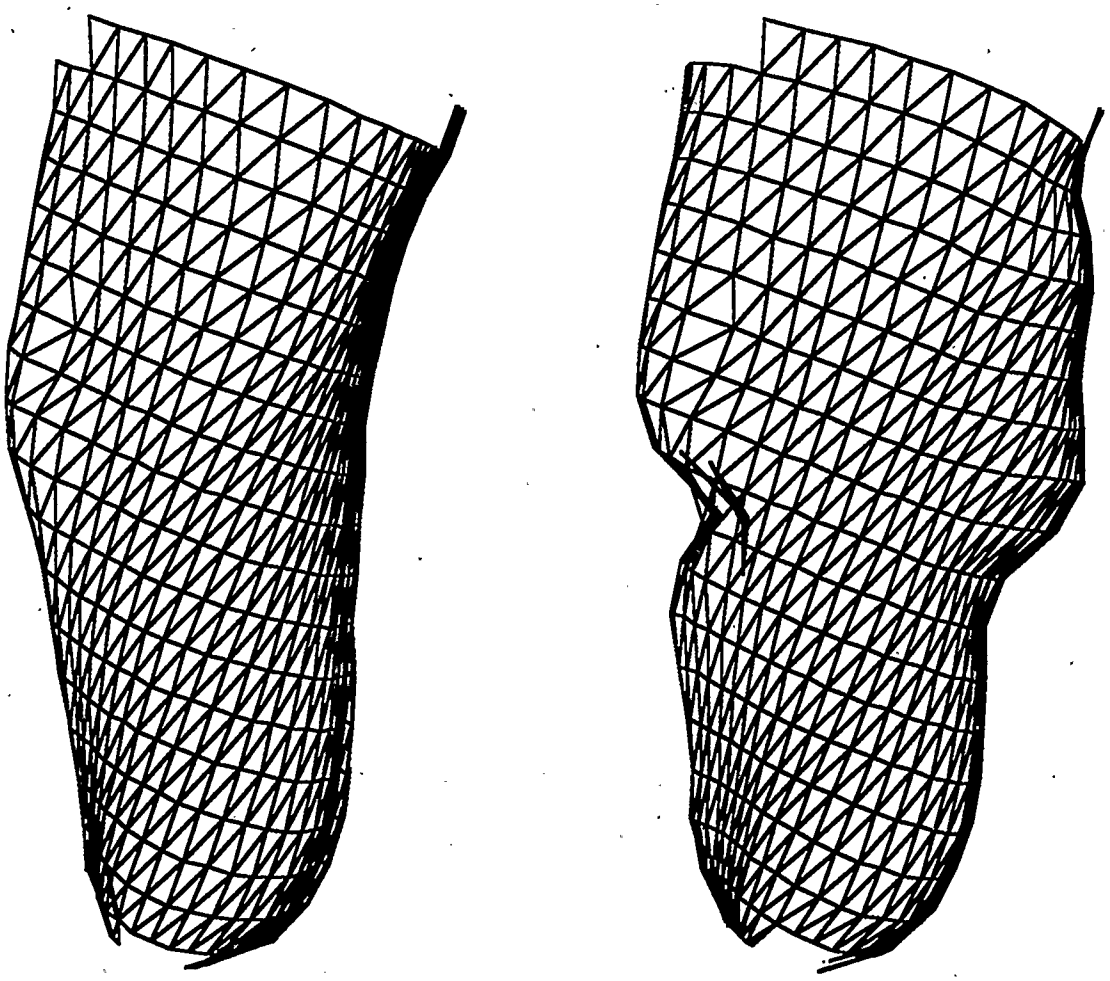


Figure 6.3: Grecian Urn with Lip and Handle



a) Residual limb

b) Modified positive mold

Figure 6.4: Residual Limb and Positive Mold

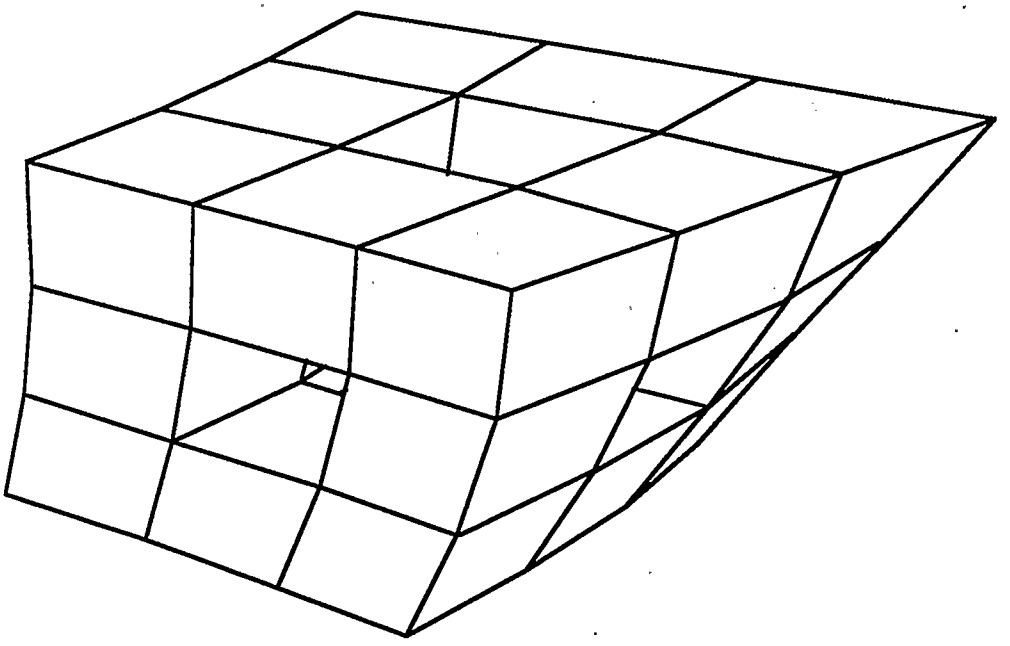


Figure 6.5: Modified Menger Sponge

The urn was modelled as a simple surface of revolution. The profile was edited with DELTA and revolved using the Graphicsland modelling system. The lip and handle were added using DELTA with a few move vertex operations. Local Euler operations would be infeasible for such a modelling task. It could be modelled as a spline surface, however this could require possibly dozens of patches. The urn does not resemble any of the quadric surfaces, and, since it is an open surface, it could not be modelled using constructive solid geometry or iso-surfaces.

The amputee limb data was provided as a cylindrical array of 3D points. Spline surfaces are the only other modelling technique that could be used to represent the limb. The data points could be used as the control matrix for an interpolating spline surface. There are two reasons why a polygon mesh is better suited, though. First, the resolution of the input is extremely high. There are as many as 512^2 data points

in the digitiser input[1]. With such fine resolution, interpolation is not necessary and would be extremely slow. Second, the input for the digital milling machine is the same as the output of the digitiser. A spline surface would have to be converted back into discrete points when output.

The animal head is based on an example in [Forsey/Bartels 88] used to illustrate refinement of B-splines. The model was reproduced with DELTA to demonstrate the equivalent capability of polygon mesh modelling.

The undeformed Menger sponge could be modelled with CSG or Euler operations, but neither of these techniques could readily produce the deformed version.

[1]Figure 6.4 shows only one point for every 100 in the model.

Chapter 7

CONCLUSIONS

7.1 Summary of the Thesis

Chapter 2 proposed a set of criteria for evaluating CG modelling techniques. These criteria were based on previous characterisation work carried out in the field of CAD.

Chapter 3 classified the modelling techniques commonly used for computer graphics into a taxonomy and surveyed a sample of them. The techniques were each evaluated using the criteria presented in Chapter 2.

Chapter 4 presented a new modelling methodology, called polygon mesh modelling, based on a simple move vertex operation and several extensions.

Chapter 5 presented the details of an implementation of a polygon mesh modelling system called DELTA, which was developed as part of the research for this thesis.

Chapter 6 presented the results of this research in four parts. The polygon mesh modelling technique was analysed using to the criteria presented in Chapter 2. The evaluations of the modelling techniques surveyed, and of polygon mesh modelling were summarised. Several examples of objects modelled with DELTA were presented. Polygon mesh modelling was compared to the surveyed techniques.

This chapter presents the conclusions of this research in three parts. First, applications of the criteria for modelling techniques are presented. Second, the viability of polygon mesh modelling is summarised. Finally, recommendations for further research are made.

7.2 Applications of the Criteria for CG Modelling Techniques

Chapter 2 presented the following list of criteria for evaluation and comparison of CG modelling techniques based on related work in the field of solid modelling.

1. Expressive Power
2. Precision and Resolution
3. Representational Fidelity
4. Geometric Consistency
5. Convertability
6. Space Efficiency
7. Computational Efficiency
8. Manipulation Speed
9. Applicability to Animation
10. Applicability to Rendering

The proposed CG criteria served as a standardised basis on which several modelling techniques, including polygon mesh modelling, were evaluated. There are three areas where the criteria can be applied.

First, computer graphic designers can use the criteria to choose a modelling technique. The criteria cannot be used to draw an overall comparative rating of the

techniques, since the importance of each criterion depends on the modelling task at hand. However, for a specific type of task, the importance of each criterion can be defined and used to make the decision.

Second, designers of CG systems can use the criteria as a basis for choosing among the many CG modelling techniques to implement. Again, the importance of each criterion will depend on the types of modelling tasks for which the CG system will be used.

Third, and possibly most important, the criteria can help researchers gain a better understanding of modelling techniques. They provide a set of terminology for thinking and communicating about CG modelling. They also serve to decompose the complex field of CG modelling into manageable concepts which can be explored in more detail individually, then synthesised into new concepts.

7.3 Polygon Mesh Modelling

Chapter 4 presented a methodology for polygon mesh modelling based on the following concepts:

- Current vertex,
- Move vertex operation (parallel and non-parallel),
- Range of influence,
- Decay function,
- Bound and anchored vertices,

- Subdivision of the range of influence.

Based on the analysis and examples presented in Chapter 6, the polygon mesh modelling methodology is a viable technique for CG modelling.

Polygon meshes can be used to represent any topological class of model: both open and bounding surfaces, with breaks or holes. They offer a high degree of modelling precision and representational fidelity, as well as high manipulation speed and computational efficiency. They are also fast to render.

The polygon mesh modelling methodology provides ease of use and general modelling power to the designer through the move vertex operation, range of influence, decay function, and bound and anchored vertices. The ability to divide all polygons in the range of influence allows the designer to change the mesh resolution as needed. Script files provide several capabilities, such as programmability and N-step undo.

Polygon meshes have the added advantages that they can be used to approximate most other modelling representations, and they can accurately represent digitised data.

The disadvantages of polygon mesh modelling are the planar nature of polygon meshes, the large storage space requirement, and the possibility of self-intersections.

The major disadvantage is probably the planar, faceted nature of the mesh itself. Even a finely divided mesh rendered with a smoothing algorithm will have a faceted silhouette. However, when polygon meshes must be used, due to available data or the nature of a modelling task, polygon mesh modelling offers capabilities which in most cases match, and sometimes surpass, those of other modelling techniques.

Script files can reduce the space requirement considerably. Self-intersections must be avoided by the designer, as with most surface modelling techniques.

7.4 Recommendations for Further Research

7.4.1 Criteria for CG Modelling Techniques

Although the criteria served as a standardised basis for analysis and comparison of CG modelling techniques, they have not been established as an appropriate metric for such analysis. A subject of assessment would have to be performed by a body of experts or through empirical experimentation, to determine the validity of the proposed criteria.

7.4.2 Polygon Mesh Modelling

The number and variety of research directions for polygon mesh modelling are very large. The following list is motivated by practical experience using DELTA.

Range of Influence The "shape" of the range of influence is always spherical. Differently shaped ranges could be useful (see regions of influence for soft objects in Section 3.4).

Decay Functions Rather than adding decay functions to the system *ad hoc*, it would be more useful to create a facility for the designer to specify new decay functions. Three possible ways to do this are: (1) provide a parser for analytic expressions, (2) allow the designer to graphically manipulate a curve (eg. cubic spline), or (3) interpolate a table of values generated by the designer.

Move Vertex Operations The set of move vertex operations could be extended to include operations such as twisting a mesh around an axis, causing one part of a mesh to resemble another, or replacing the vertex positions with the scaled movement vector rather than adding to the old positions.

Geometric Constraints There is already one form of geometric constraint, anchoring vertices to points in space. Vertices could be constrained to higher dimensional loci such as lines or planes.

Another form of geometric constraint involves the relationships among mesh components. The distance between any two vertices (not necessarily joined by an edge) or the angle between two edges or two polygons could be restricted. These restrictions could be either single values or ranges.

Surface Attributes Colour and other surface attributes could be assigned to vertices, edges, or polygons one component at a time, or edited using the range and decay function. For example, the decay function could be used to interpolate one colour at the current vertex into another at the range boundary.

Graphical 3D Interaction Techniques for view specification and point location could be investigated further experimentally.

Bibliography

[Allan/Wyvill 87] Jeffrey Allan and Brian Wyvill. The geometry tool box. Dept. Computer Science, University of Calgary. Internal research report no. 87/284/32, December 1987.

[Barr 81] Alan H. Barr. Superquadrics and angle-preserving transformations. *Computer Graphics and Applications*, 1(1), January 1981.

[Barsky/Beatty 83] Brian A. Barsky and John C. Beatty. Local control of bias and tension in beta-splines. *ACM Transaction on Graphics*, 2(2):109–134, April 1983.

[Bartels *et al* 87] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann, Los Altos, CA, 1987.

[Baumgart 75] Bruce G. Baumgart. A polyhedron representation for computer vision. *AFIPS Conference Proceedings*, 44:589-596, 1975.

[Bezier 74] Pierre E. Bezier. Mathematical and practical possibilities of UNISURF, in *Computer Aided Geometric Design*, Robert E. Barnhill and Richard F. Riesenfeld (Eds.), Academic Press, N.Y., 127–152, 1974.

[Blinn 82] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[Blinn 86] James F. Blinn. The algebraic properties of homogeneous second order surfaces. *ACM SIGGRAPH Tutorial Notes*, 1986.

[Bloomenthal 87] Jules Bloomenthal. Boundary representation of implicit surfaces. Xerox PARC internal report no. CSL-87-2. October 1987.

[Chiyokura/Kimura 83] H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. *Computer Graphics (SIGGRAPH 83 Proceedings)*, 17(3):289–298, July 1983.

[Clark 82] James H. Clark. The geometry engine: a VLSI geometry system for graphics. *Computer Graphics (SIGGRAPH 82 Proceedings)*, 16(3):127–133, July 1982.

[Foley/Wallace 74] James D. Foley and Victor L. Wallace. The art of natural graphic man-machine conversation. *Proceedings of the IEEE*, 62(4), April 1974.

[Foley/van Dam 83] J. D. Foley and A. van Dam. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, Reading, Mass., 1983.

[Forsey/Wilhelms 88] David R. Forsey and Jane Wilhelms. Techniques for interactive manipulation of articulated bodies using dynamic analysis. *Graphics Interface Conference Proceedings*, 8–15, June 6-10 1988.

[Forsey/Bartels 88] David R. Forsey and Richard H. Bartels. Hierarchical B-spline refinement. *Computer Graphics (SIGGRAPH 88 Proceedings)*, 22(4):205–212, August 1988.

[Fournier *et al* 82] Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models, *Communications of the ACM*, 25(6):371–384, June 1982.

[Gardner 85] Geoffrey Y. Gardner. Visual simulation of clouds. *Computer Graphics (SIGGRAPH 85 Proceedings)*, 19(3):297–303, July 1985.

[Gardan/Lucas 84] Yvon Gardon and Michel Lucas. *Interactive Graphics in CAD.* Kogan Page, London, 1984.

[Garey *et al* 78] Michael R. Garey, David S. Johnson, Franco P. Preparata, and Robert E. Tarjan. Triangulating a simple polygon. *Information Processing Letters*, 7(4):175–179, June 1978.

[Goldstein/Nagel 71] Robert A. Goldstein and Roger Nagel. 3-D visual simulation. *Simulation*, 16(1):25–31, January 1971.

[Gordon/Riesenfeld 74] William J. Gordon and Richard F. Riesenfeld. B-spline curves and surfaces, in *Computer Aided Geometric Design*, Robert E. Barnhill and Richard F. Riesenfeld (Eds.), Academic Press, N.Y., 95–126, 1974.

[Greenberg *et al* 86] D. P. Greenberg, M. F. Cohen, and K. E. Torrance. Radiocity: a method for computing global illumination. *Visual Computer*, 2(5):26–35, 1986.

[Hill 84] David R. Hill. Designing for human-computer interaction: some rules and their derivation. Dept. Computer Science, University of Calgary, Internal research report no. 84/166/24, 1984. also appeared in Jecl and Hide: some practical questions for the Jade user interface. *Proceedings CIPS Session 84*, Calgary, May 9–11, 373–380.

[Kernighan/Ritchie 78] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language.* Prentice-Hall, New Jersey, 1987.

[Klincsek 80] G.T. Klincsek. Minimal triangulations of polygonal domains. *Annals of Discrete Mathematics*, 9:121–123, 1980.

[Kochanek/Bartels 84] Dorris H.U. Kochanek and Richard H. Bartels. Interpolating splines with local tension, continuity and bias control. *Computer Graphics (SIGGRAPH 84 Proceedings)*, 18(3):33–41, July 1984.

[Kunii/Wyvill 85] Tosiyasu L. Kunii and Geoff Wyvill. A simple but systematic CSG system. *Graphics Interface Conference Proceedings*, 1985.

[Lane *et al* 80] Jeffrey M. Lane, Loren C. Carpenter, Turner Whitted, and James F. Blinn. Scan line methods for displaying parametrically defined surfaces. *Communications of the ACM*, 23(1):23–34, January 1980.

[Liewald/Kennicott 82] Michael H. Liewald and Philip R. Kennicott. Intersystem data transfer via IGES. *Computer Graphics and Applications*, 2(3):55–63, May 1982.

[Mandelbrot 83] Benoit B. Mandelbrot. *The Fractal Geometry of Nature.* W. H. Freeman, New York, 1983.

[Mantyla 88] Martti Mantyla. *An Introduction to Solid Modelling.* Computer Science Press, Rockville, Maryland, 1988.

[Newman/Sproull 79] William M. Newman and Robert F. Sproull. *Principles of Interactive Computer Graphics.* McGraw-Hill, New York, 1979.

[Nishimura *et al* 85] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura. Object modeling by distribution function and a method of image generation. *Journal of papers given at the Electronics Communication Conference '85*, J68-D(4), 1985. (in Japanese).

[Pratt 88] M.J. Pratt. Solid modelling - survey and current research issues. Tutorial Notes. State of the Art in Computer Graphics. Exeter, Devon, UK, July 4–8, 1988.

[Preparata/Shamos 85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: an introduction.* Springer-Verlag, New York, 1985.

[Prime 86] Prime Computer Inc. *Prime Medusa: technical summary.* Natick, Massachusetts, 1986.

[Prusinkiewicz/Streibel 86] Przemyslaw Prusinkiewicz and Dale Streibel. Constraint-based modeling of three-dimensional shapes. *Graphics Interface/ Vision Interface Conference Proceedings*, 158–163, May 1986.

[Reeves 83] William T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91–108, April 1983.

[Requicha/Voelcker 83] A.A.G. Requicha and H.B. Voelcker. Boolean operations in solid modelling: boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 3(7):30–44, Oct. 1983.

[Rogers/Adams 76] David F. Rogers and J.A. Adams. *Mathematical Elements for Computer Graphics.* McGraw-Hill Inc, 1976.

[Sutherland *et al* 74] Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. A characterization of ten hidden-surface algorithms. *Computing Surveys*, 6(1), March 1974.

[Thalmann *et al* 85] Nadia Magnenat-Thalmann and Daniel Thalmann. *Computer Animation: theory and practice.* Springer-Verlag, Tokyo, 1985.

[Thalmann *et al* 87] Nadia Magnenat-Thalmann and Daniel Thalmann. *Image Synthesis: theory and practice.* Springer-Verlag, Tokyo, 1987.

[Whitted 80] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[Wyvill *et al* 84] Brian Wyvill, Breen Liblong, and Norman Hutchinson. Using recursion to describe polygonal surfaces. *Graphics Interface Conference Proceedings*, 167–171, May-June 1984.

[Wyvill *et al* 86a] Brian Wyvill, Craig McPheeters, and Geoff Wyvill. Animating soft objects. *Visual Computer*, 2:235–242, 1986.

[Wyvill *et al* 86b] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *Visual Computer*, 2:227–234, 1986.

[Wyvill *et al* 86c] Brian Wyvill, Craig McPheeters, and Rick Garbutt. University of Calgary 3-D Computer Animation System. *Society of Motion Picture and Television Engineers Journal*, 95(6):629–636, June 1986.

[Wyvill 88] Brian Wyvill and Geoff Wyvill. Field functions for implicit surface. in *New Trends in Computer Graphics* (Proceedings of GC Internationsal '88). N. Magnenat-Thalmann and D. Thalmann (Eds.) Springer-Verlag, Berlin, 328–338, 1988.

[Wyvill *et al* 88] Brian Wyvill, *et al. The Great Train Rubbery.* A computer animation produced by the Graphicsland Animation Group, University of Calgary, in *ACM SIGGRAPH Video Review* (in press), 1988.

[Yamaguchi *et al* 84] K. Yamaguchi, T.L. Kunii, and K. Fujimura. Octree related data structures. *Computer Graphics and Applications*, 4(1):53–59, January 1984.