STANFORD ARTIFICIAL INTELLIGENCE LABORATORY
MEMO AIM-249

STAN-CS-74-463

GEOMETRIC MODELING FOR COMPUTER VISION

BY

BRUCE GUENTHER BAUMGART

SUPPORTED BY

ADVANCED RESEARCH PROJECTS AGENCY
ARPA ORDER NO. 2494

OCTOBER 1974

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

STANFORD ARTIFICIAL INTELLIGENCE LABORATORY
MEMO AIM 249
COMPUTER SCIENCE DEPARTMENT REPORT NO. CS-463

OCTOBER 1974

GEOMETRIC MODELING FOR COMPUTER VISION.

Bruce Guenther Baumgart

ABSTRACT:

The main contribution of this thesis is the development of a three dimensional geometric modeling system for application to computer vision. In computer vision geometric models provide a goal for descriptive image analysis, an origin for verification image synthesis, and a context for spatial problem solving. Some of the design ideas presented have been implemented in two programs named GEOMED and CRE; the programs are demonstrated in situations involving camera motion relative to a static world.

This research was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense under Contract No. DAHC 15-73-C-0435 The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Project Agency or the United States Government.

TABLE OF CONTENTS.

TABLE OF CONTENTS.

LIST OF BOXES.

LIST OF FIGURES.

ACKNOWLEDGEMENTS.

The following people personally contributed to this work:

Thesis Adviser: John McCarthy
Readers: Donald E. Knuth, Alan C. Kay, Ken Colby.

Jerry Agin, Leona Baumgart, Tom Binford, Jack Buchanan, Whitfield Diffie, Les Earnest, Jerome Feldman, Tom Gafford, Steve Gibson, Ralph Gorin, Carl Hewitt, Jack Holloway, Tovar Mock, Andy Moorer, Hans Moravec, Richard Orban, Ted Panofsky, Lou Paul, Phil Petit, Dave Poole, Lynn Quam, Jeff Raskin, Ron Rivest, Rod Schmidt, Clem Smith, Irwin Sobel, Robert Sproull, Dan Swinehart, Russell Taylor, Marty Tenenbaum, Larry Tesler, Arthur Thomas, Fred Wright.

TYPOGRAPHY

The orginal copy of this document was produced on a Xerox Graphics Printer with a resolution of two hundred points per inch. The principal font is News Gothic Boldface, 25 units high, which originated at Carnegie Mellon University. The page layout, text justification, boxes, halftones and line drawings were done using the author's document-formating program, XIP. The source files were prepared using the text editor, E, created by Dan Swinehart and Fred Wright.

SECTION 0.

INTRODUCTION.

"For the purpose of presenting my argument I must first explain the basic premise of sorcery as don Juan presented it to me. He said that for a sorcerer, the world of everyday life is not real, or out there, as we believe it is. For a sorcerer, reality or the world we all know, is only a description. For the sake of validating this premise don Juan concentrated the best of his efforts into leading me to a genuine conviction that what I held in mind as the world at hand was merely a description of the world; a description that had been pounded into me from the moment I was born."

- Carlos Castaneda. Journey to Ixtlan.

This thesis is about computer techniques for handling 3-D geometric descriptions of the world; the world that can be visually perceived with a television camera. The overall design idea may be characterized as an inverse computer graphics approach to computer vision. In computer graphics, the world is represented in sufficient detail so that the image forming process can be numerically simulated to generate synthetic television images; in the inverse, perceived television pictures (from a real TV camera) are analysed to compute detailed geometric models. For example, the polyhedra in Figure 0.1 on page two were computed from views of a plastic horse on a turntable. It is hoped, that visually acquired 3-D geometric models can be of use to other robotic processes such as manipulation, navigation or recognition.

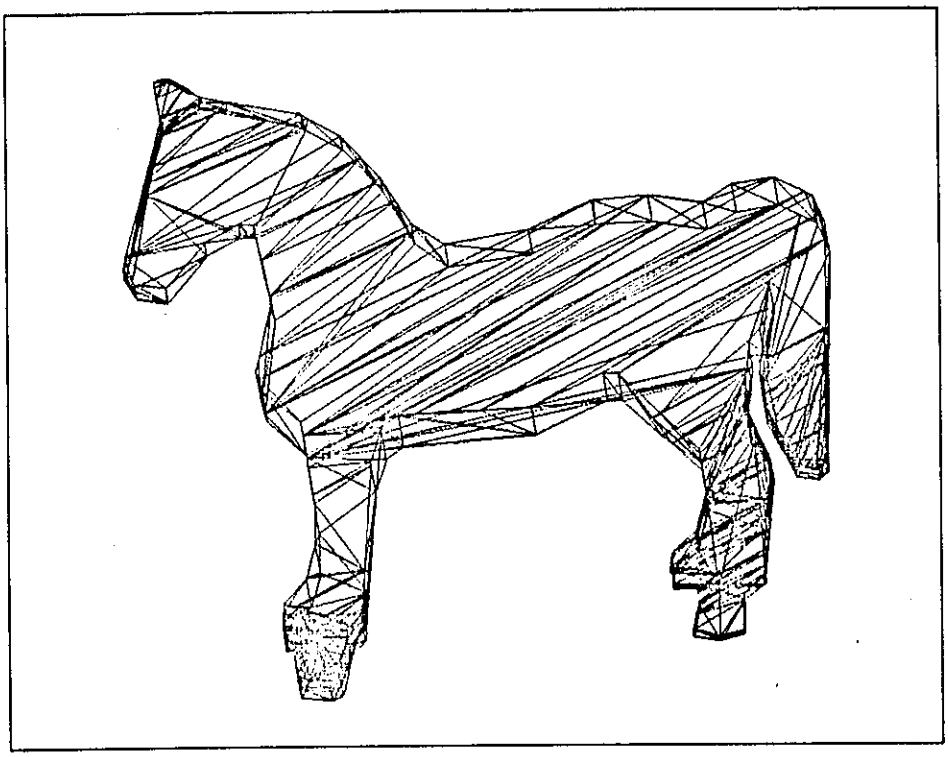
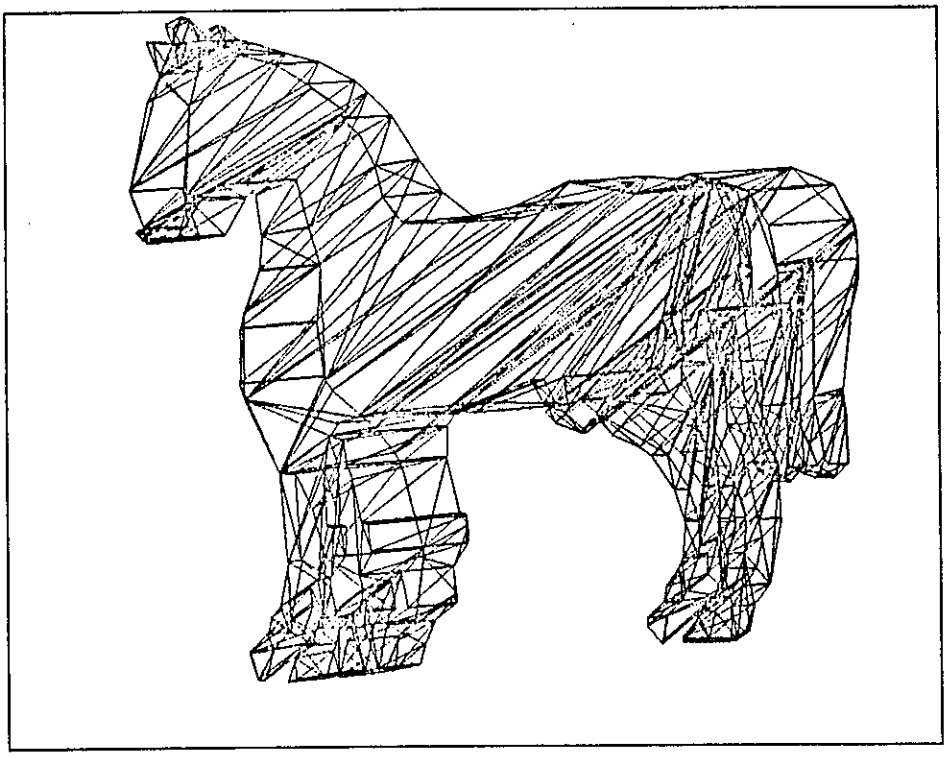
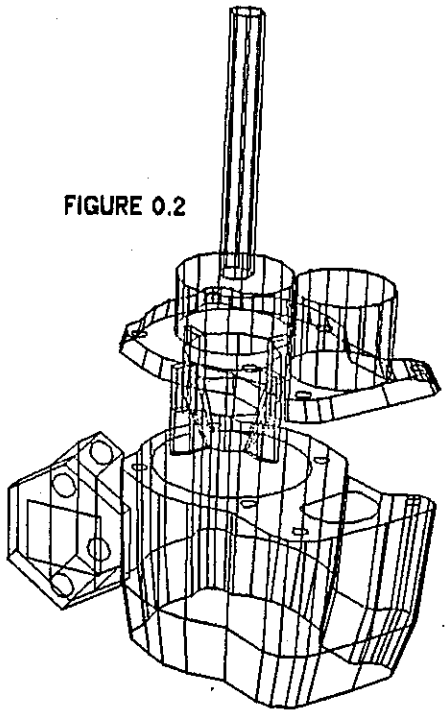


FIGURE 0.1 - HORSE SHAPED POLYHEDRA DERIVED FROM VIDEO IMAGES.



Once acquired, a 3-D model can be used to anticipate the appearance of an object in a scene, making feasible a quantitative form of visual feedback. For example, the appearance of the two machine parts depicted in Figure 0.2 can be computed and analyzed (top halves of Figures 0.3 and 0.4) and compared with an anaylsis of an actual video image of the parts (bottom halves of Figures 0.3 and 0.4). By comparing the predicted image with a perceived image, the correspondence between features of the internal model and features of the external reality can be established and a corrected location of the parts and the camera can be measured.

FIGURE 0.2



Finally by way of introduction, I wish to emphasive that the kind of vision being attempted is metric rather than linguistic and that the results achieved to date are modest. Feature classification and recognition in terms of English words is not being attempted, rather a system of prediction and correction between a 3-D world model and a sequence of images is contemplated. The chapters of this thesis proceed twice from theory through an implementation, with the first five chapters dealing with modeling and the last five chapters dealing with vision. Theory on geometric modeling is in Chapter 1 and theory on computer vision in Chapter 6. The implementation consists of two main programs named GEOMED and CRE. GEOMED is a system of 3-D modeling routines with which arbitrary polyhedra may be constructed, altered, or viewed in perspective with hidden lines eliminated; and CRE is a solution to the problem of finding intensity contours in a sequence of television pictures and of linking corresponding contours between pictures. Auxiliary programs perform top level task control, comparing and locus solving.

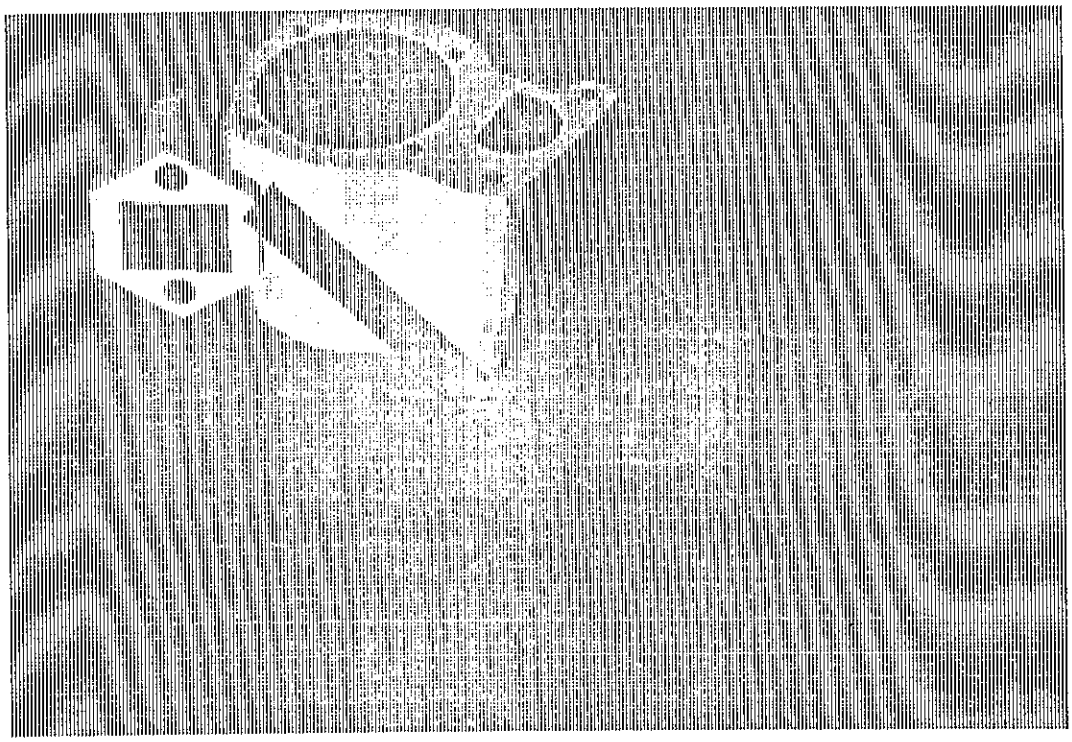


FIGURE 0.3 - PREDICTED VIDEO ↑ AND PERCEIVED VIDEO ↓.

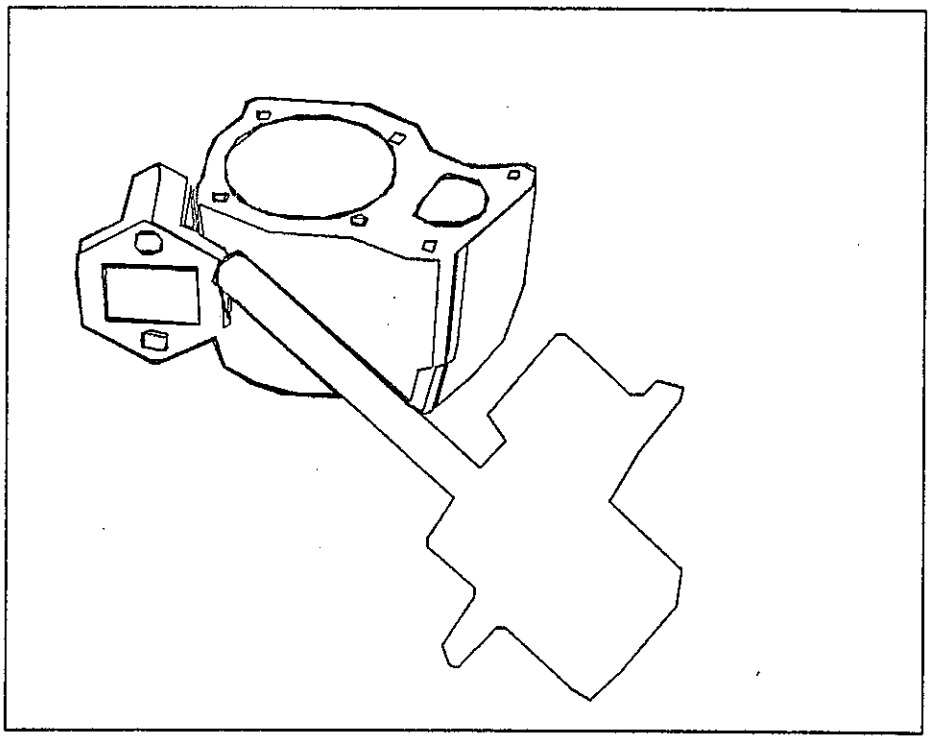
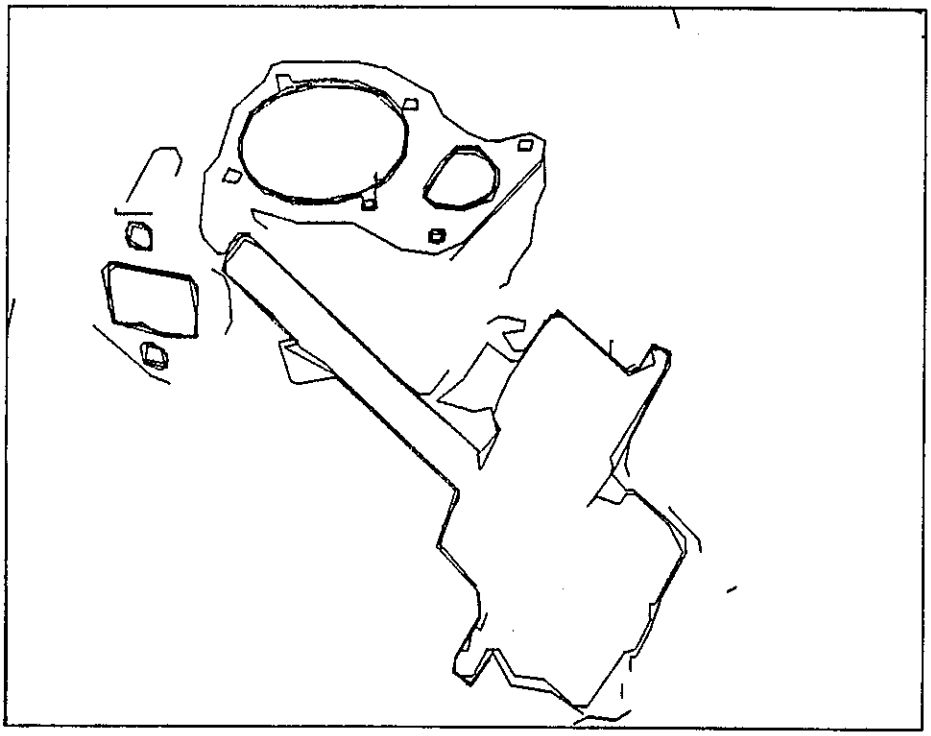


FIGURE 0.4 - PREDICTED IMAGE ↑ AND PERCEIVED IMAGE ↓.



SECTION 1.

GEOMETRIC MODELING THEORY.

1.0 Introduction to Geometric Modeling.

In the specific context of computer vision and graphics, *geometric modeling* refers to the construction of computer representations of physical objects, cameras, images and light for the sake of simulating their behavior. In Artificial Intelligence, a geometric model is a kind of world model; ignoring subtleties, geometric world modeling is distinguished from semantic and logical world modeling in that it is quantitative and numerical rather than qualitative and symbolic. The notion of a world model requires an external world environment to be modeled, an internal computer environment to hold the model, and a task-performing entity to use the model. In Geometry, modeling is a synthetic problem, like a construction with ruler and straight edge; modeling problems require an algorithmic solution rather than a proof. The word *geometric* is an appropriate adjective to this kind of modeling in that it is a combination of the Greek words γηο (world) and μετρια (measuring) which is exactly the activity to be automated.

1.1 Kinds of Geometric Models.

The main problem of geometric modeling is to invent methods for representing arbitrary physical objects in a computer. For the present discussion, the class of physical objects is restricted to objects that are solid, rigid, opaque, and macroscopic with a mathematically well behaved surface. Such objects include: the earth, chairs, roads, and plastic toy horses; other objects, for which models will not be attempted, include glass, fog, hair, Jello, liquids and cloth. Physical objects can move about in space with the restriction that two objects can not occupy the same space at the same time. The scope of the modeling problem can be appreciated by examining the models listed in Box 1.1.

BOX 1.1 TEN KINDS OF GEOMETRIC MODELS.

Space Oriented:	Object Oriented:
1. 3-D Space Array.	6. Manifolds.
2. Recursive Cells.	7. Polyhedra.
3. 3-D Density Function.	8. Volume Elements.
4. 2-D Surface Functions.	9. Cross Sections.
5. Parametric Surface Functions.	10. Skeletons.

For a naive start, first consider a 3-D array in which each element indicates the presence or absence of solid matter in a cube of space. Such a 3-D space array has the very desirable properties of *spatial addressing* and *spatial uniqueness* in their most direct and natural form. Spatial addressing refers to finding out what the model contains within a distance R of a locus X,Y,Z; spatial uniqueness refers to the property that physical solids can not occupy the same space simultaneously. A first drawback of the space array idea is illustrated by the apparently legal FORTRAN statement:

```
DIMENSION SPACE(100000,100000,100000)
```

The problem with such a dimension statement is that no present day computer memory is large enough to contain a 10^{15} element array. Smaller space arrays can be useful but necessarily can not model large volumes with high resolution. A further drawback of space arrays is that objects and surfaces are not readily accessible as entities; that is a space array lacks the property of *object coherence*. In computer graphics, the term *coherent* denotes both the quality of holding together as parts of the same mass and the quality of not changing too drastically from one point to the next. The meaning of *coherent* approachs the mathematical notion of topologically connected and locally continuous. The word is used to refer to the frame coherence of a film as well as to the object coherence of a model.

The space array idea can be salvaged by grouping blocks of elements with the same value together; the addressing process becomes more complicated but the overall memory required is reduced and the two desired properties can be maintained. One way of doing this (which has been discovered in several applications) is *recursive cells*; the whole space is considered to be a cell; if the space is not homogeneous then the first cell is divided into two (or four or eight) sub cells and the criterion is applied again. This technique allows the spatial sorting of objects when the object models can be subdivided at each recursion without losing their properties as objects.

Another salvageable naive modeling idea is that arbitrary objects can be expressed as algebraic functions. In physics, physical objects are frequently referred to as three dimensional density functions $W=\rho(X,Y,Z)$. Unfortunately such density functions can not be written out for objects such as a typing chair or a plastic horse without resorting to a programming language or an extensive table (which is equivalent to the space array model). Objects that are essentially 2-D can be approximated by a surface function $Z = F(X,Y)$. For example landscape may be represented by geodetic maps in such a 2-D fashion.

By definition, a function is single valued; consequently the description of even modestly complicated objects cannot be expressed by giving one coordinate, e.g. Z, as a function of the other two, e.g. X and Y. It is necessary either to adopt parametric functions or to subdivide the object into portions that can be described by simple functions of Cartesian variables. The former course involves establishing a system of surface coordinates (U,V), latitudes and longitudes, on the object in which functions for the X,Y,Z locus of the object's surface are expressed. The advantage of parametric functions is that extended arbitrary curve surfaces can be expressed; some of the disadvantages are that parametric curves may be self intersecting, they are not easy to modified locally, and the functions become impractical before the shapes of mundane artifacts can be achieved. Consequently parametric representations are combined with object subdivision, which is called *segmentation*. The process of usefully segmenting an object without destroying its coherence is a major problem requiring the combination of spatial, functional and objective representations.

In passing from space oriented models to object oriented models, I wish to note that sophisticated representation of time is beyond the scope of this work. Although an advanced problem solving robot will need to run world simulations along multiple time paths, the discussion will concentrate on representing the geometry of the world at a single moment in time.

After existence in space and time, another general property of physical objects is that they can be enclosed by an unbroken two dimensional surface with an unambiguous inside and outside; which touchs upon the mathematical topic (celebrated in song by Tom Lehrer) of the algebraic topology of locally Euclidean transitions of infinitely differentiable oriented Riemann manifolds. A *manifold* is the mathematical abstraction of a surface; a *Riemann* manifold has a metric function; an *oriented* manifold has a unambiguous inside and an outside; the phrase *infinitely differentiable* can be taken to mean that the surface is smooth; and the phrase *locally Euclidean transitions* refers to the process of segmenting the object into portions that can be approximated by relatively simple functions. In particular, the 2-D Riemann submanifold embedded in 3-D Euclidean space is the mathematical object that comes closest to representing the shape and extent of the surface of a physical object; such manifolds are conveniently approached through the topology of surfaces which in turn is computationally approached by means of polyhedra.

One way to describe the topology of a 2-D Riemann submanifold embedded in a 3-D Euclidean space is in terms of three kinds of simplex: the 0-Simplex (or vertex), the 1-Simplex (or edge), and the 2-Simplex (or triangle). In topological analysis 2-D Riemann submanifolds may be divided into faces, edges and vertices such that Euler's equation F-E+V=2-2*H is satisfied (where F is the number of faces, E is the number of edges, V is the number of vertices and H is the genus or number of handles of the manifold); and such that the surface of the manifold can be approximated by local functions over each face which are Euclidean and which fit together smoothly at all the edges. By introducing a sufficient (but finite) number of triangles the manifold can be approximated to within any epsilon by constant functions, yielding the geometric object called the *polyhedron*.

One advantage of a polyhedral model is its connected surface topology of faces, edges and vertices. Such a surface can be subdivided without losing its coherence or the coherence of the object.

The disadvantages of polyhedra include the lack of spatial uniqueness and spatial addressing which necessitates computation to be done to detect and prevent spatial conflict and to find the portions of an entity occupying a given volume. Another feature of polyhedra (which can be an advantage or disadvantage) is that all the (*Gaussian*) curvature happens suddenly at the vertices; however by associating higher order approximation functions with each face the model of a continuous 2-D manifold can be made which is a more conventional curved object representation. Nevertheless, polyhedra are intrinsically a general curved object representation.

Returning to the survey, arbitrary objects can also be described by listing a set of cross sections taken at a sufficient number of cutting planes; this is how the shape of a ship's hull or an airplane's wing is specified. Cross sections have the interesting feature of good space modeling on one axis. Forsaking arbitrary shaped objects, large classes of things can be described in terms of a small set of basic volume elements. For example, Roberts (63)* and others have built models of familiar objects using only rectangular and triangular right prisms. Arbitrary solid polyhedra can be constructed out of tetrahedra (the 3-simplex); however no significant general modeling system exists using this potentially interesting approach.

Skeletal models are based on abstracting an object into a stick figure and by associating a diameter or set of cross sections with the sticks. In particular, spine cross section models have been pursued at Stanford by Agin (72) and Nevatia (74). Spine cross section models have the advantage of being able to express many objects in a concise form suitable for recognition, but they cannot be used directly for arbitrary shapes.

Finally, it is often useful to represent physical objects by weak geometric models such as by sets of spheres or by sets of unconnected surface points. It is interesting to note that the *reality* that the robot in Winograd's thesis (Winograd 71) could talk about, was a blocks world based on a geometric model consisting only of points, size of block, and a two page LISP subroutine named FINDSPACE.

---

* Parenthesized names and numerals are references listed in Section 11.1

Beyond the particular kinds of geometric models, four general purpose modeling techniques deserve special mention and isolation: prototype instance structure, parts tree structure, resolution limited structure, and procedure generated structure. Superficially, the prototype instance structure is a memory efficiency technique based on storing generalizations (prototypes) which can be bound to specific cases (instances) as the occasion demands. Parts tree structure is a memory management technique of organizing the whole universe of discourse as a tree data structure, where objects are composed of subobjects. Resolution limited structure is a memory accessing technique, where depending on a specified scale of interest different models are retrieved or even generated. Finally, procedure generated structure concerns the trade-off between storing and recomputing a model; namely recomputing the details of a model as they are needed is a good idea for extending computational resources.

The danger to be avoided is to mistake the general modeling techniques for the geometric model itself. Given a modeling regime it can be improved by prototyping, parts-treeing, resolution-limiting and procedural-generating; without a good basic geometric model the general techniques amplify the background noise.

BOX 1.2 DESIRABLE PROPERTIES FOR A GEOMETRIC MODEL.

1. Spatial addressing.
2. Spatial uniqueness.
3. Object coherence.
4. Surface coherence.
5. Shape generality.
6. Large extent with high resolution.
7. Easy modifiablity.
8. Suitability for physical simulation.
9. Efficiency of memory and computation use,
10. Suitability for automatic model acquisition.

To the best of my knowledge, this survey is complete. As of this year, 1974, there are no other significantly different kinds of simple geometric models. The desirable properties that have turned up in this survey are listed in Box 1.2. The final desirable property is that there be some hope that the computer can derive the model by measurements it can make itself, although it is quite likely that one model will be best for input and another model will be best for simulation.

1.2 Polyhedron Definitions and Properties.

In computational modeling, definitions are not used formally, but are rather employed piecemeal in terms of individual properties which may or may not be present as polyhedra are generated and processed. In particular, the properties listed in Box 1.3 (given in order of relevance) can be taken as a working definition of a polyhedron for modeling a physical object.

BOX 1.3 PROPERTIES OF POLYHEDRA.

1. Eulerian ..... Satisfies the Euler equation: F-E+V=2-2*H.
2. Surface Homogeneity ..... The polyhedron does not intersect itself.
3. Trivalence ..... All vertices and faces have three or more edges.
4. Face Planarity ..... All vertices of a face are coplanar.
5. Solidity ..... The volume measure is nonzero, finite and positive.
6. Simply Connected Faces ..... Face perimeters have one loop of edges.
7. Face Convexity ..... All the faces are convex.
8. Edge Aplanarity ..... Faces which share an edge are not coplanar.

Topologically, the surface elements of a polyhedron form a graph that satisfies Euler's F-E+V=2-2*H equation; where as before F, E and V are the number of faces, edges and vertices of the polyhedron; and where H is the number of holes in (or genus of) the polyhedron. However, not all Eulerian graphs of faces, edges and vertices correspond to the usual notion of a solid polyhedron without the surface homogeneity and trivalence restrictions. Surface homogeneity is the property that for any point on the polyhedron a small enough sphere will cut from the surface a region homeomorphic to a disk; this restriction implies that the surface cannot intersect itself and that an edge can belong to only two different faces. The trivalence restriction insures that there are no degenerate two edged faces or one edged vertices; although a two edged vertex has a reasonable interpretation it is excluded by trivalence for the sake of face-vertex duality and canonical form. The last property, of aplanarity of faces with a common edge, is also for the sake of canonical form and is sacrificed to face convexity when necessary.

Geometrically, the faces of a polyhedron are planar, that is lie in a plane. It is also frequently relevant to further restrict the faces of a polyhedron to be convex, that is to require that every possible line segment between points of a face is contained within the face. To assure solidity, the volume measure must be restricted to be finite and positive; this restriction orients the surface to have

an exterior and an interior in the expected fashion. This restriction excludes non-orientable structures such as Mobius bands and Klein bottles for which the volume measure is undefined; however the restriction will be relaxed in Chapter 5 in order to exploit the concept of negative volumes.

The working definition was derived from more formal definitions such the following which defines a polyhedron as a special kind of a two dimensional manifold:

> "A polyhedron is a connected, unbounded two-dimensional manifold formed by a finite set of non-re-entrant, simply-connected plane polygons."
> - Coxeter, Regular Polytopes (Coxeter 1963).

In a *connected* manifold there exists a path between any two points that does not leave the manifold. An *unbounded* manifold is one with no cuts or gaps in its surface, that is no boundaries. A polyhedral manifold is composed of planar, simply-connected, non-re-entrant polygons; that is flat polygons with a perimeter of edges that form one loop that doesn't intersect itself. The polyhedron restrictions and properties are directed towards modeling physical objects and are maintained by computational mechanisms; consequently the word *polyhedron* comes to represent an intent, rather than the fulfillment of any particular set of defining properties.

1.3 Camera, Light and Image Modeling.

Common to both computer graphics and vision is the necessity to model cameras, light and images so that pictures may be synthesized or analyzed. The basic camera model has eight degrees of freedom, three in location, three in orientation and two in projection:

Location:	CX, CY, CZ	Vector to camera lens center.
Orientation:	WX, WY, WZ	Orientation vector.
Projection:	AR, FR	Aspect Ratio and Focal Ratio.

The orientation vector is explained in Section 3.3, the perspective projection is defined in Section 3.4, and the derivation of the camera parameters is the main topic of Chapter 9. In modeling light and physical objects, the most important and difficult property to simulate is opacity. Techniques for modeling opaque objects are presented in Chapter 4.

Finally, an image is a 2-D geometric object representing the content of a rectangle from the pattern of light of light formed by a thin lens on a television vidicon. The video image is the interface to the external reality. Image modeling is analogous to 3-D geometric modeling, since the same tradeoffs between spatial structure and object structure arise. A 2-D image may be represented as a video raster, which is a 2-D space array; or as a set of feature loci, which is an object oriented description. Image structures and processors for generating and comparing image representations are discussed in Chapters 7 and 8. Together camera, light and image modeling are the essential elements required to apply a geometric model to computer vision.

1.4 Related Modeling Work.

Although geometric modeling per se has a long history and a rich literature in mathematics, physics and engineering, very little such modeling has been done using a computer at the level of detail required for visual perception. This level falls between the generality typical in physics and mathematics and the specificity typical of engineering. Computer science research in geometric modeling has already been cited in Section 1.2; similar ideas are available from computer graphics sources (Newman and Sproull 73). In computer graphics, the typical modeling paper invariably has a long discussion about the implementation of a node/link modeling language (CORAL, LEAP, ASP, and others) and very little discussion on how the actual geometric modeling is to be done in the given language. In mathematics, I have found the work of the Canadian geometer Coxeter, (Coxeter 61) and (Coxeter 63) to be my best source of ideas relevant to modeling; along with the observations from recreational mathematicians (Gardner 59), (Gardner 61) and (Stewart 70); and geometry textbook authors (Eves 65), (Snyder 14) and (Graustein 35). The translation of Hilbert's book (Hilbert 52) presenting Geometry for the non-mathematician is also a good source of ideas. From Physics, material on classical mechanics is useful in modeling rotation and inertia tensors (Goldstein 50), (Feynman et al 63) and (Symon 53). In engineering, books on geodetic surveying, mechanical drawing and architectural drawing contain ideas relevant to modeling particular classes of objects; I have selected (Luzadder 71) and (Muller 67) almost at random, as introductions to engineering and architectural drawing, respectively.

SECTION 2.

THE WINGED EDGE POLYHEDRON REPRESENTATION.

2.0 Introduction to the Winged Edge.

In this chapter, a particular computer representation for polyhedra is presented and some of its virtues and faults are explained. The representation is implemented as a data structure composed of small blocks of words containing pointers and data in the fashion usual to graphics and simulation. An introduction to such data structures can be found in Chapter 2 of Knuth's Art of Computer Programming (Knuth 68). Quickly reviewing Knuth's terminology, a node is a group of consecutive words of memory, a field is a named portion of a node and a link is the machine address of a node. The notation for referring to a field of a node consists simply of the field name followed by a link expression enclosed in parentheses. For example, the two faces of an edge node whose link is stored in the variable named "edge", are found in the fields named NFACE and PFACE, and are referred to as NFACE(edge) and PFACE(edge). Although my latest language of implementation is PDP-10 machine code, examples in this chapter will be given in a fictional programming language which combines ALGOL with Knuthian node/link notation. (As an exercise, the energetic reader should write out a possible representation for general polyhedra, before reading any further.)

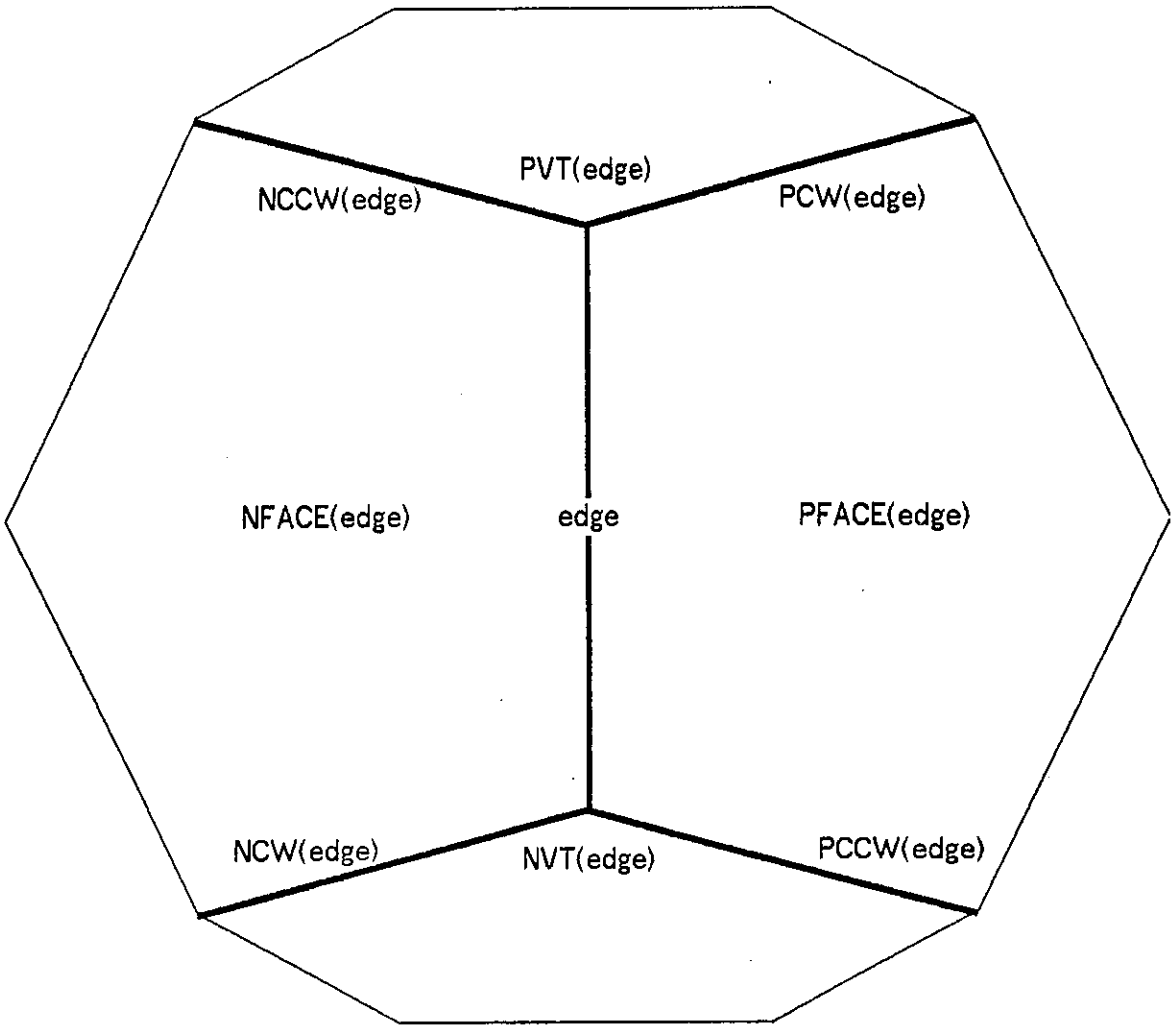


FIGURE 2.1 - Winged Edge Topology.

The orientation of links is as viewed from the exterior side of the surface.
The eight mnemonics in the figure, were derived as follows:

NFACE(edge) Negative Face of edge.
PFACE(edge) Positive Face of edge.
PVT(edge) Positive Vertex of edge.
NVT(edge) Negative Vertex of edge.
NCW(edge) edge in Negative face Clockwise from edge.
PCW(edge) edge in Positive face Clockwise from edge.
NCCW(edge) edge in Negative face Counter Clockwise from edge.
PCCW(edge) edge in Positive face Counter Clockwise from edge.

2.1 Winged Edge Link Fields.

A polyhedron in made up of four kinds of nodes: bodies, faces, edges and vertices. The body node is the head of three rings: a ring of faces, a ring of edges and a ring of vertices. In this context, a ring is a doubly linked circular list with a head node. Each face and each vertex points directly at only one of the edges on its perimeter. Each edge points at its two faces and its two vertices. Completing the topology, each edge node contains a link to each of its four immediate neighboring edges clockwise and counter clockwise about its face perimeters as seen from the exterior side of the surface of the polyhedron. These last four links are the wings of the edge, which provide the basis for efficient face perimeter and vertex perimeter accessing. Finally, the links of the edge nodes can be consistently oriented with respect to the surface of the polyhedron so that the surface always has two sides: the inside and the outside.

BOX 2.1 WINGED EDGE STRUCTURES AND LINK NAMES.

Data Structures	Link Names	
1. Face Ring of a Body.	NFACE	PFACE
2. Edge Ring of a Body.	NED	PED
3. Vertex Ring of a Body.	NVT	PVT
4. First Edge of a Vertex.		PED
5. First Edge of a Face.		PED
6. The two faces of an edge:	NFACE	PFACE
7. The two vertices of an edge:	NVT	PVT
8. The four wing edges of an edge:	NCW PCW	NCCW PCCW

Observe that there are twenty-two link fields in the basic representation: bodies contain six links, faces three links, vertices three links and edges ten links. If we allow a link name such as PED to serve different roles depending on whether it applies to a body, face, edge or vertex; then the minimum number of different link field names that need to be coined is ten. The data structures and the link fields comprising the structures are listed in Box 2.1. The ten link names include: NFACE and PFACE for two fields that contain face links in edges and the face ring, NED and PED for two fields that contain edge links, NVT and PVT for two fields that contain vertex links, and NCW, PCW, NCCW and PCCW for the four fields that contain edge links and are called the wings.

By constraining the arrangement of links in an edge node both the surface orientation (interior and exterior) and a linear orientation of the edge as a directed vector can be encoded. Figure 2.1 diagrams the arrangement of the links comprising the topology of an edge of a polyhedron as viewed from the exterior side of its surface. Although the vertices in Figure 2.1 are shown with only three edges, vertices may have any number of edges; the other potential edges would not be directly linked to the middle edge of the figure and so were not shown.

To complete the representation, space is allocated to contain the 3-D coordinates of each vertex in fields named XWC, YWC and ZWC; the initials "WC" stand for *World Coordinates*. For the sake of vision and display, three more words are allocated to hold the *Perspective Projected coordinates* of each vertex in fields named XPP, YPP and ZPP. Also a word of thirty six status bits is carried in every node: permanent status bits specify the type (body, face, edge, vertex, etc.) of every node, temporary bits provide space for operations such as hidden line elimination that require marking. Passing now from necessities to conveniences, faces carry exterior pointing normal vectors and several words of photometric surface characteristics. The face vectors are derived from surface topology and vertex loci, and so they are not basic geometric data as in some representations. Bodies carry a print name, as well as four link fields (DAD, SON, BRO, SIS) for implementing a parts tree data structure; and two link fields (CW and CCW) for a body ring of all the bodies in the world model. Node formats are given in Section 11.2 for an implementation based on fixed sized (twelve word) nodes.

The Winged Edge Polyhedron Representation as just presented is complete. Edge nodes carry most of the topology, vertex nodes carry the geometry, face nodes carry the photometry and body nodes carry the linguistics (nomenclature) and parts tree structure. The point that remains to be demonstrated, is that the appropriate subroutines for creating, maintaining and exploiting edge orientation execute efficiently and provide good primitives for solving such geometric problems as hidden line elimination and polyhedral intersection.

2.2 Sequential Accessing.

An immediate consequence of the ring structures is that the faces, edges and vertices of a body are sequentially accessible in the manner illustrated by the following lines of code:

```
COMMENT APPLY A FUNCTION TO ALL THE FACES, EDGES AND VERTICES OF A BODY;
PROCEDURE APPLY (PROCEDURE FN; INTEGER B);
BEGIN
        INTEGER F,E,V;
        F ← B; WHILE B≠(F←PFACE(F)) DO FN(F);   COMMENT APPLY FUNCTION TO FACES OF A BODY;
        E ← B; WHILE B≠(E←PED(E)) DO FN(E);     COMMENT APPLY FUNCTION TO EDGES OF A BODY;
        V ← B; WHILE B≠(V←PVT(V)) DO FN(V);     COMMENT APPLY FUNCTION TO VERTICES OF A BODY;
END;
```

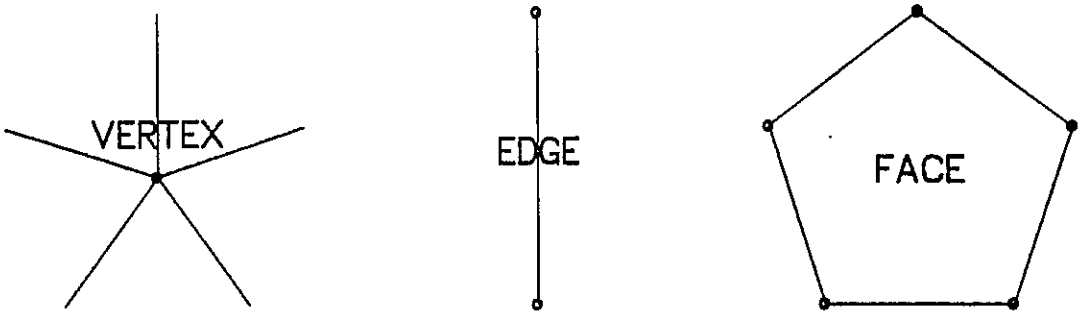
The rings could of course have been traversed in the other direction by invoking NVT, NED and NFACE in place of PVT, PED and PFACE. The reason for doubly linked lists (i.e. rings) is rapid deletion. Finally, observe that the face and vertex rings could be eliminated at the cost of having a more complicated face/vertex sequential accessing method requiring a visitation marking bit in the status word of face and vertex nodes. The idea might be coded as follows:

```
COMMENT APPLY A FUNCTION TO ALL THE FACES OF A BODY WITHOUT USING THE FACE RINGS;
PROCEDURE APPLY (PROCEDURE FN; INTEGER B);
BEGIN
        INTEGER F,E,M;
        E ← B;                                  COMMENT FIRST EDGE OF BODY;
        M ← MARK(PFACE(E));                     COMMENT READ INITIAL STATE OF MARKING BIT;
        DO FOR F ← PFACE(E),NFACE(E) DO         COMMENT FOR BOTH FACES OF EACH EDGE...;
        BEGIN
                IF M=MARK(F) THEN FN(F);        COMMENT APPLY FUNCTION TO "UN-RE-MARKED" FACE;
                MARK(F) ← ¬M;                   COMMENT FLIP THE MARKING BIT;
        END;
        UNTIL B=(E←PED(E));                     COMMENT ALL THE EDGES OF THE BODY;
END;
```

2.3 Perimeter Accessing.

The perimeter of a face is an ordered list of edges and vertices, the perimeter of a vertex is an ordered list of edges and faces, and the perimeter of an edge is an ordered list consisting of exactly two faces and two vertices. The perimeter definitions are caricatured in Figure 2.2. One virtue of the winged edge representation is that both vertex and face perimeters can be traversed in either direction (clockwise or counter clockwise) while being dynamically maintained in "*one ring*".

FIGURE 2.2 - Three Kinds of Perimeters.



A Vertex is surrounded by Edges and Faces

An Edge is surrounded by Faces and Vertices

A Face is surrounded by Edges and Vertices

Given one edge of a face (or vertex) perimeter, the next edge clockwise (or counter clockwise) from the given edge about the particular face (or vertex) can be retrieved from the data structure with the assistance of two subroutines called ECW and ECCW. The idea of the edge clocking routines is to match the given face (or vertex) with one of the faces (or vertices) of the given edge and to then return the appropriate wing. A possible coding of ECCW and ECW might be as follows:

```
COMMENT FETCH EDGE CCW FROM E ABOUT FV;
INTEGER PROCEDURE ECCW (INTEGER E,FV);
BEGIN "ECCW"
        IF PFACE(E)=FV THEN RETURN(PCCW(E));
        IF NFACE(E)=FV THEN RETURN(NCCW(E));
        IF PVT(E)=FV   THEN RETURN(PCW(E));
        IF NVT(E)=FV   THEN RETURN(NCW(E));
        FATAL;
END "ECCW";
```

```
COMMENT FETCH EDGE CLOCKWISE FROM E ABOUT FV;
INTEGER PROCEDURE ECW (INTEGER E,FV);
BEGIN "ECW"
        IF PFACE(E)=FV THEN RETURN(PCW(E));
        IF NFACE(E)=FV THEN RETURN(NCW(E));
        IF PVT(E)=FV   THEN RETURN(NCCW(E));
        IF NVT(E)=FV   THEN RETURN(PCCW(E));
        FATAL;
END "ECW";
```

The first edge of a face or vertex is (of course) immediately available from the PED field of the face or vertex. For example, the two procedures below can be used to visit all the edges of a face or all the edges of a vertex, respectively.

```
COMMENT APPLY FUNCTION TO EDGES OF A FACE;
PROCEDURE APPLY (PROCEDURE FN; INTEGER F);
BEGIN
        INTEGER E,E0;
        E←E0←PED(F);
        DO FN(E) UNTIL E0=(E←ECCW(E,F));
END;
```

```
COMMENT APPLY FUNCTION TO EDGES OF A VERTEX;
PROCEDURE APPLY (PROCEDURE FN; INTEGER V);
BEGIN
        INTEGER E,E0;
        E←E0←PED(V);
        DO FN(E) UNTIL E0=(E←ECCW(E,V));
END;
```

Using the same idea as in the edge clocking routines, a face or vertex can be retrieved relative to a given edge and a given face or vertex. These routines include: FCW and FCCW which return the

face clockwise or counter clockwise from a given edge with respect to a given vertex; VCW and VCCW which return the vertex clockwise or counter clockwise from a given edge with respect to a given face; and OTHER which returns the face or vertex of the given edge opposite the given face or vertex. Together the seven routines: ECW, ECCW, VCW, VCCW, FCW, FCCW and OTHER exhaust the possible oriented retrievals from an edge node; they also alleviate the need to ever explicitly reference a wing field when traveling the surface of a polyhedron. With node type checking the primitives can be made stronger, for example ECCW(vertex,face) is implemented to return the edge counter clockwise from the given vertex about the given face. With node type checking and signed arguments the seven perimeter accessing routines could even be replaced by a single routine perhaps named PERIMETER_FETCH or PGET. On the other hand, I favor having the proliferation of accessing names for the sake of documenting the clocking direction and the types of nodes involved.

Two remaining surface accessing routines, of minor importance, are BGET(entity) and LINKED(entity,entity). BGET of a face, edge or vertex merely cycles the appropriate ring to retrieve the body of the given entity. The LINKED routine determines whether its two arguments (faces, edges or vertices) are adjacent; there are six LINKED cases: (i) Face-Face, returns a common edge or FALSE; (ii) Face-Edge, returns boolean value F=PFACE(E) v F=NFACE(E); (iii) Edge-Edge, returns a common vertex or false; (v) Edge-Vertex, returns boolean value V=PVT(E) v V=NVT(E); (vi) Vertex-Vertex, returns common edge or FALSE. (As in LISP, zero is false and non-zero is true).

2.4 Basic Polyhedron Synthesis.

BOX 2.2 LOWEST LEVEL WINGED EDGE ROUTINES.

Node Makers:	MKNODE, MKB, MKF, MKE, MKV, MKTRAM.
Node Killers:	KLNODE, KLB, KLF, KLE, KLV.
Wing Mungers:	WING, INVERT, EVERT.
Surface Fetchers:	ECW, ECCW, OTHER, VCW, VCCW, FCW, FCCW, LINKED.
Parts Tree Routines:	BDET, BATT, BGET.

There are sixteen routines for node creation and link manipulation which when combined with the nine accessing routines of the previous section form the nucleus of a polyhedron modeling system. These routines are very low level in that the final applications user of winged polyhedra will never

explicitly need to make a node or mung a link. The word *mung* (meaning to modify an existing structure by altering links in place) is LISP slang that deserves to be promoted into the technical jargon; traditionally, a mung routine is one which makes applications of the LISP primitives RPLACA and RPLACD. The twenty five routines listed in Box 2.2 are the bedrock foundation for the Euler primitives presented in Chapter 3.

Node Makers and Killers. The MKNODE and KLNODE are the raw storage allocation routines which fetch or return a node from the available free storage. The MKB routine creates a body node with empty face, edge and vertex rings; the body is placed into the body ring of the world model. The MKF, MKE and MKV each take one argument and create a new face, edge or vertex node in the ring of the given entity; with type checking these three primitives could be consolidated. Finally the MKTRAM node creates a *tram node*, which consists of twelve real numbers that represent either a Euclidean transformation or a Cartesian frame of reference depending on the context. (Tram nodes are explained in Section 3.3.) The corresponding kill routines KLB, KLF, KLE and KLV remove the entity from its respective ring and return its node to free storage.

Wing Mungers. The WING(edge1,edge2) routine finds which face and vertex the arguments edge1 and edge2 have in common and stores the wing pointers between edge1 and edge2 accordingly; the exact link manipulations are illustrated in the example coding of the WING procedure immediately following this paragraph. Recalling that edges are directed vectors, the INVERT(E) routine flips the direction of an edge by swapping the contents of the appropriate fields as follows: PFACE(E)↔NFACE(E); PVT(E)↔NVT(E); NCW(E)↔NCCW(E) and PCW(E)↔PCCW(E). Finally, the EVERT(B) routine turns a body inside out, by performing the following link swaps on all the edges of the given body: PFACE(E)↔NFACE(E); NCW(E)↔PCCW(E); and NCCW(E)↔PCW(E).

```
PROCEDURE WING(INTEGER  E1,E2);
BEGIN
        IF PVT(E1)=PVT(E2)∧PFACE(E1)=NFACE(E2)THEN BEGIN  PCW(E1)←E2;NCCW(E2)←E1;END;
        IF PVT(E1)=PVT(E2)∧NFACE(E1)=PFACE(E2)THEN BEGIN NCCW(E1)←E2; PCW(E2)←E1;END;
        IF PVT(E1)=NVT(E2)∧PFACE(E1)=PFACE(E2)THEN BEGIN  PCW(E1)←E2;PCCW(E2)←E1;END;
        IF PVT(E1)=NVT(E2)∧NFACE(E1)=NFACE(E2)THEN BEGIN NCCW(E1)←E2; NCW(E2)←E1;END;
        IF NVT(E1)=PVT(E2)∧PFACE(E1)=PFACE(E2)THEN BEGIN PCCW(E1)←E2; PCW(E2)←E1;END;
        IF NVT(E1)=PVT(E2)∧NFACE(E1)=NFACE(E2)THEN BEGIN  NCW(E1)←E2;NCCW(E2)←E1;END;
        IF NVT(E1)=NVT(E2)∧PFACE(E1)=NFACE(E2)THEN BEGIN PCCW(E1)←E2; NCW(E2)←E1;END;
        IF NVT(E1)=NVT(E2)∧NFACE(E1)=PFACE(E2)THEN BEGIN  NCW(E1)←E2;PCCW(E2)←E1;END;
END;
```

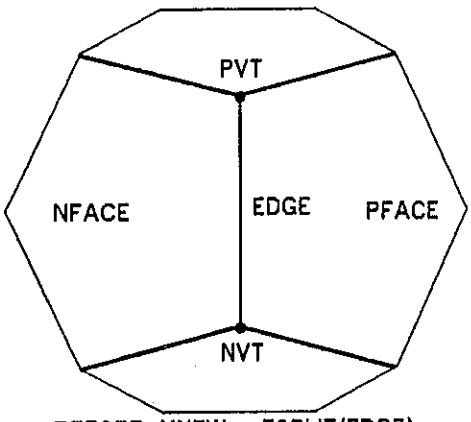
Part Tree Routines. As mentioned before, body nodes can be grouped into a tree structure or parts. The parts tree consumes four link positions (DAD, SON, BRO, SIS) and is maintained in body nodes by the following primitives: BDET(body) detachs a body node from the parts tree, BATT(body1,body2) attachs body1 to the ring of children belonging to body2, and BGET(entity) returns the body node at the head of the given face, edge or vertex ring. The SON field of a body may contain a pointer to a headless ring of subpart bodies, the ring of subparts is maintained in the BRO (brother) and SIS (sister) fields, and each subpart contains a pointer back to its parent in its DAD field. At present, the notion of a body is coincident with the notion of a connected polyhedron; however by allowing several bodies to be associated with a single polyhedral surface, a flexible object such as an animal could be represented.

2.4 Edge and Face Splitting.

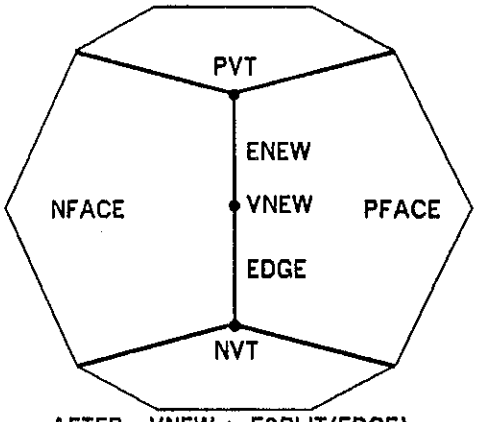
One of the most important properties of the winged edge representation is that edges and faces can be split using subroutines that make only local alterations to the data structure; and the splits can easily be removed (since the doubly linked rings allow rapid deletion of nodes from a body). The edge split routine, ESPLIT, makes a new edge and a new vertex and places them into the surface topology as shown in Figure 2.3; the kill edge-vertex routine, KLEV, undoes an ESPLIT. The face split routine, MKFE, creates a new edge and a new face and places them into the surface topology as shown in Figure 2.4; the kill face-edge routine, KLFE, undoes a MKFE.

The rest of this section concerns implementation; it may be skipped by the applications oriented reader. The split and kill routines are examples of a pattern which applies to the coding of operators that alter winged edge structures. In a typical situation, there are five steps: first, get the proper kinds of nodes into the body rings using the MKF, MKE, MKV primitives; second, position the vertices by setting their XWC, YWC, ZWC fields; third, connect each vertex and face to one of its edges by setting face/vertex PED fields; fourth, connect each edge to its two faces and its two vertices by setting the NFACE, PFACE, NVT, PVT fields of the edge; finally, set up the wing perimeter pointers by applying the WING primitive to the pairs of edges to be mated.

FIGURE 2.3 - ESPLIT AND KLEV.



BEFORE: VNEW ← ESPLIT(EDGE);
AFTER: EDGE ← KLEV(VNEW);



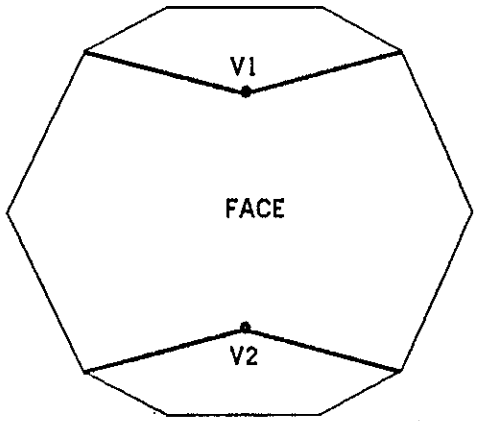
AFTER: VNEW ← ESPLIT(EDGE);
BEFORE: EDGE ← KLEV(VNEW);

```
INTEGER PROCEDURE ESPLIT (INTEGER EDGE);
BEGIN "ESPLIT"
        INTEGER VNEW,ENEW;
COMMENT CREATE A NEW EDGE AND VERTEX;
        VNEW ← MKV(PVT(EDGE));
        ENEW ← MKE(EDGE);
COMMENT CONNECT VERTICES & FACES TO EDGES;
        PVT(ENEW) ← PVT(EDGE);
        NVT(ENEW) ← VNEW;
        PVT(EDGE) ← VNEW;
        PFACE(ENEW) ← PFACE(EDGE);
        NFACE(ENEW) ← NFACE(EDGE);
COMMENT CONNECT EDGES TO VERTICES;
        IF PED(PVT(EDGE))=EDGE THEN
          PED(PVT(EDGE))←ENEW;
        PED(VNEW)←ENEW;
COMMENT LINK THE WINGS TOGETHER;
        NCW(ENEW) ← EDGE; PCCW(ENEW) ← EDGE;
        PCW(EDGE) ← ENEW; PCCW(EDGE) ← ENEW;
        WING(NCCW(EDGE),ENEW);
        WING(PCW(EDGE),ENEW);
        RETURN(VNEW);
END "ESPLIT";
```

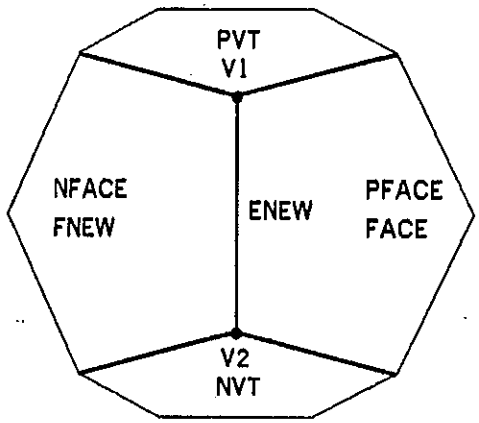
```
INTEGER PROCEDURE KLEV (INTEGER VNEW);
BEGIN "KLEV"
        INTEGER EDGE,ENEW,V,F,B;
        ENEW ← PED(VNEW);
        EDGE ← ECCW(ENEW,VNEW);
COMMENT ORIENT EDGES AS IN DIAGRAM;
        IF NVT(ENEW) ≠ VNEW THEN INVERT(ENEW);
        IF PVT(EDGE) ≠ VNEW THEN INVERT(EDGE);
COMMENT TIE E TO ITS NEW UPPER VERTEX AND WINGS;
        V ← PVT(EDGE) ← PVT(ENEW);
        WING(PCW(ENEW),EDGE);
        WING(NCCW(ENEW),EDGE);
COMMENT ELIMINATE OCCURRENCES OF ENEW IN F AND V;
        IF PED(V)=ENEW THEN PED(V) ← EDGE
        IF PED(PFACE(EDGE))=ENEW THEN
          PED(PFACE(EDGE))←EDGE;
        IF PED(NFACE(EDGE))=ENEW THEN
          PED(NFACE(EDGE))←EDGE;
COMMENT REMOVE NODES FROM RINGS AND RETURN EDGE;
        KLV(VNEW);
        KLE(ENEW);
        RETURN(EDGE);
END "KLEV";
```

The actual routines differ slightly from those given above in that they do argument type checking and data structure checking; nevertheless, a diagnostic trace of the implemented version reveals that the ESPLIT routine executes an average of 170 PDP-10 instructions and the KLEV routine executes an average of 200 instructions.

FIGURE 2.4 - MKFE AND KLFE.



BEFORE: ENEW ← MKFE(V1,FACE,V2);
AFTER: FACE ← KLFE(ENEW);



AFTER: ENEW ← MKFE(V1,FACE,V2);
BEFORE: FACE ← KLFE(ENEW);

```
INTEGER PROCEDURE MKFE(INTEGER V1,FACE,V2);
BEGIN "MKFE"
        INTEGER V1,V2,FNEW,ENEW,E,E0,B,V;
COMMENT CREATE NEW FACE & EDGE;
        FNEW ← MKF(FACE); ENEW ← MKE(PED(FACE));
COMMENT LINK NEW EDGES TO ITS FACES & VERTICES;
        PED(F) ← PED(FNEW) ← ENEW;
        PFACE(ENEW) ← F; NFACE(ENEW) ← FNEW;
        PVT(ENEW) ← V1; NVT(ENEW) ← V2;
COMMENT GET THE WINGS OF THE NEW EDGE;
        E2 ← PED(V1);
        DO E2←ECW((E1←E2),V1) UNTIL FCW(E1,V1)=FACE;
        E4 ← PED(V1);
        DO E4←ECW((E3←E4),V2) UNTIL FCW(E3,V2)=FACE;
COMMENT SCAN CCW FROM V1 REPLACING F'S WITH FNEW;
        E ← E2;
        DO IF PFACE(E)=FACE THEN PFACE(E)←FNEW
         ELSE NFACE(E)←FNEW;
        UNTIL E4 = (E←ECCW(E,FNEW));
COMMENT LINK THE WINGS;
        WING(E1,ENEW); WING(E2,ENEW);
        WING(E3,ENEW); WING(E4,ENEW);
        RETURN(ENEW);
END;
```

```
INTEGER PROCEDURE KLFE (INTEGER ENEW);
BEGIN "KLFE"
        INTEGER FNEW,FACE,V1,V2,E,E1,E2,E3,E4;
COMMENT PICKUP ALL THE LINKS OF ENEW;
        FACE ← PFACE(ENEW); FNEW ← NFACE(ENEW);
        V1 ← PVT(ENEW);  V2 ← NVT(ENEW);
        E1 ← PCW(ENEW); E2 ← NCCW(ENEW);
        E3 ← NCW(ENEW); E4 ← PCCW(ENEW);
COMMENT GET ENEW LINKS OUT OF FACE, V1 AND V2;
        IF PED(V1) = ENEW THEN PED(V1) ← E1;
        IF PED(V2) = ENEW THEN PED(V2) ← E3;
        IF PED(FACE)=ENEW THEN PED(FACE)←E3;
COMMENT GET RID OF FNEW APPEARANCES;
        E ← E2;
        DO IF PFACE(E)=FNEW THEN PFACE(E)←FACE
         ELSE NFACE(E)←FACE;
        UNTIL E4 = (E←ECCW(E,FNEW));
COMMENT LINK WINGS TOGETHER ABOUT FACE;
        WING(E2,E1);WING(E4,E3);
        KLF(FNEW);KLE(ENEW);
        RETURN(FACE);
END;
```

Again, the actual routines differ from those given above in that they do argument type checking and data structure checking. The above two routines typically take about twice as long to execute as the previous pair; notice that the execution time is dependent on the length of face perimeters, which are mostly three or four edges long.

2.5 Coordinate Free Polyhedron Representation.

As in general relativity, all geometric entities can be represented in a coordinate free form. In particular, the vertex coordinates of a polyhedron can be recovered from edge lengths and dihedral angles (the angle formed by the two faces at each edge). Having the geometry carried by only two numbers per edge rather than by three numbers per vertex does not necessarily yield a more concise representation because edges always outnumber vertices two for one, and in the case of a triangulated polyhedron edges outnumber vertices by three to one.

One application of a coordinate free representation arises when it is necessary to measure a shape with simple tools such as a caliper and straight edge. For example, one way to go about recording the topology and geometry of an arbitrary object is to draw a triangulated polyhedron on its surface with serial numbered vertices and to record for each edge its length, its two vertices and its *signed dihedral length*. The dihedral length is the distance between the vertices opposite the edge in each of the edge's two triangles; the length can be given a sign convention to indicate whether the edge is concave or convex. The required dihedral angles can then be computed from the signed dihedral lengths.

SECTION 3.

A GEOMETRIC MODELING SYSTEM.

3.0 Introduction to GEOMED.

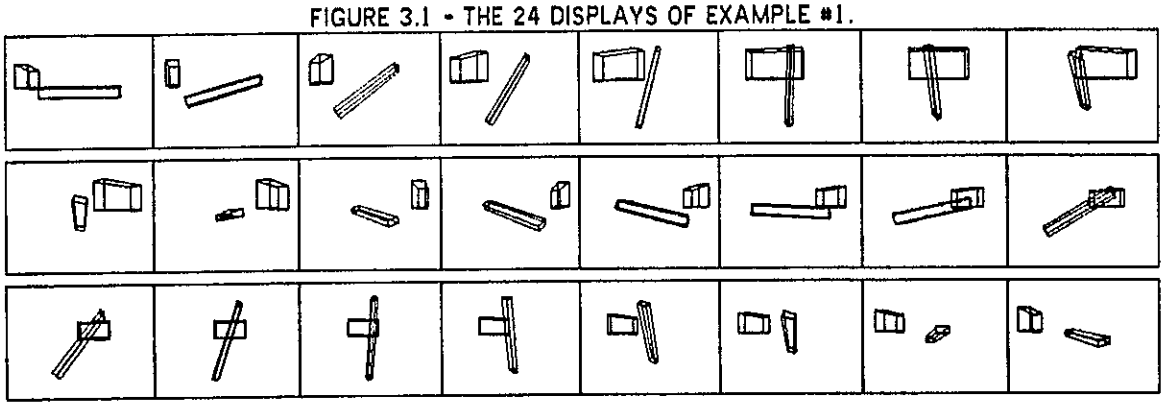
GEOMED (Geometric Editor) is a system of subroutines for manipulating winged edge polyhedra. The system has two manifestations: first, it appears as an interactive 3-D drawing program and second, it appears as a geometric modeling command language. It is the latter manifestation along with some of the details of implementation that is the subject of this chapter; the interactive drawing program is documented in (Baumgart 74). As a language, GEOMED is all semantics with no particular syntax of its own; there are about two hundred subroutines which take from zero to four arguments, return one or no values and which usually have considerable side effects on the data structures. The subroutines can be grouped into five classes: utility routines, Euler routines, Euclidean routines, image synthesis and image analysis routines. The utility routines include input/output, trigonometric functions, memory management, a command scanner, and device dependent display routines; the utility routines will not be further elaborated. The Euler routines perform topological operations on links, the Euclidean routines perform geometric computations on data, and the image synthesis routines perform photographic simulations on the model as a whole. The fifth class, image analysis routines, consists at present solely

of an interface between GEOMED and CRE, the fifth group lacks the completeness of the other parts of the system.

As in the previous chapter, the programming notation used will continue to have an ALGOL appearance with specific examples of actual GEOMED code being given in the language SAIL (Stanford ALGOL) as is example #1 immediately below. The program in example #1 creates two cubic prisms and

```
BEGIN "EXAMPLE ONE"
        REQUIRE "GEOMES.HDR[GEM,HE]" SOURCE_FILE;      COMMENT DECLARE GEOMED EMBEDDED IN SAIL;
        DEFINE PI="3.1415927";
        INTEGER B1,B2,I;                               COMMENT TWO BODIES AND AN IMAGE COUNTER;
        MKUNIV;                                        COMMENT INITIALIZE THE DATA STRUCTURES;
        B1 ← MKCUBE(8,1,0.5);                          COMMENT CREATE A COUPLE OF CUBIC PRISMS;
        B2 ← MKCUBE(1,2,4);
        TRANSL(B2,-7,1.5,0);                           COMMENT DISPLACE ONE OF THEM;
        FOR I←1 STEP 1 THRU 24 DO                      COMMENT MAKE 24 IMAGES;
BEGIN
        GEODPY;                                        COMMENT DISPLAY REFRESH;
        PLOTO("TMP."&CVS(I));                          COMMENT OUTPUT LATEST DISPLAY TO DISK;
        ROTATE(B1,PI/18,PI/12,PI/13);                  COMMENT ACTION WITH RESPECT TO ...;
        ROTATE(B2,0,2*PI/23,0);                        COMMENT ...WORLD COORDINATES;
END;
END "EXAMPLE ONE";
```

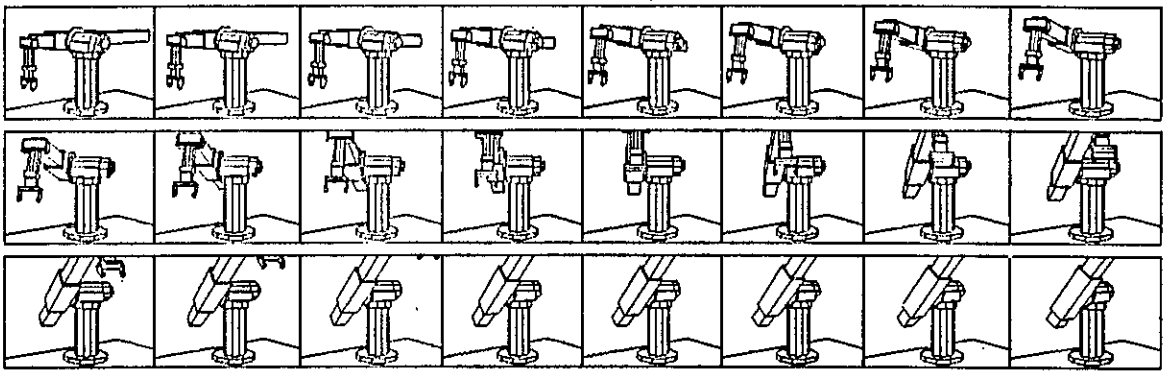
FIGURE 3.1 - THE 24 DISPLAYS OF EXAMPLE #1.



displays them rotating. The header file, GEOMES.HDR, is kept on a disk area [GEM,HE] and contains the names of the necessary load modules, the declarations of all the modeling routines and SAIL macros for accessing GEOMED data structures. After the header, the first routine to execute is MKUNIV (make universe), which initializes the data structures. Next two polyhedra are created using the MKCUBE routine which takes three arguments: width, breadth and height for specifying a rectangular right parallelepiped. All such creation routines return an integer which is the machine address of the node of the entity created. The first routine of the FOR-loop is GEODPY which refreshes the display of the

model. Finally, the example calls TRANSL and ROTATE which perform translation and rotation. TRANSL takes four argument: the thing to be moved followed by the three components of a translation vector; similarly ROTATE takes four arguments: the thing to be moved followed by the three components of a rotation vector; there are several other ways to specify translation and rotation.

FIGURE 3.2 - THE 24 DISPLAYS OF EXAMPLE #2.



```
BEGIN "EXAMPLE TWO"
        REQUIRE "GEOMES.HDR[GEM,HE]" SOURCE_FILE;       ∝ GEOMED EMBEDDED IN SAIL;
        DEFINE ∝="COMMENT"; DEFINE PI="3.1415927";      ∝ DECLARE COMMENT PREFIX;
        INTEGER B1,B2,J1,J2,J3,J4,J5,J6,C1,CHR,I;
        MKUNIV;GEOOPY;
        B1 ← INB3D("ARM[DAT,BGB]");     ∝ MODEL OF THE YELLOW ARM;
        B2 ← INB3D("TABLE[DAT,BGB]");   ∝ MODEL OF THE HAND/EYE TABLE;
        J1 ← FDNAME("JOINT1");          ∝ SHOULDER - ABOUT VERTICAL;
        J2 ← FDNAME("JOINT2");          ∝ ARM - ABOUT HORIZONTAL;
        J3 ← FDNAME("JOINT3");          ∝ SLIDE;
        J4 ← FDNAME("JOINT4");          ∝ WRIST TWIST;
        J5 ← FDNAME("JOINT5");          ∝ WRIST FLAP;
        J6 ← FDNAME("JOINT6");          ∝ HAND;
        C1 ← INCAM("ARMCAM[DAT,BGB]"); ∝ INPUT A PARTICULAR CAMERA MODEL;
FOR I←1 STEP 1 UNTIL 24 DO              ∝ TWENTY FOUR IMAGES FOR FIGURE 3.2;
BEGIN
        SHOW2(0,0);                     ∝ HIDDEN LINE ELIMINATION DISPLAY REFRESH;
        PLOTO("PLTX2."&CVS(I));         ∝ OUTPUT LATEST DISPLAY FILE TO DISK;
        ROTATE(-J1,0,0,PI/40);          ∝ ACTION WITH RESPECT TO BODY COORDINATES...;
        ROTATE(-J2,0,0,-PI/80);         ∝ ...WHEN BODY ARGUMENT IS GIVEN NEGATIVE;
        TRANSL(-J3,0,0,0.06);
END;
END "EXAMPLE TWO";
```

In example #2, the model of an actual robot arm is read in and the first three joints are run through a simulated arm motion. The routine INB3D reads a B3D polyhedron file from the disk. The arm was drawn from measurements using the interactive form of GEOMED. The FDNAME, find name, routine retrieves a body by its print name; FDNAME returns zero when a name is not found. The routine INCAM reads in a camera file. Finally, the routine SHOW2 calls the hidden line eliminator; when SHOW2's arguments are zero, default options are assumed. The arm model was originally made

to illustrate an arm trajectory for a thesis on arm control (Paul 69) and has been used two times since in projects concerning arm trajectory planning and arm collision avoidance.

GEOMED is a hierarcy of several levels of routines that are finally invoked by syntactically trivial subroutine calls. The point illustrated by the examples is that some applications level GEOMED code has a quite ordinary appearance that does not require mastery of the many underlying primitives which are explained in the next several sections.

3.1 Euler Primitives.

The Euler routines are based on the idea that an arbitrary polyhedron can be created in steps that always maintain the Euler relation: F-E+V=2*(B-H). Topologically, a connected Eulerian polyhedral graph can be built up with only four creation primitives: MKBFV, MKEV, MKFE and GLUEE or taken apart with four kill primitives: KLBFEV, KLEV, KLFE and UNGLUEE. The prefixes "MK" and "KL", stand for *make* and *kill*; the initials "B", "F", "E" and "V" invariably stand for *body*, *face*, *edge* and *vertex* and tend to appear in that order. The notion of *GLUE* is associated with the process of forming (or removing) a handle which increases (or decreases) the topological genus of the surface by one unit. The MKBFV primitive takes no arguments and creates a degenerate point polyhedron of one vertex, one face and one body which is the minimal non-zero binding satisfying the Euler relation. The MKEV creates a new edge and a new vertex, the new edge is attached to the old vertex as a spur in the perimeter of the given face. The MKFE creates a new face and a new edge, the new edge is placed between the two given vertices. And the GLUEE routine creates a handle or kills a body node by placing a new edge between two given vertices and by removing the second of two given faces. Completing the set, the ESPLIT routine (explained in Section 2.5) is included as a form of MKEV.

In principle, the advantages of the pure Euler primitives are that they assure valid topology, full generality, reasonable simplicity and they achieve a semantic level slightly higher than that of manipulating the nodes and links directly. However, the Euler primitives only satisfy the first of the conditions defining a solid polyhedron; imposing no particular restrictions on surface orientation, face/vertex trivalence, face planarity, face convexity or surface self intersection. Furthermore, even

some low level topological operations (such as body intersection, Chapter 5) are inconvenient to specify in term of the Euler primitives. Nevertheless in practice, the Euler primitives perform a useful role as a topological foundation for coding routines which embody more algebra and geometry and which lead to higher semantic levels.

BOX 3.1 THE EULER PRIMITIVES.

EULER MAKE PRIMITIVES:

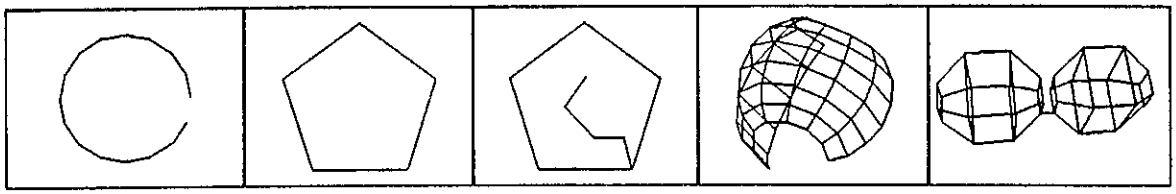
1.	BNEW ← MKBFV;	Makes point polyhedron.
2.	VNEW ← MKEV(F,V);	Makes new edge and vertex.
	VNEW ← ESPLIT(E);	Makes new edge and vertex.
3.	ENEW ← MKFE(V1,F,V2);	Makes new face and edge.
4.	ENEW ← GLUEE(F1,V1,F2,V2);	Makes new edge, kills F2, and makes a hole or kills a body.

EULER KILL PRIMITIVES:

1.	QNEW ← KLBFEV(Q);	Kills bodies, faces, edge and vertices.
2.	FACE ← KLFE(E);	Kills E and NFACE(E). Returns PFACE(E).
3.	EDGE ← KLEV(V);	Kills V and PED(V). Returns other E of V.
	VERT ← KLEV(E);	Kills E and NVT(E). Returns PVT(E).
4.	FNEW ← UNGLUE(E);	Kills E, makes F. Returns the new face, and kills a hole or makes a body.

The remainder of this section consists of more explanation and examples of the Euler primitives and may be skipped by the reader who does not need an elaboration of this level of modeling. *Non-solid polyhedra*: Intermediate between Eulerian and solid polyhedra are the wire, dangling-wire (or spur), lamina, sheet and wasp-edged polyhedra which are transition states for creating and altering polyhedral solids. The *wire* polyhedron consists of one face, N edges and N+1 vertices. A *lamina* is a two faced polyhedron with no interior edges or dangling wire. A *dangling wire* or *spur* is made when a MKEV is applied to a vertex of an already closed simply connected face perimeter; dangling wire spurs are ultimately "closed" or "tied down" by a MKFE application. A *sheet* is an array of lamina, with the exception of ruled surfaces of rotation, commands for folding and manipulating sheets have not been developed. Finally, a *wasp* polyhedron is a transition state formed by the GLUEE primitive; this degenerate polyhedron is named for the wasp waisted face perimeter which (like a spur) is eliminated by appropriate MKFE applications.

FIGURE 3.3 - FIVE KINDS OF NON-SOLID POLYHEDRA.



The use of the Euler primitives is limited to the above transition states. MKEV sweeps a MKBFV point body into a wire, the wire may be continued (at only its newest end) by additional MKEVs until it is closed into a lamina by MKFEing the first and last vertices of the wire. The MKFE is oriented such that if the wire is planar and the resulting lamina is homogeneous (non-self-intersecting); then the exterior vector of the newly created face points into the counter clockwise halfspace of the lamina, the halfspace from which the order of creation of the vertices appears to be counter clockwise. This particular generation by Euler sweeping from point, through wire and lamina, to solid is illustrated by the make hexahedron example #3 and by the make tetrahedron example #4; the final example of this section, example #5, illustrates the use of GLUEE.

Example 3 - Make Hexahedron.

```
BEGIN "EXAMPLE THREE"
        REQUIRE "GEOMES.HDR[GEM,HE]" SOURCE_FILE;       α GEOMED EMBEDDED IN SAIL;
INTEGER PROCEDURE MAKECUBE(REAL DX,DY,DZ);
BEGIN "MAKECUBE"
        INTEGER B,F,E,V1,V2,V3,V4;
        DEFINE α="COMMENT";                             α COMMENT DELIMITER;
α MAKE RECTANGULAR LAMINA;
        B ← MKBFV;      F ← PFACE(B);   V1 ← PVT(B);    α MAKE POINT POLYHDERA;
        XWC(V1) ← DX/2; YWC(V1) ← DY/2; ZWC(V1) ←-DZ/2; α POSITION FIRST VERTEX;
        V2 ← MKEV(F,V1); XWC(V2) ← -DX/2;               α MAKE AND POSITION 2ND VERTEX;
        V3 ← MKEV(F,V2); YWC(V3) ← -DY/2;               α MAKE AND POSITION 3RD VERTEX;
        V4 ← MKEV(F,V3); XWC(V4) ←  DX/2;               α MAKE AND POSITION 4TH VERTEX;
        MKFE(V1,F,V4); F ← PFACE(F);
α MAKE FOUR SPURS ON THE LAMINA;
        V1 ← MKEV(F,V1);V2 ← MKEV(F,V2);
        V3 ← MKEV(F,V3);V4 ← MKEV(F,V4);
        ZWC(V1) ← ZWC(V2) ← ZWC(V3) ← ZWC(V4) ← DZ/2;   α POSITION LAST FOUR VERTICES;
α JOIN SPURS TO FORM FINAL FACE;
        MKFE(V1,F,V2);  MKFE(V2,F,V3);
        MKFE(V3,F,V4);  MKFE(V4,F,V1);
        RETURN(B);
END "MAKECUBE";
        MKUNIV; MAKECUBE(10,8,6);                       α TEST CALL ON MAKECUBE;
END "EXAMPLE THREE";
```

Example 4 - Make Regular Tetrahedron.

```
BEGIN "EXAMPLE FOUR"
        REQUIRE "GEOMES.HDR[GEM,HE]" SOURCE_FILE;       α GEOMED EMBEDDED IN SAIL;
        DEFINE α="COMMENT";DEFINE PI="3.1415927";
INTEGER PROCEDURE MKTETRA (REAL R);                     α MAKE TETRAHEDRON;
BEGIN "MKTETRA"
        INTEGER B,F1,F2,V1,V2,V3,V4;
        B ← MKBFV; F1 ← PFACE(B); V1 ← PVT(B);          α MAKE POINT POLYHDERA;
        XWC(V1) ← ABS(R*0.942809); ZWC(V1) ← -ABS(R/3); α POSITION FIRST VERTEX;
        V2 ← MKEV(F1,V1); ROTATE(V2,0,0,2*PI/3);        α MAKE AND POSITION 2ND VERTEX;
        V3 ← MKEV(F1,V2); ROTATE(V3,0,0,2*PI/3);        α MAKE AND POSITION 3RD VERTEX;
        V4 ← MKEV(F1,V3);                               α MAKE AND POSITION 4TH VERTEX;
        XWC(V4)←YWC(V4)←0;ZWC(V4)←ABS(R);
        MKFE(V1,F1,V4); F2 ← PFACE(F1);                 α CLOSE SKEW QUADRILATERAL;
        MKFE(V1,F1,V3); MKFE(V2,F2,V4);
        RETURN(B);                                      α RETURN THE CREATION;
END "MKTETRA";
        MKUNIV; MKTETRA(6);                             α INITIALIZE AND TEST MKTETRA;
        GEODPY;                                         α DISPLAY REFRESH;
END "EXAMPLE FOUR";
```

Example 5 - Glue two N-edged faces together.

```
BEGIN "EXAMPLE FIVE"
        REQUIRE "GEOMES.HDR[GEM,HE]" SOURCE_FILE;       α GEOMED EMBEDDED IN SAIL;
        DEFINE α="COMMENT"; DEFINE PI="3.1415927";
        INTEGER B1,B2;                                  α TWO TEST CUBES;
INTEGER PROCEDURE GLUEFF(INTEGER FACE1,FACE2);          α DEMO GLUE FACE TO FACE;
BEGIN "GLUEFF"
        INTEGER V,V1,V2,E,E0,I; REAL DMIN,D;
        V1 ← VCCW(PED(FACE1),FACE1);                    α PICK ONE VERTEX OF FACE1;
α FIND VERTEX OF FACE2 THAT IS CLOSEST TO V1;
        DMIN ← 10@10; E ← E0 ← PED(FACE2);              α INITIALIZE MINIMAL DISTANCE;
        DO BEGIN
                V ← VCCW(E,FACE2);D ← DISTAN(V1,V);     α SCAN FACE2 FOR VERTEX CLOSEST TO V1;
                IF D<DMIN THEN BEGIN DMIN←D;V2←V;END;
        END UNTIL E0 = (E←ECCW(E,FACE2));
α MAKE THE WASP EDGE;
        E ← GLUEE(FACE1,V1,FACE2,V2);                   α FACE2 AND BODY ARE KILLED;
α CLOSE OTHER EDGES;
        V ← OTHER(NCCW(E),V1);                          α LAST VERTEX, TO STOP SCAN;
        DO BEGIN
                V1 ← OTHER(PCW(E),V1);                  α FETCH NEXT PAIR OF VERTICES;
                V2 ← OTHER(PCCW(E),V2);
                E ← MKFE(V1,FACE1,V2);                  α CLOSE AN EDGE;
        END UNTIL V=V1;
        RETURN(BGET(E));                                α RETURN THE SURVIVING BODY;
END "GLUEFF";
        MKUNIV;                                         α INITIALIZATION;
        B1 ← MKCUBE(2,2,2); B2 ← MKCUBE(3,3,3);         α TWO TEST CUBES;
        ROTATE(B1,0,-PI/2,0);TRANSL(B1,-3,0,0);         α ORIENT CUBES SO FIRST FACES...;
        ROTATE(B2,0,+PI/2,0);TRANSL(B2,+4,0,0);         α ...ARE OPPOSITE;
        GLUEFF(PFACE(B1),PFACE(B2));                    α TEST THE FUNCTION;
        GEODPY;                                         α DISPLAY REFRESH;
END "EXAMPLE FIVE";
```

3.2 Routines using Euler Primitives.

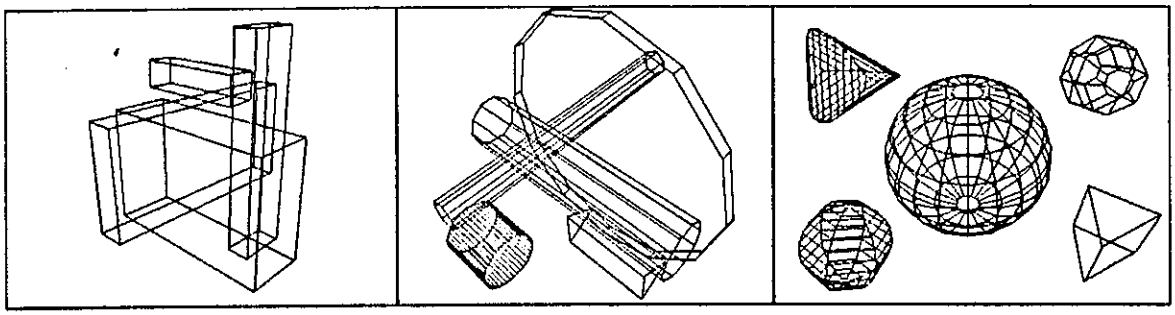
Further methods of polyhedral construction can readily be coded using the Euler primitives. For example, the routines listed in Box 3.2 illustrate the direct generation of simple prototypical polyhedra, as well as contruction by sweeping, cutting, glueing, copying and duality.

BOX 3.2 ROUTINES USING EULER PRIMITIVES.

1.	BNEW ←	MKCUBE(DX,DY,DZ);	Create right rectangular prism.
2.	BNEW ←	MKCYLN(RADIUS,N,DZ);	Create cylinder approximation.
3.	BNEW ←	MKBALL(RADIUS,M,N);	Create sphere approximation.
4.	FACE ←	SWEEP(FACE,FLAG);	Make prism on face (or sweep wire).
5.	FACE ←	ROTCOM(FACE);	Rotation sweep wire face completion.
6.	PEAK ←	PYRAMID(FV);	Make pyramid on a face (or vertex).
7.	BODY ←	GLUE(FACE1,FACE2);	Removes face1 and face2.
8.	BNEW ←	MKCUT(BODY,X,Y,Z);	Divide body at cutting plane.
9.	QNEW ←	MKCOPY(ENTITY);	Copy an entity.
10.	BODY ←	FVDUAL(BODY);	Apply face/vertex duality to a body.

The first three routines make cubic prisms as well as polyhedral approximations to circular cylinders and spheres; or more accurately, MKCUBE creates rectangular right prisms, MKCYLN creates regular polygonal right cylinders and MKBALL creates hedrons faceted by two N-sided regular polar polygons and N*(M-1) trapezoidal polygons with all vertices lying on the surface of a sphere of a given radius.

FIGURE 3.4 - Examples of MKCUBE, MKCYLN and MKBALL.

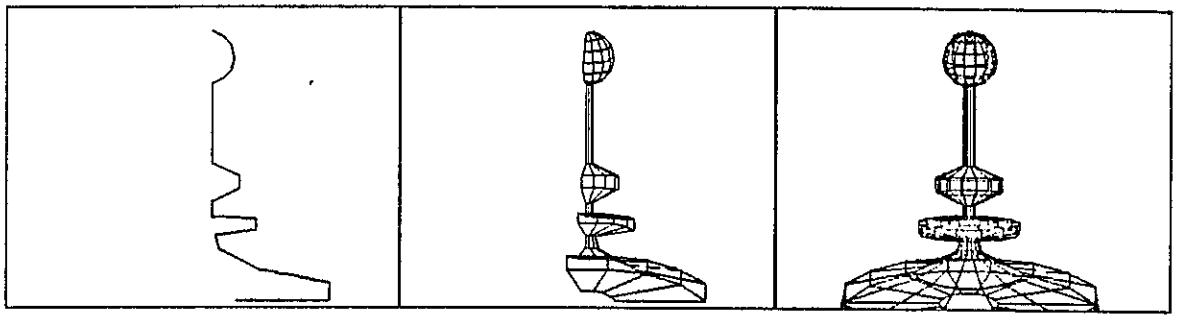


MKCUBE Results MKCYLN Results MKBALL Results

Although, the implementation of curved edges and curved faces in GEOMED has always been *just around the corner*, I have balked at the idea because it would require additional nodes connected to edges and faces or it would require expanding the node size, which I have always before taken as

an omen for restarting from scratch. There have so far been four cold starts: GEOMED I, 1969, was based on sweep primitives and was written in LEAP/SAIL; GEOMED II, 1970, was based on winged edge primitives and was written SAIL without using LEAP; GEOMED III, 1971, was written SAIL and FAIL; GEOMED IV, 1972 to present, is written in FAIL. Future mythical GEOMED's include export GEOMED V, coded in simple international ALGOL for export; a big GEOMED VI, larger nodes for curved object representation of smooth manifolds rather than polyhedra; a small GEOMED VII coded for a mini computer; and finally a 4-D GEOMED VIII for four dimensional modeling.

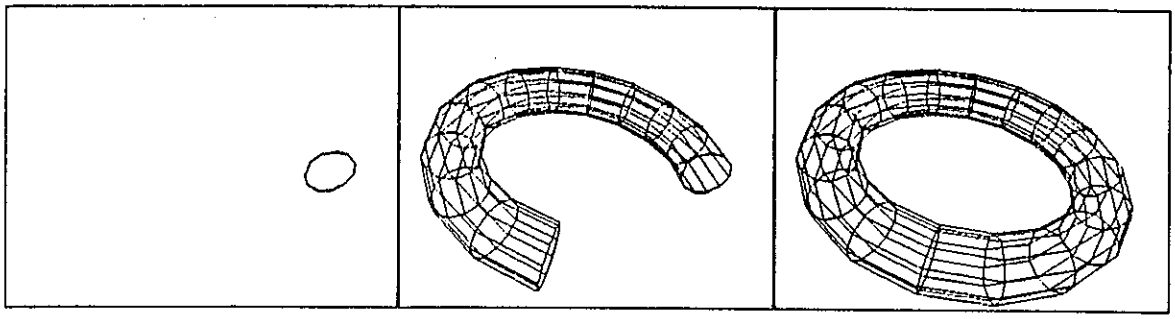
FIGURE 3.5 - Creation of a Solid of Rotation by Sweeping a Wire.



Initial Wire | After four SWEEPs | After ROTCOM

The three sweep primitives SWEEP, ROTCOM and PYRAMID involve the non-solid Euler polyhedra: wire, lamina and sheets. A lone vertex body can be swept into a wire, a wire can be closed to form a lamina or a wire can be swept into a sheet, and a sheet can be closed to form a solid polyhedron. Figure 3.5 illustrates the creation of a solid by sweeping a wire-face, using SWEEP(FACE,0), to form a sheet. Figure 3.6 illustrates the creation of a solid by sweeping a normal face as well as the use of the GLUE(FACE1,FACE2) primitive to close a torus.

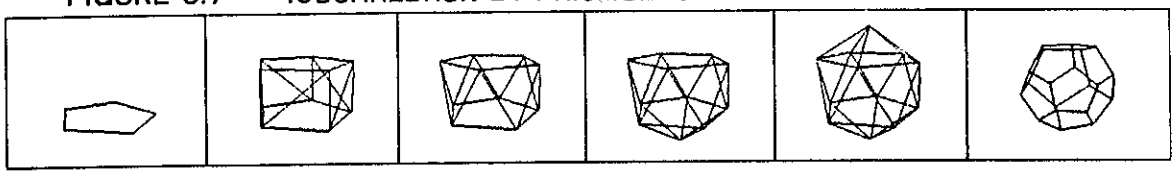
FIGURE 3.6 - Sweep and Glue.



Initial Face Lamina | After twelve SWEEPS | After GLUE

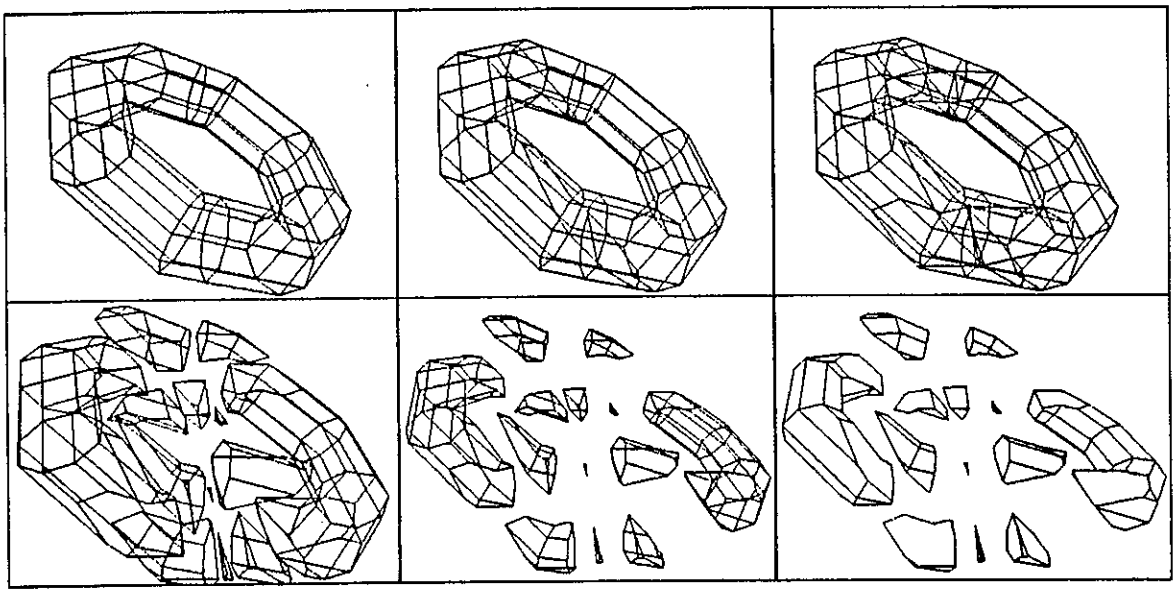
The sweep flag argument determines whether triangles (flag non-zero) or rectangles (flag zero) are to be formed as the sweep of the edges of the face. Sweeping out rectangles forms prisms, sweeping out triangles forms prismoids. The PYRAMID routine when applied to a face creates a peak vertex at the average locus of vertices of the face and connects all the vertices of the given face to the peak vertex. PYRAMID applied to a vertex coerces all the faces of the vertex to be triangles, the interpretation being that the given vertex is to be made like a peak of a pyramid. Prismoid sweep and face pyramiding are illustrated by the construction of an icosahedron in Figure 3.7; the icosahedron can be changed into a dodecahedron by the DUAL routine. The DUAL routine mungs face nodes into vertex nodes and vertex nodes into face nodes; the new vertices are placed at the arithmetic mean of the vertices of the old faces, consequently the dual is not its own inverse since objects tend to shrink.

FIGURE 3.7 - ICOSAHEDRON BY PRISMOID SWEEP AND PYRAMID SWEEP.



The MKCUT(BODY,X,Y,Z) primitive divides a body at cutting plane into as many pieces as necessary. Figure 3.8 illustrates how to cut a toroidal polyhedron into thirteen pieces using only three cutting planes, after Figure 63 of (Gardner 61). The action of MKCOPY should be obvious - a new polyhedron is returned that has the same topology, geometry and photometry as the given polyhedron. More routines using Euler primitives could be coded for particular applications in architecture, computer animation, mechanical design, numerical machine control, assembly diagraming and so on.

FIGURE 3.8 - THREE CUT TORUS DISSECTION INTO THIRTEEN PARTS.



3.3 Euclidean Routines.

The Euclidean routines of GEOMED fall roughly into four groups: transformations, metrics, tram routines and space simulators. The Euclidean transformations are translation, rotation, dilation and reflection following Klein's Erlangen Program, 1872. The Euclidean metric routines compute distances, angles, areas, volumes and inertia tensors. The tram routines create or alter tram nodes which are the main topic of this section. The final group of routines perform spatial simulations such as collision, intersection, propinquity, occupancy and occultation.

Tram Nodes. A tram node contains twelve real numbers. Fundamental to all the Euclidean routines is the curious fact that tram nodes have two interpretations: they may represent a coordinate system or they may represent a Euclidean transformation. As a coordinate system, the twelve numbers contain a location of the origin of the coordinate system as well as the three components of each of the three unit vectors of the axes of the coordinate system. As a transformation, the application of a tram node to a vertex is defined by the procedure named SCREW, given below.

Tram as a Coordinate System:	*Tram Node Data Field Names*			
location of origin of coordinates:	XWC,	YWC,	ZWC,	LOCATION VECTOR.
components of X-axis unit vector:	IX,	IY,	IZ,	
components of Y-axis unit vector:	JX,	JY,	JZ,	ORIENTATION MATRIX.
components of Z-axis unit vector:	KX,	KY,	KZ.	

Tram as a Transformation:

```
COMMENT APPLY TRAM Q TO VERTEX V POSTFIX;
PROCEDURE SCREW (INTEGER V,Q);
BEGIN   REAL X,Y,Z;
        X ← XWC(V);     Y ← YWC(V);     Z ← ZWC(V);
        XWC(V) ← X*IX(Q) + Y*JX(Q) + Z*KX(Q) + XWC(Q);
        YWC(V) ← X*IY(Q) + Y*JY(Q) + Z*KY(Q) + YWC(Q);
        ZWC(V) ← X*IZ(Q) + Y*JZ(Q) + Z*KZ(Q) + ZWC(Q);
END;
```

Generalizing, the procedure APTRAM(ENTITY,TRAM) applies a tram to an arbitrary entity. The APTRAM procedure is formed by surrounding the SCREW procedure with suitable type checking and data structure tracing mechanisms so that a tram can be applied (postfix) to almost anything: bodies, faces, edges, vertices, as well as to other trams, camera models and window nodes.

To repeat for emphasis, a tram node has two interpretations; a tram node may be interpreted as a coordinate system and the very same tram node may be interpreted as a Euclidean transformation. A source of confusion, is that a coordinate system tram is a definition of one coordiate system (call it the body coordinates) in terms of another coordinate system (call it the world coordinates). The application of a body coordinate system tram to an entity in body coordinates brings the entity down into the world coordinate system in which the tram is defined. To say it another way, the rule is that APTRAM(BODY,TRAM) converts from body coordinates to world coordinates, whereas APTRAM(BODY,INTRAM(TRAM)) converts world coordinates to body coordinates. The procedure INTRAM inverts a tram node in the manner given below. As alluded to in example #2, body nodes carry a pointer to a tram defining a system of body coordinates so that Euclidean transformtions can be relocated relative to arbitrary coordinate systems.

```
INTEGER PROCEDURE INTRAN (INTEGER Q);
BEGIN "INTRAM"
        REAL X,Y,Z;
        X ← XWC(Q);     Y ← YWC(Q);     Z ← ZWC(Q);
        XWC(Q) ← -(X*IX(Q) + Y*IY(Q) + Z*IZ(Q));
        YWC(Q) ← -(X*JX(Q) + Y*JY(Q) + Z*JZ(Q));
        ZWC(Q) ← -(X*KX(Q) + Y*KY(Q) + Z*KZ(Q));
        IY(Q) ↔ JX(Q);  IZ(Q) ↔ KX(Q);  JZ(Q) ↔ KY(Q);  COMMENT TRANSPOSE;
        RETURN(Q);
END "INTRAM";
```

BOX 3.3 EUCLIDEAN TRANSFORMATIONS

```
ENTITY  ←  APTRAM(ENTITY,TRAM);
TRAM    ←  INTRAM(TRAM);
RESULT  ←  TRANSL(XWD(TRAM,ENTITY),DX,DY,DZ);
RESULT  ←  ROTATE(XWD(TRAM,ENTITY),WX,WY,WZ);
RESULT  ←  SHRINK(XWD(TRAM,ENTITY),SX,SY,SZ);
```

Pragmatically, the creation, relocation and application of a tram node are invoked all at once by an appropriate Euclidean transformation routine. The transformation routines are listed in Box 3.3 with APTRAM and INTRAM. As a further pragmatic device, the first argument of the Euclideans is "microcoded" using the XWD notation which packs two links into one word. The expression XWD(A,B) is equivalent to the expression (A*2↑18 + (B MOD 2↑18)), where A and B are positive integers. When the entity of the first argument of the Euclidean routines is zero, the transformations create and return a tram node; when the entity of the first argument is nonzero, the transformations create a tram, apply

it to the entity, kill the tram node and return the entity. When the first argument carries a tram as well as an entity (using the XWD notation) the desired transformation (or creation) is done with respect to the coordinate system defined in the given tram, (this is called coordinate relocation). When the first argument is negative the body coordinates tram is retrieved and used for relocation of the transformation. Most bodies carry a tram pointer (in the link field named TRAM) which defines body coordinates; the body coordinates of a face, edge or vertex are taken as the TRAM of the BGET of the face, edge or body; a zero TRAM link is mapped into a zero translation, unit rotation matrix tram by all the Euclidean routines. Finally, the actual transformation is specified by giving three components of a vector; the meaning of a translation vector is obvious, rotation vectors are explained in a subsequent paragraph and a scale vector is a triple of factors which are multiplied into the corresponding components of all the vertices of an entity with respect to the axes of transformation. Reflections are specified as negative shrinks; a reflection on one or on three axes will evert a body's surface orientation.

Further routines to create and alter tram nodes are listed in Box 3.4. The MKTRAM routine simply returns an identity tram with zero translation and zero rotation (that is a unit rotation matrix). The MKTRMA routine creates a tram from the Euler angles pan, tilt and swing; see (Goldstein 1950). The Euler angles come conveniently close to the rotational degrees of freedom of automatic camera mounts, but unlike a rotation vector the Euler angles are discontinous at zenith and nadir.

BOX 3.4 TRAM ROUTINES

TRAM ← MKTRAM;	Returns an identity tram.
TRAM ← MKTRMA(PAN,TILT,SWING);	Makes a tram from Euler angles.
TRAM ← MKTRMF(FACE);	Makes a tram from a Face.
TRAM ← MKTRME(EDGE);	Makes a tram from an Edge.
TRAM ← MKTRMV(WX,WY,WZ);	Makes a tram from a rotation vector.
TRAM ← NORM(TRAM);	Normalization to unit vectors.
TRAM ← ORTHO1(TRAM);	Orthogonalize by worst case.
TRAM ← ORTHO2(TRAM);	Orthogonalize by two cross products: K ← (I CROSS J) and J ← (K CROSS I).

The Rotation Matrix. The nine elements named IX, IY, IZ, JX, JY, JZ, KX, KY and KZ form what is know as a three by three rotation matrix. By virtue of the definition of rigid object rotation, the tram rotation matrix must be maintained orthonormal. (The trams created by SHRINK are tolerated as a

special case which are not considered to be rigid rotations.) Orthonormality is maintained with the aid of three routines: NORM(TRAM) which normalizes the row vectors of a tram rotation matrix; ORTHO1 which orthogonalizes a rotation matrix by comparing the sums of pairs of dot products of pairs of the three unit vectors; the unit vector that is most out of allignment is recomputed by crossing the other two (ORTHO1 performs its check twice and then exits); and ORTHO2, which coerces orthogonality by setting row vector K to the cross product of rows I and J, followed by setting row vector J to the cross product of rows K and I.

The Rotation Vector. All 3-D rotations can be expressed as a vector where the direction of the vector specifies the axis of rotation and where the magnitude of the vector specifies the amount of rotation in radians. Given such a rotation vector WX, WY, WZ with direction cosines CX, CY, CZ and magnitude W in radians; let CW be cosine(W) and SW be sine(W); and let a function called SIGN return positive or negative one depending on whether its argument is positive or negative; then the relation between a rotation matrix and a rotation vector can be listed:

Rotation vector to Rotation matrix:

```
IX = (1-CW)*CX*CX + CW;    IY = (1-CW)*CY*CX + CZ*SW;    IZ = (1-CW)*CZ*CX - CY*SW;
JX = (1-CW)*CX*CY - CZ*SW; JY = (1-CW)*CY*CY + CW;       JZ = (1-CW)*CZ*CY + CX*SW;
KX = (1-CW)*CX*CZ + CY*SW; KY = (1-CW)*CY*CZ - CX*SW;    KZ = (1-CW)*CZ*CZ + CW;
```

Rotation matrix to Rotation vector:

```
WX = SIGN(JZ-KY)*ACOS(0.5*(IX+JY+KZ-1))*SQRT(+IX-JY-KZ)/(3-IX-JY-KZ));
WY = SIGN(KX-IZ)*ACOS(0.5*(IX+JY+KZ-1))*SQRT(-IX+JY-KZ)/(3-IX-JY-KZ));
WZ = SIGN(IY-JX)*ACOS(0.5*(IX+JY+KZ-1))*SQRT(-IX-JY+KZ)/(3-IX-JY-KZ));
```

Homogeneous Coordinates. The Euclidean routines involving trams could be written out in terms of the 4-D homogeneous coordinates frequently found in computer graphics, by prefixing a column to each tram and a fourth component to each vertex.

$$\text{TRAM4D} = \begin{matrix} 1 & XWC & YWC & ZWC \\ 0 & IX & IY & IZ \\ 0 & JX & JY & JZ \\ 0 & KX & KY & KZ \end{matrix}$$

I did not use homogeneous coordinates in GEOMED for three reasons: first, the computer at hand, (a PDP-10) has floating point arithmetic hardware so that homogeneous components were not needed for

numerical scaling; second, the homogeneous representation requires more coordinates per vertex and more multiplications per transformation than the GEOMED representation; and third, my intuition is stronger in affine metric geometry than it is in homogeneous projective geometry.

Standard Conventions. There are several nettlesome details related to rotation, translation and projection among which a computer geometer must distinguish: (i). matrix vs. algebraic notation; (ii). postfix vs. prefix transformation application; (iii). row vs. column vertices; (iv). 4-D homogeneous vs. 3-D affine coordinates; (v). rotation vector vs. Euler angles and so on. At the moment, I favor algebraic notation, postfix transformations, row vertices, 3-D coordinates and rotation specification by vector; a demonstrably superior natural set of standard conventions probably does not exist.

In GEOMED, tram nodes were until recently called frame nodes, however I wish to abandon all use of the word *frame* for three reasons: first, the term is ambiguous and overused (even within graphics alone); second, the term does not include the notion of transformation; and third, the term risks confusion (or association) with the connotations of (Minsky 74) and (Winograd 74); i.e. the connotation of a *Frame System* as a modular mental universe of stereotyped world situations. In geometric modeling, the word *frame* can be replaced in all three of its usual graphics applications: the *frame of reference* or *coordinate frame* is now a *coordinate system*, the *frame* of a movie film is now an *image*, the *frame* of a display screen is now a *window* or *border*.

Metric Routines. Given one or several geometric entities, the Euclidean metric routines listed in Box 3.5 compute length, area, volume, angle or moments of inertia. The DISTANCE routine computes the distance between two anythings in a reasonable manner; the measure routine returns the volume, area or length of bodies, faces or edges respectively (by a pragmatic argument hack, the measure of a negative body is its surface area). The ANGLE routine computes the angle between two entities by returning the arc cosine of the normalized inner product of two vectors: vertices are interpreted as vectors from the origin of the body in which they belong, edge are vectors from their NVT to their PVT, faces are taken as their normal vector and bodies are represented by the K unit vector of their body coordinates tram; trams and cameras also are mapped into K unit vectors.

BOX 3.5 METRIC ROUTINES

VALUE	←	DISTANCE(ENTITY,ENTITY);
VALUE	←	MEASURE(ENTITY);
RADIANS	←	ANGLE(ENTITY,ENTITY);
RADIANS	←	ANGL3V(V1,V2,V3);
RADIANS	←	ANGLCW(EDGE);
RADIANS	←	ANGLCCW(EDGE);
VALUE	←	DETERM(TRAM);
NODE	←	INERTIA(BODY);

Since the arc cosine function returns an angular value between zero and pi; the routines ANGL3V, ANGLCW and ANGLCCW employ the arc tangent to compute an angular value between negative pi and positive pi. The ANGL3V return the angle between the vector (V3-V2) and (V2-V1), the ANGLCW(E) returns the angle between E and PCW(E), ANGLCW(-E) returns arctan of E and NCW(E); likewise ANGLCCW returns values for E and PCCW(E) or E and NCCW(W). The DETERM of a tram is the determinate of the rotation matrix of a tram. Finally, the INERTIA of a body is a sixtuple: MXX, MYY, MZZ, PXY, PXZ, PYZ packed into the first six words of a node and representing the moments and products of the intertia tensor of a polyhedron of uniform (unit) density associated with the given body. The inertia routine takes the liberty of updating the origin of the body coordinates to correspond to the center of mass and to orient the K unit vector of the body parallel to the principal axis of inertia.

Spatial Simulation. The difficult space routines perform occultation and intersection and are explained in Chapters 4 and 5 respectively. The simple space routines, listed in Box 3.6, perform propinquity, collision detection and spatial compare.

BOX 3.6 SIMPLE SPACE ROUTINES

HEXAHEDRON	←	MKBUCK(BODY);
V-PIERCE	←	COMPFE(FACE,EDGE);
FLAG	←	COMPEE(EDGE,EDGE);
FLAG	←	WITH2D(FACE,VERTEX);
FLAG	←	WITH3D(BODY,VERTEX);
FLAG	←	COLDET(B1,B2,EPSILON).

The MKBUCK routine returns a hexahedron that buckets the given body. The COMPFE compares a face and an edge in 3-D for intersection, if the arguments are disjoint then zero is returned, if the

arguments intersect then the edge is split and the new vertex is positioned at the locus where the edge pierces the face. The COMPEE routine determines whether two edges cross in a given perspective view. The within 2-D routine, WITH2D, determines whether a vertex appears to be interior to a given face in a perspective view and the WITH3D determines whether a given vertex falls interior to a body in 3-D. The COLDET routine compares all the vertices and faces of two objects for propinquity within an epsilson as well as all the edges of the two objects. Temporary collision pointers are left between vertices and the nearest alien collision face as well as between temporary collision vertices. Collision vertices are formed at the foot of the shortest line segment between the skew lines of two edges that pass within the epsilon distance of each other.

3.4 Image Synthesis: Perspective Projection and Clipping.

Image synthesis is the process of generating various kinds of images: vector display, video, contour map or mosaic. Independent of the final image representation the process always requires the operations of perspective projection and clipping. The perspective projection takes the 3-D world locus of every potentially visible vertex and computes a 3-D camera center coordinate locus followed by a perspective projection in the fashion illustrated in the PROJECT procedure given below.

```
INTEGER PROCEDURE PROJECT (INTEGER V,CAMERA);
BEGIN "PROJECT"
        INTEGER TRM; REAL X,Y,Z,XCC,YCC,ZCC;
COMMENT TRANSFORM FROM WORLD COORDINATES TO CAMERA COORDIATES;
        TRM ← TRAN(CAMERA);
        X ← XWC(V) - XWC(TRM);
        Y ← YWC(V) - YWC(TRM);
        Z ← ZWC(V) - ZWC(TRM);
        XCC ← X*IX(TRM) + Y*IY(TRM) + Z*IZ(TRM);
        YCC ← X*JX(TRM) + Y*JY(TRM) + Z*JZ(TRM);
        ZCC ← X*KX(TRM) + Y*KY(TRM) + Z*KZ(TRM);
COMMENT PERSPECTIVE PROJECTION TRANSFORMATION;
COMMENT NOTA BENE: ZPP(V) is positive when vertex is in view of camera ! ;
        XPP(V) ← SCALEX(CAMERA)*XCC/ZCC;        COMMENT ( SCALEX = -FOCAL/PDX );
        YPP(V) ← SCALEY(CAMERA)*YCC/ZCC;        COMMENT ( SCALEY = -FOCAL/PDY );
        ZPP(V) ← SCALEZ(CAMERA)    /ZCC;        COMMENT ( SCALEZ = -FOCAL/PDZ );
        RETURN (V);
END "PROJECT";
```

The perspective projection transformation is a 3-D to 3-D mapping; the third component, ZPP, allows the hidden line eliminator to perform orthographic depth comparisons. The perspective projection

quite literally is taking the whole world model and crushing it into a slanty space between the camera lens center and the camera focal plane. The camera scales are defined in terms of the ficticious 3-D pixel dimensions PDX, PDY, PDZ and the physical camera focal plane distance, FOCAL. The pixel dimensions are arbitrarily defined as PDY=PDZ=40 microns and PDX=AR*PDY where AR is the aspect ratio of the camera; the aspect ratio can be directly measured by taking the ratio of the width to height of the image of a large black sphere on a white background, AR is usually almost one. The focal plane distance is typically between 10 and 50 millimeters and is derived from definition (FOCAL=FR*PDY) of the focal ratio, FR, which can be simply measured as explained in Section 9.1.

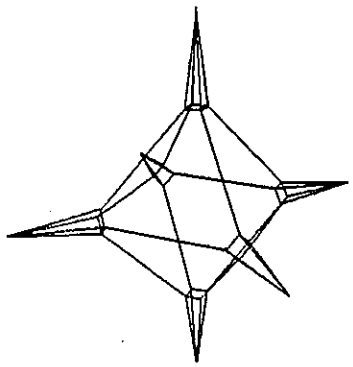
The term clipping refers to the process of computing which parts of the world model are in view of the camera. In GEOMED there are several clipper routines: one for fast transparent refresh, three for hidden line elimination and one more for clipping the contents of 2-D display windows that may be scrolled about. Three dimensional clipping can be factored into a Z-clipper and an XY-clipper. The Z-clipper determines which portions of the model are in the visible 3-D halfspace and splits edges and faces that cross the focal plane. The XY-clipper determines which portion of a 2-D perspective edge is within a given 2-D rectangular window (with sides parallel to the coordiate axes). The XY-clip is done by first applying an easy outsider test: endpoints of the edge both below, above, left or right of the window; followed by an easy insider test: endpoints of the edge both inside the window; followed by the evaluation of four polynomials of the form A*X+B*Y+C where A,B,C are the edge coefficents and X,Y are the locus of corners of the clip window. If all four polynomials have the same sign the edge is a hard outsider, otherwise the intersection of a side of the window and the edge can be detected from alternating signs and the locus of intersection can be computed from the edge coefficients.

3.5 Image Analysis: Interface to CRE.

Although there are no actual honest image analysis routines currently implemented in GEOMED, the internal GEOMED environment was designed for image based model synthesis and model verification. The routine INCRE(FILENAME) inputs from a disk file a CRE node structure that consists of a film of contour images, contour images consist of levels, levels consist of polygons and polygons

consist of vectors. In GEOMED, the CRE polygons become two-faced lamina bodies; the contour levels hierarchy becomes a parts tree structure; and a new kind of GEOMED node called an image is introduced.

The root of the GEOMED data structure is a universe node, which is the head of a ring of world nodes. World nodes have a ring of body nodes and a ring of camera nodes each camera represents a physical camera so that there might be at most three or four camera nodes. Each camera has two rings of images: a ring of perceived images and a corresponding ring of simulated images. The perceived image ring is created by INCRE and the simulated image ring is created by the hidden line eliminator, thus providing a environment for the development of polygon based image analysis. This completes the general description of the geometric modeling system called GEOMED.



SECTION 4.

HIDDEN LINE ELIMINATION FOR COMPUTER VISION.

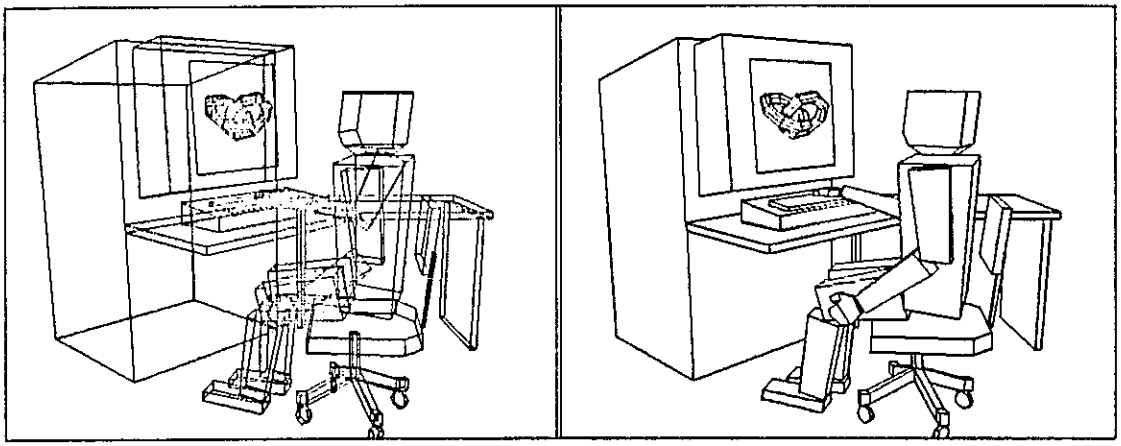
4.0 Introduction to Hidden Line Elimination.

Hidden line elimination refers to the process of simulating the appearance of opaque three dimensional objects. The phrase *hidden line elimination* dates from when the problem only involved deleting the undesired, that is the *hidden* lines, from a line drawing (Figure 4.1); today the phrase persists but connotes the wider problem of synthesizing realistic images using a computer. The present discussion is about techniques which have been implemented in a particular hidden line eliminator named OCCULT, from the Latin word *occultare* meaning *to hide*. OCCULT illustrates novel solutions to the graphics problems of exploiting object coherence and image coherence, of combining image space with model space techniques, and of sorting faces, edges and vertices in two dimensions.

OCCULT is further characterized by its intended application to computer vision and robotics. The distinguishing design requirement of a hidden line eliminator intended for vision is that it must maintain back pointers from the final 2-D images to the initial 3-D models so that the identity of features can be recovered. In computer graphics, the results of hidden line elimination are intended for human viewing

so the correspondence between the image and the model is not usually retained (unless image based model editing is being attempted). Another design goal for OCCULT was to output a connected graph of regions, edges and vertices that covers the image with no holes missing, no regions overlapping and no dangling edges. It was naively assumed that such a highly structured image representation, called a *mosaic image*, would provide a suitable basis for deriving features such as the location and orientation of high contrast edges without having to generate video images.

FIGURE 4.1 - EXAMPLE OF HIDDEN LINE ELIMINATION.



BEFORE AFTER

Hidden line eliminators appear in two previous vision systems: one by Roberts (63) and the other by Falk (70); the present system is a direct heir of the work of Falk in that the last version of the Falk system contained one of the first versions of OCCULT (installed by Richard Orban). As with image analysis, image synthesis (i.e. hidden line elimination), is a perennial research problem because it cannot be fully isolated from physical modeling. Metaphorically, hidden line elimination is the visible tip of the iceberg of physical simulation. The weaknesses of the underlying model literally show up in passing through the process of image synthesis. The present day collection of techniques is still quite lacking in realism, economy, flexibility and even reliability.

OCCULT is not a simple hidden line eliminator. In overall structure it is a combination of five techniques, Box 4.1. The first method, called *culling*, eliminates portions of the model which are hidden because of some easy to compute heuristic reason. The cull heuristics (detailed in Section 4.1) include: elimination by clipping planes, elimination by face vectors, elimination by inspection of concave

corners, and elimination by previous occultation. After the culls have been applied, the next three techniques are arranged in a three level heirarchy which comprises the main part of OCCULT. At the outermost level there is a Warnock (68) like recursive windowing method, which calls an edge-edge comparing method on small enough windows, which in turn calls a coherent object tracing method to split off and mark the portions of an object that are hidden. The methods are explained in bottom-up order: hide tracing in Section 4.2, edge-edge comparing in Section 4.3 and recursive windowing in Section 4.4. The fifth technique is a face-vertex compare method that is occasionally required to solve a particular class of cases that are missed by the edge-edge compare. The difficult part in building an OCCULT like hidden line eliminator lies in getting all the unruly beasts in harness together; the mystery being that no one beast is sufficiently strong to carry the whole burden by itself.

BOX 4.1 THE FIVE HIDDEN LINE ELIMINATION TECHNIQUES OF OCCULT.

1. Initialization Hide Culling.
2. Recursive Windowing.
3. Coherent Object Hide Tracing.
4. Edge-Edge Comparing.
5. Face-Vertex Comparing.

4.1 Initialization and Culling.

A substantial part of sophisticated hidden line elimination lies in careful attention to initial preparations. As it has now stood for the past two years, OCCULT has two input restrictions imposed for the sake of execution speed: no conflicting bodies are allowed and no concave faces are allowed. Conflicting bodies are those that occupy the same space at the same time; concave faces are faces with interiors containing a pair of points such that the line segment between the points is not contained in the face. The rational for both these restrictions is based on the optimization technique of getting computations out of inner loops; conflicting bodies and concave faces can be eliminated by employing certain polyhedral construction primitives prior to hidden line elimination. The restrictions are not inherent limitations of any of the techniques in OCCULT, so that a less restricted but slower implementation is feasible.

OCCULT is a marking algorithm, the temporary marking bits are listed in Box 4.2. The combination (POTENT and ¬VISIBLE) means potentially visible; (¬POTENT and VISIBLE) means actually visible; (¬POTENT and ¬VISIBLE) means hidden; and the combination (POTENT and VISIBLE) is an unused state that happens to be interpreted as VISIBLE.

BOX 4.2 STATUS BITS FOR OCCULT MARKINGS.

POTENT	Potentially Visible Entity.
VISIBLE	Actually Visible Entity.
PZZ	Behind the camera image plane, Positive Zcc.
NZZ	Before the camera image plane, Negative Zcc.
TMPBIT	Temporary Split edge of vertex.
FOLDED	Edge with only one POTENT face.
JOTBIT	Joint over T vertex.
JUTBIT	Joint under T vertex.

The initialization is performed in three steps: (1). vertex marking and vertex perspective projection; (2). face marking, face Z-clipping, and computation of face coefficients; and (3). edge marking and computation of edge coefficients. Two cull heuristics are done during the initialization: clipping and backside face elimination; and the other two culls are done immediately afterwards: concave corners check and the hide last hidden check.

Vertex initialization includes the prespective projection of every vertex in the model and the marking of every vertex that is in front of the camera as POTENT (potentially visible) if its perspective projected Z coordinate, ZPP(V), is greater than the simulated image plane distance, FOCAL. Two further status bits, named PZZ and NZZ, indicate positive ZCC (camera coordinates) or negative ZCC are inclusive ORed into all the faces and edges of each vertex for the sake of the Z-clipper.

Face initialization consists of Z-clipping: if a face has only its NZZ bit turned on, then it is completely behind the camera and is immediately dropped from all futher condsideration (i.e. culled out); if the face has both its PZZ and its NZZ turn on then it is Z-clipped by using the camera's image plane as a cutting plane. Next for faces in view of the camera, the 3-D perspective projected face coefficients are computed (equations given below) and the faces with their backsides towards the camera are culled out (Figure 4.2); faces surviving to this point are marked as POTENT and are placed into a list of faces of the first window of the recursive window sort.

Edge initialization consists of computing the normalized 2-D edge coefficients (equation given below) and of marking the edge as FOLDED or -FOLDED depending on whether it has one face POTENT or two faces POTENT, respectively. FOLDED edges are then inverted if necessary so that the POTENT face is the PFACE. Folded edges are illustrated in the rightmost panel of Figure 4.2. The *folded edges* are called *contour edges* by Appel(71) and Sutherland(73). The folded bit is passed along to (inclusive ORed into) the vertices of folded edges.

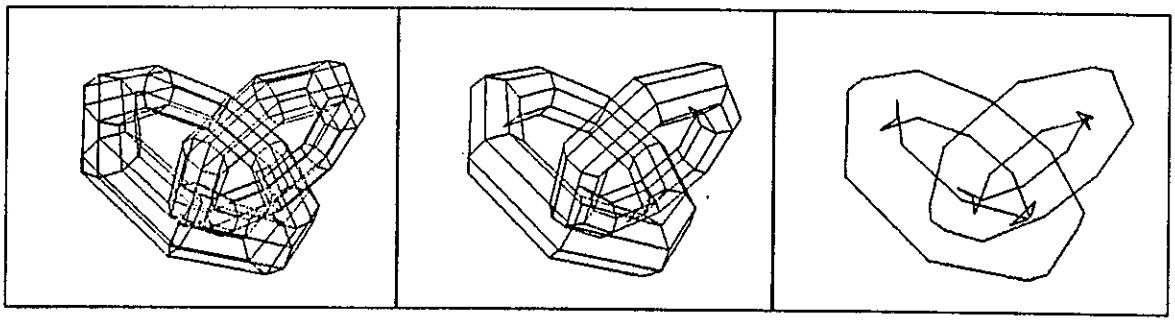
BOX 4.3 Normalized 3-D Face Coefficients:

```
E ← PED(F);V1 ← VCW(E,F);V2 ← VCCW(E,F); E ← ECCW(E,F);V3 ← VCCW(E,F);
KK(F) ← XPP(V1)*(ZPP(V2)*YPP(V3)-YPP(V2)*ZPP(V3))
      + YPP(V1)*(XPP(V2)*ZPP(V3)-ZPP(V2)*XPP(V3))
      + ZPP(V1)*(YPP(V2)*XPP(V3)-XPP(V2)*YPP(V3));
AA(F) ← (ZPP(V1)*(YPP(V2)-YPP(V3)) + ZPP(V2)*(YPP(V3)-YPP(V1)) + ZPP(V3)*(YPP(V1)-YPP(V2)));
BB(F) ← (XPP(V1)*(ZPP(V2)-ZPP(V3)) + XPP(V2)*(ZPP(V3)-ZPP(V1)) + XPP(V3)*(ZPP(V1)-ZPP(V2)));
CC(F) ← (XPP(V1)*(YPP(V3)-YPP(V2)) + XPP(V2)*(YPP(V1)-YPP(V3)) + XPP(V3)*(YPP(V2)-YPP(V1)));
TMP  ← 1/SQRT(AA(E)↑2 + BB(F)↑2 + CC(F)↑2);
AA(F) ← TMP*AA(F);BB(F) ← TMP*BB(F);CC(F) ← TMP*CC(F);
```

Normalized 2-D Edge Coefficients:

```
AA(E) ← YPP(PVT(E)) - YPP(NVT(E));
BB(E) ← XPP(NVT(E)) - XPP(PVT(E));
CC(E) ← XPP(PVT(E))*YPP(NVT(E)) - XPP(NVT(E))*YPP(PVT(E));
TMP   ← SQRT(AA(E)↑2 + BB(E)↑2);
AA(E) ← AA(E)/TMP;  BB(E) ← BB(E)/TMP;  CC(E) ← CC(E)/TMP;
```

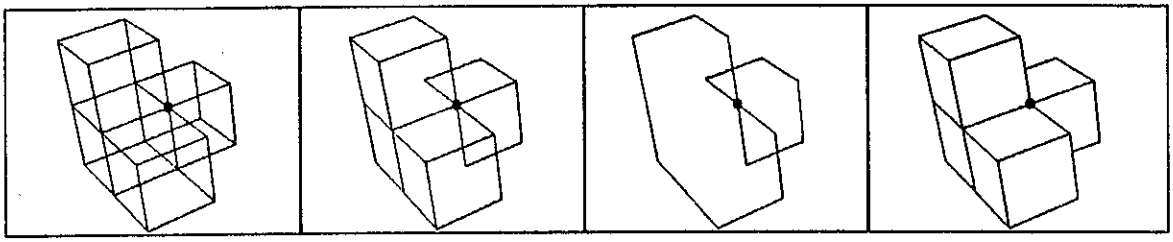
FIGURE 4.2 - FRONT FACES AND FOLDED EDGES.



After face, edge and vertex initialization two culls are applicable. The concave corner cull checks folded vertices of valence four or more for edges of the vertex that are hidden by a face of the same vertex; the corner marked by a heavy dot in Figure 4.3 is a concave corner with two folded

edges that are easily discovered to be hidden (i.e the end of the edge that is connected to the corner is hidden by a face of that corner). The second cull is applicable when hidden line elimination is being done on a sequence of images which are not changing very much from one picture to the next. By saving a pointer to the *overface* that covered each hidden vertex in the immediately preceding hidden line elimination, the previous overface can be quickly checked to see if it still covers the vertex. In the case of arm animation (example #2, Section 3.0) this form of exploiting *frame-coherence* realized a twenty-five percent savings in computation time (under timesharing, but with no other user programs).

FIGURE 4.3 - FRONT FACES AND FOLDS OF A CONCAVE CORNER.



Inspite of the complexity explained so far, still further measures could be taken to make the hidden line eliminator even faster, For example more 3-D clipping or spatial recusive cell sorting would allow the earlier elimination of objects that are out of sight.

4.2 Hide Marking a Coherent Object.

OCCULT marks the faces, edges and vertices of a polyhedral scene as being either visible or hidden with respect to a simulated camera. Edges that were at first partially visible are split into pieces so that each piece is either fully visible or fully hidden. All splits are undone and all OCCULT bits are cleared by a fixup routine named UNCULT. In a modeling environment that provides coherent polyhedra that can be easily traveled and modified, the simple technique of hide marking the neighbors of entities already hidden can be used to do almost all of the actual hiding, once a starting place has been found.

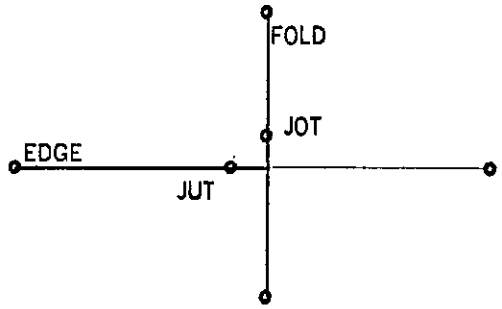
In OCCULT, the two innermost routines, EHIDE and VHIDE, perform this kind of marking and splitting. The routine VHIDE takes two arguments: the vertex, V, which is to be marked as hidden and the face, F, that is known to hide V; the routine then simply calls EHIDE for each potentially visible

edge of V's perimeter. EHIDE in turn takes three arguments: an overface, F, an edge, E, and one vertex, V, of that edge which is known to be hidden by F. EHIDE then checks to see whether or not E leaves its overface, F, there are three basic cases: (i) E does not leave F, so it is marked as hidden and VHIDE is applied to the vertex OTHER(E,V); (ii) E does leave overface F by crossing under a ¬FOLDED edge which provides a new overface for EHIDE to check; or (iii) E leaves F by crossing under a folded edge, so EHIDE splits the original edge, E, and the folded edge to form a T-joint (explained below) marking the hidden portion of E as hidden and leaving the remaining portion of E potentially visible.

A T-joint occurs in the image, when a folded edge hides a second edge that is further away from the camera. When OCCULT discovers a T-joint, both edges are ESPLIT and two new vertices are created the further one is called the JUT, Joint-Under-T, vertex the nearer one is called the JOT, Joint-Over-T, vertex. Juts and Jots point at each other using a temporary link field named TJOINT.

FIGURE 4.4 - T-JOINT DIAGRAM.

(The diagram is a view from slightly to the left and below the camera from which JOT and JUT appear coincident.)



There are several techniques for finding hidden starting places, the major techniques involve doing an edge-edge or a face-vertex compare using all the potentially visible faces, edges and vertices; the minor techniques include the concave corner cull and the hidden on last hide cull.

4.3 Edge-Edge and Face-Vertex Comparing.

In OCCULT, two particular compares stand out as most basic, the edge-edge compare and the face-vertex compare which are implemented in procedures named COMPEE and COMPFV, respectively.

The edge-edge compare routine, COMPEE, determines whether or not two edges intersect in the 2-D image coordinates, XPP and YPP. The basic edge-edge intersection test requires passing two opposition conditions: the ends of one edge must be in the opposite halfplane with respect to the line containing the other edge and vice versa. Halfplane opposition is checked by two evaluating the normal equation of the line using the edge coefficients AA, BB, CC and the vertex coordinates XPP and YPP. Consequently, it can be assumed that the two edges cross if the following expressions both return negative values:

```
FLAG1 ← (AA(E1)*XPP(PVT(E2)) + BB(E1)*YPP(PVT(E2)) + CC(E1))
    XOR (AA(E1)*XPP(NVT(E2)) + BB(E1)*YPP(NVT(E2)) + CC(E1));
FLAG2 ← (AA(E2)*XPP(PVT(E1)) + BB(E2)*YPP(PVT(E1)) + CC(E2))
    XOR (AA(E2)*XPP(NVT(E1)) + BB(E2)*YPP(NVT(E1)) + CC(E2));
```

The infix operator XOR (exclusive OR) is for toggling the sign bits in the same fashion as a multiplication would in more conventional ALGOL. When the crossing condition is true, the locus of intersection can be computed by solving two equations in two unknowns:

```
TMP     ←    (AA(E1)*BB(E2) - AA(E2)*BB(E1));
XPP(V)  ←    (CC(E1)*BB(E2) - CC(E2)*BB(E1))/TMP;
YPP(V)  ←    (AA(E1)*CC(E2) - AA(E2)*CC(E1))/TMP;
```

An alternate edge-edge compare method would be to solve the two equations in two unknowns first and then to see whether the intersection locus is interior to the line segments of both edges. Since, disjoint pairs of edges occur much more frequently than intersecting edges, the alternate method requires more floating arithmetic on the average than the first method which can discover about half of the disjoint cases by computing FLAG1. Furthermore the alternate method does not lend itself to distinguishing the almost touching cases which must be nudged to be disjoint. The OCCULT design depends on coercing edges to intersect at one unique point or not at all, the steps listed in Box 4.4 handle the special cases requiste to such a crossing discipline. The nudge is done in image coordinates, so the accuracy of world coordinates is maintained.

BOX 4.4 Edge-Edge Compare Steps.

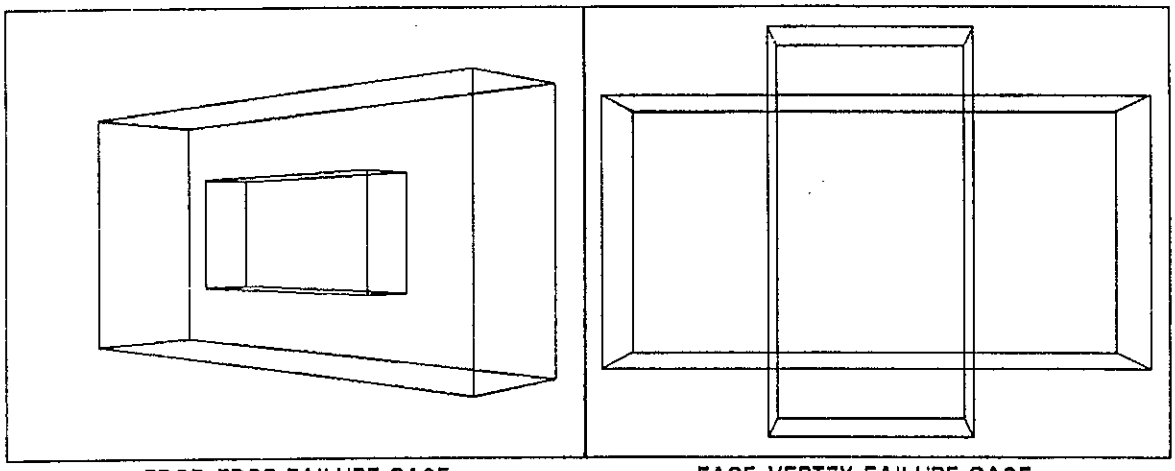
i. Test for Identity: same edge twice.
ii. Test for Topological connection: Edges with vertex or T-joint in common.
iii. Test for span Overlap in XPP and YPP: To prevent nasty collinear cases.
iv. Compare for crossing: Opposition Tests and Crossing Solver.
v. Nudge (Move off line, towards right and down).

The face-vertex compare routine, COMPFV has two parts: *Z-depth compare* for vertex under the plane of the face, and *2-D within compare* for vertex enclosure by the face perimeter. The first compare is done by evaluating the Z-depth of the vertex with respect to the plane of the face. The second compare tests whether the vertex falls outside of the face with respect to any of the edges of the face perimeter, since faces are convex and since polyhedra are oriented the properly directed edges coefficient are available. The Z-depth test is performed first because it is quicker.

Two very simple but important kinds of hidden line eliminators (that almost work) are based on combining edge-edge comparing or face-vertex comparing with coherent object hiding. In the edge-edge compare method all the edges (or even merely all the folded edges) of the image are compared with each other, N*(N-1)/2 compares, for crossings; when a crossing is found a T-joint is made and the hidden portion of the under edge is given to an EHIDE routine. In the face-vertex compare method all the vertices are compared with all the faces, (face count)*(vertex count) compares, for enclosure and covering; when a vertex is found hidden under and within a face it is given to a VHIDE routine. Together the EE-compare method and the FV-compare method form one slow but sure hidden line elimination algorithm; alone the EE-method fails to detect hidden objects with edges that don't intersect any edges of the occluding object as in the left panel of Figure 4.5 which shows two bricks of the same size but one behind the other. Likewise the FV-method fails to detect hidden objects in scenes where no vertex of the object is surround or covered by a face, right panel of Figure 4.5.

In OCCULT, the edge-edge compare is done after recursive windowing has isolated a reasonably small number of edges (twelve). A face-vertex compare is done only if any potentially visible vertices remain after all the other techniques have finished; in particular face-vertex comparing is only done when the case illustrated in the left panel of Figure 4.5 actually occurs and the set of faces that are used are only the faces that intersect the recursive window that contains the vertex.

FIGURE 4.5 - EE AND FV UNDETECTED HIDDEN OBJECT CASES.



EDGE-EDGE FAILURE CASE. FACE-VERTEX FAILURE CASE.

4.4 Recursive Windowing.

Recursive Windowing is a two dimensional spatial sorting technique for partitioning the faces, edges and vertices associated with a rectangular region called a window into two subwindows. The technique is applied recursively until a desired condition is achieved. The usual termination condition is that the population of entites in the window becomes sufficiently low or that the window becomes extremely small. The idea is implement in a routine called ESORT which resembles the hidden line eliminators of (Warnock 68) and (Sutherland 69). However ESORT is unique in that it maintains a data structure which allows edges to be split during the sort. The potentially nasty fixups are accomplished using a data structure that maintains a coherent image of both windows and edges. Metaphorically, the data structure is a cloth with a warp of windows and a woof of edges, where each warp thread is bound to a woof fiber by a bead.

Window Structure. The sort window itself is a twelve word node which contains data fields named XLO, XHI, YLO and YHI which specify the boundary of the window and data fields named PENCNT, SURCNT, EDGCNT and VCNT which specify the number of faces that penetrate the window, the number of faces that surround the window, the number of edges that pass through the window and the number of vertices that fall within the window, respectively. The window contains link fields to

hold pointers to the head of the pen-face list (penetrating faces), the sur-face list (surrounding faces), the vertex list, the head and tail the edge list and a pointer to its antecedent window.

Bead Structure. A bead is a two word node that contains four pointers and which represents one instance of an edge passing through a window. Each edge has a list of beads representing an ordered list of the windows through which it passes; and each window has a list of beads representing a list of the edges it contains. The link fields named WND and EDG of a bead, point to the particular window and the particular edge to which the bead belongs. The link fields named WNBL and EDBL of a bead contain the necessary links for the window's bead list and for the edge's bead list.

BOX 4.5 RECURSIVE WINDOWING ROUTINES.

1.	MKSWN	Make Sort Window.
2.	PSHSWN	Push Sort Window.
3.	PENSUR	Update penetrator and surrounder lists.
4.	POPSWN	Pop Sort Window.
5.	BLED	Bead List Edit.

The actual sort is composed of five routines (Box 4.5) which perform all the necessary creations and alterations to the window/edge/bead data structure. Initialization is done by the make sort window routine, MKSWN, which places all the potentially visible faces, edges and vertices into the first sort window along with the population counts and the extreme location of vertices in the positive and negative, XPP and YPP directions.

If the population counts of the window are too large, the pushdown sort windowing routine, PSHSWN, creates a new window node, places the node into the sort-window pushdown list, halves the original window's rectangle (spliting the longer sides) leaving the left (or upper) half of the rectangle in the old window node and allocating the right (or lower) half to the new window node. Next the vertex list is partitioned, each vertex falls into only one or the other window. Next the original window's bead list of penetrating edge is scanned, each edge must fall into one or the other or both windows. If an edge falls into both windows then a new bead is made and is placed in order into the bead list of the edge so that the beads of every edge indicate window penetrations in order from upper-left-most to lower-right-most. Finally PSHSWN applies PENSUR to each of the two windows.

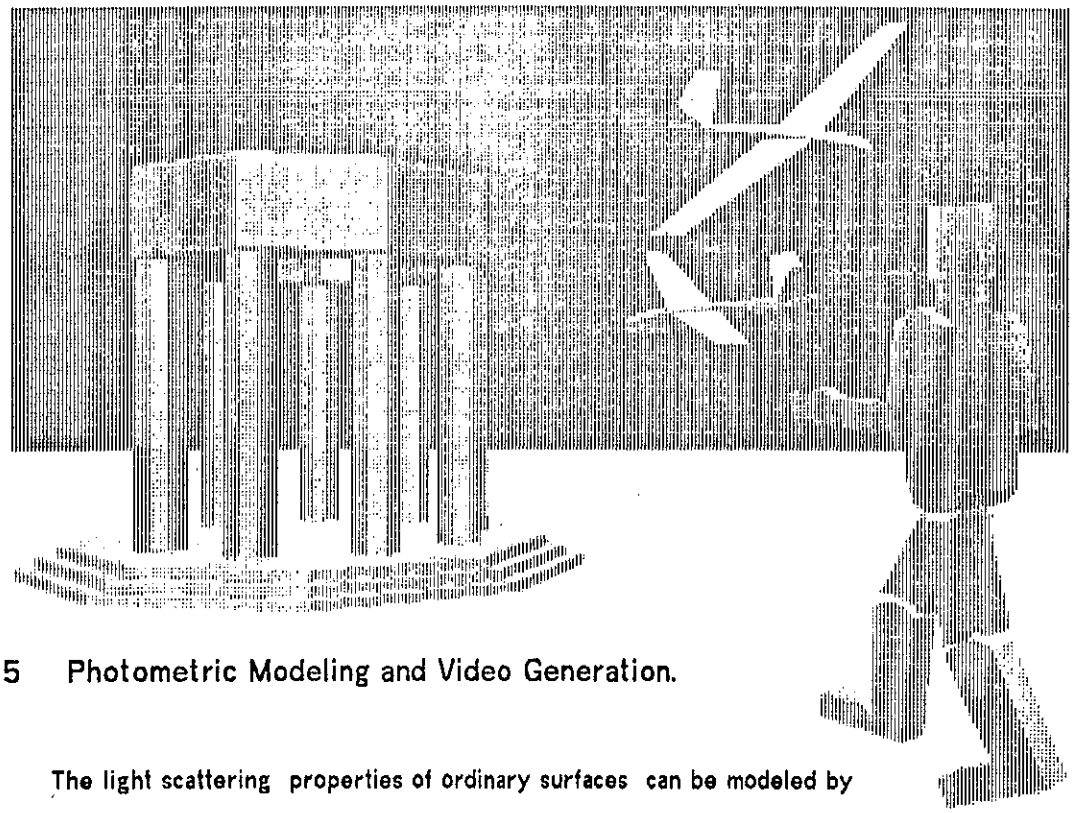
The penetrator and surrounder face routine, PENSUR, scans the new bead lists of penetrating edges of the two subwindows and marks the faces of those edges as penetrators and places them on the pen-list of the new window; next the routine scans the old penetrator list of the parent window and tests (and clears) the markings. Unmarked faces must be either surrounders or outsiders; the surrounders are placed in the sur-list of the new window.

If the populations of the window are sufficiently low the hidden line eliminator (or the body intersector, Chapter 5) processes the window (does the edge-edge compares) and calls the pop sort window routine, POPSWN. POPSWN zeroes the window field, WND, of beads of the window as an indication that the window is dead and so are its beads; dead beads are returned to free storage by the BLED routine explained below. Next the POPSWN scans the vertices or the window and places the pen-list and sur-list pointers of the window into temporary fields of each vertex; this trick preserves the results of the recursive window sort for the sake of possible face-vertex comparing. Finally the window node is popped off the pushdown window list and returned to free storage.

During both hidden line elimination and body intersection, edges are split in order to isolate the portion that is hidden or in order to create face piercing points. When an edge is split its bead list of windows is also split by means of the bead list edit routine, BLED. Since beads of an edge are ordered upper-left to lower-right; the BLED routine scans the beads for the window into which the newly created split vertex falls within; the vertex is then placed on that window's vertex list and a new bead is created (since both the old and the new edges must have beads in the window that contains the split) and the old bead list is split. Dead beads that are found while scanning the bead list are returned to free storage.

Although the link manipulations are complicated to recite, the essential point is that both windows and edges can be split without losing their topological connectedness, which gives one a tool for reducing an N-squared spatial computation into an N-log-N computation. The present implementation is coded in PDP-10 machine code, an ALGOL publication version will appear in a forthcoming technical report which is beyond the scope of this paper.

FIGURE 4.6 - EXAMPLE OF VIDEO SYNTHESIS.



4.5 Photometric Modeling and Video Generation.

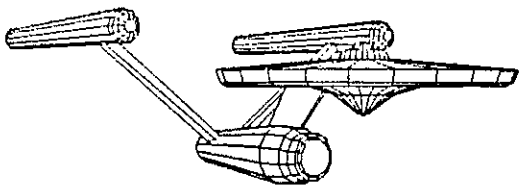
The light scattering properties of ordinary surfaces can be modeled by thinking of the surface as composed of many little mirrors. The orientation of each mirror is described by two angles, its tilt from the normal vector of the surface and its pan about the normal vector with respect to a specified reference vector in the tangent plane of the surface. For a perfect reflecting surface all the differential mirrors have a zero pan and tilt; for isotropic conventional surfaces the statistical distribution of pan orientations is flat and the distribution of tilt orientations is a blip function; and for a perfect isotropic Lambert surfaces both the pan and tilt distributions are flat.

After the visible faces have been assigned intensity values, a conversion from an OCCULT mosaic image to a raster image is done by an auxiliary program called MKVID, make video. MKVID resembles a Gouraud (71) and Watkins(70) hidden line eliminator in that it fills scan line by linear interpolation of segments between edges of the mosaic which are in their turn linear interpolations between vertices.

4.6 Performance of OCCULT and Related Work.

Ten hidden line elimination techniques were recently surveyed in (Sutherland, Sproull and Schumacker 1973), which after emphasing that hidden line elimination can be viewed as a sorting problem concluded with the remark that future implementations should be based on exploiting frame coherence, object coherence and combinations of the existing techniques. However the survey paper might be inadquate for a would-be implementer who should consult the textbook by (Sproull and Newman 73) for detailed explainations of the Warnock method and the Watkins method. Original research reports on hidden line elimators include: (Roberts 63), (Appel 67), (Warnock 68), (Warnock 69), (Watkins 70) and (Archuleta 72).

Inspite of all the activity and surveying of the literature, no quantitative commensurate study of the different methods has been attempted. In particular, the performance tables at the end of (Sutherland et al 1973) are subjective evaluations rather than experimental results of benchmark problems, as the authors clearly state. Continuing in the same subjective fashion, OCCULT is fast in that it can generate simple scenes (200 edges) of blocks in less than a second; the arm animation (524 edges) requires four to six seconds; the starship Enterprise (1230 edges) requires ten to twelve seconds; and the largest scenes that fit in core (4000 edges) take from thirty to sixty seconds.



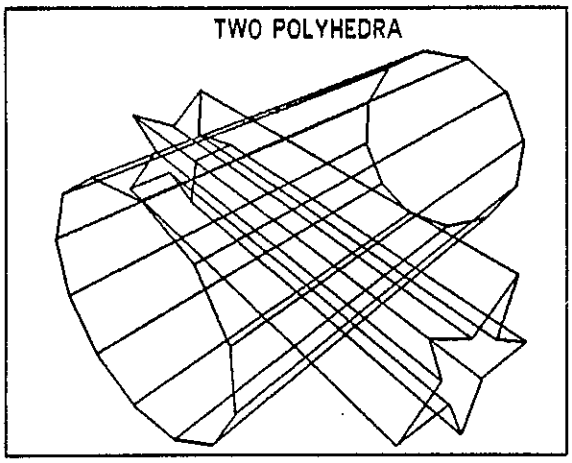
SECTION 5.

POLYHEDRON INTERSECTION.

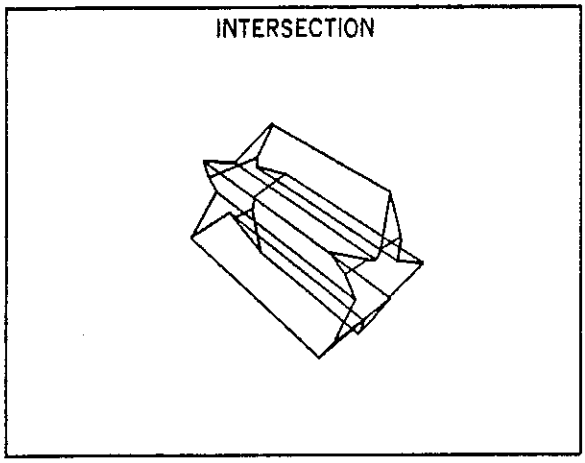
5.0 Introduction to Polyhedron Intersection.

The intersection, union, and set differences of two solid polyhedra can be computed by combining a body intersection procedure called BIN with the EVERT primitive, as Figure 5.1 illustrates. The body intersection procedure is important for three reasons: first, it is a general and conceptually elegant construction operator; second, it can be used for spatial modeling in collision detection and trajectory planning for manipulators and vehicles; and third, it can be used to localize an object in 3-D space from a sequence of silhouette views. The intersection algorithm consists of two parts: first, there is a geometric part in which all the faces and edges are compared with each other for potential face/edge intersections called piercing points; and second, there is a topological part in which the results are "copied off" of the given polyhedra; the results may consist of zero, one or many polyhedra. In the following, the term "operands" refers to the sets of polyhedra given to BIN as arguments and the term "result" refers to the set (possibly empty) of polyhedra produced by BIN.

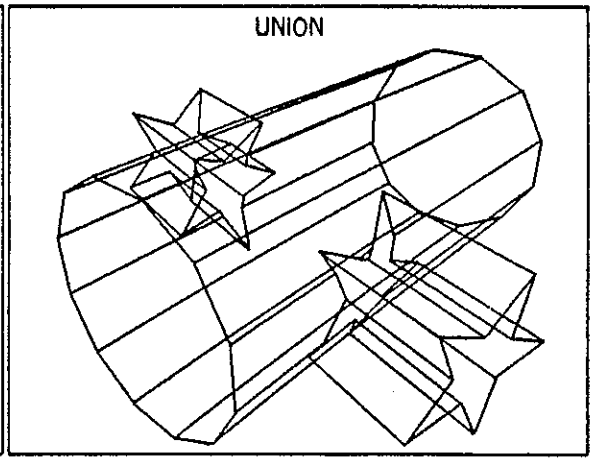
FIGURE 5.1 - POLYHEDRON INTERSECTION, UNION AND SUBTRACTION.



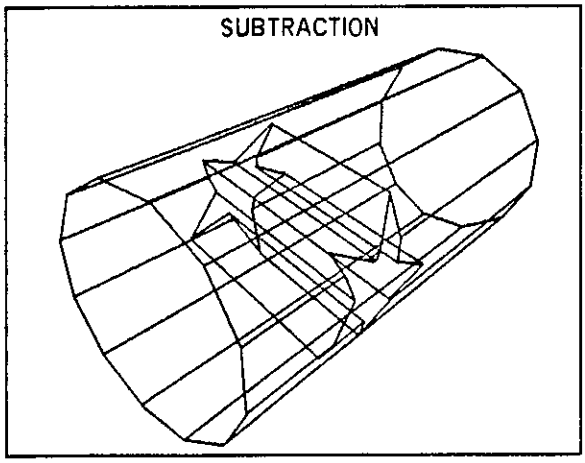
STAR AND CYLN



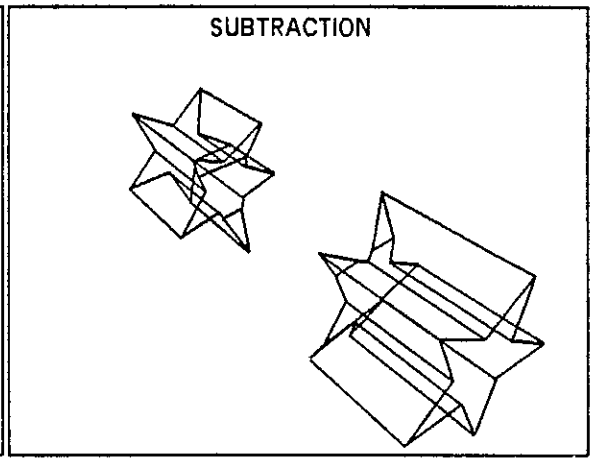
BIN(STAR,CYLN)



EVERT(BIN(EVERT(STAR),EVERT(CYLN)))



BIN(EVERT(STAR),CYLN)

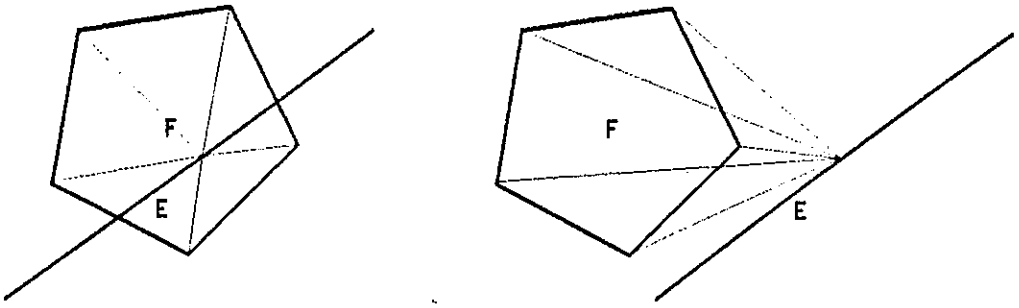


BIN(STAR,EVERT(CYLN))

5.1 Intersection Geometry.

Conceptually, the geometric part of the polyhedron intersection algorithm, BIN, consists of comparing each face of one operand with every edge of the other operand and vice versa. In practice the potentially N-squared compares are avoided by using the same recursive window partition sort that was used in the hidden line eliminator, OCCULT, Section 4.3. Ignoring the recursive windowing for a moment, the innermost face-edge compare of BIN consists of four steps: opposition, intersection, enclosure and fission.

FIGURE 5.2 - FACE PIERCING GEOMETRY.



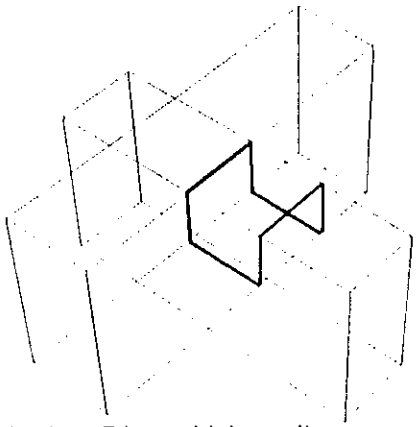
Piercing Point Within F. Piercing Point Outside F.

Opposition Test. Given a face F and an edge E, first, the endpoints of the edge are checked to see whether they are in opposite halfspaces with respect to the plane of the face. In terms of vector geometry, the dot product of the face vector and each vertex vector is taken and the signs compared; different signs indicate that the vertices are in different halfspaces. The opposition test requires six multiplications. *Intersection Locus*. The locus of the point where the edge pierces the plane of the face is computed (four multiplications). *Enclosure Test*. Next the edge is tested to see if it actually passes thru the interior of the face. In BIN, this test exploits the face convexity restriction. The test consists of crossing neighboring pairs of vectors radiating from the face-plane piercing-point to each vertex of the given face and testing for a sign change, Figure 5.2. Since only one component of the cross product needs to be evaluated, the test requires only two multiplications per edge of the face whoes plane is pierced. *Edge Fission*. If the edge pierces the face, then the edge is split (using the ESPLIT and BLED routines) forming a new vertex, called a face piercing vertex. A temporary link of the vertex node (field CW, left half of word 7) is set to point at the face that was pierced and the PED link of the new vertex is set to point at the one of its two edges that is external to the surface.

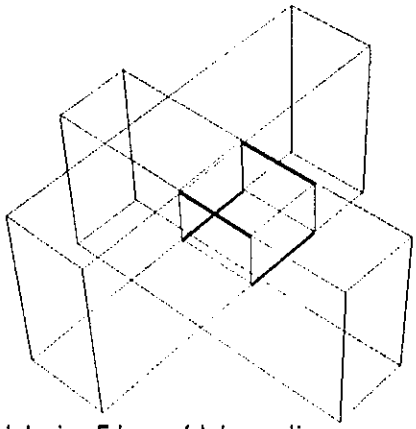
5.2 Intersection Topology.

After the face-piercing vertices have been made (assuming no pathological cases, Section 5.3), the edges and vertices of the result can be created in relation to the faces, edges, and vertices of the operands. The relation between the operands and the results is established in terms of two kinds of edges: interior edges and surface edges as illustrated in Figure 5.3. Surface edges correspond to the intersections of pairs of operand faces and interior edges correspond to edges of one operand that are enclosed inside the surface of the other operand. Surface edges always form connected loops. In Figure 5.3, two solid prisms are being intersected, on the left the surface edges of the intersection (a surface loop) is intensified in heavy lines, on the right the interior edges are intensified.

FIGURE 5.3 - THE SURFACE AND INTERIOR EDGES OF INTERSECTION.



Surface Edges of Intersection.



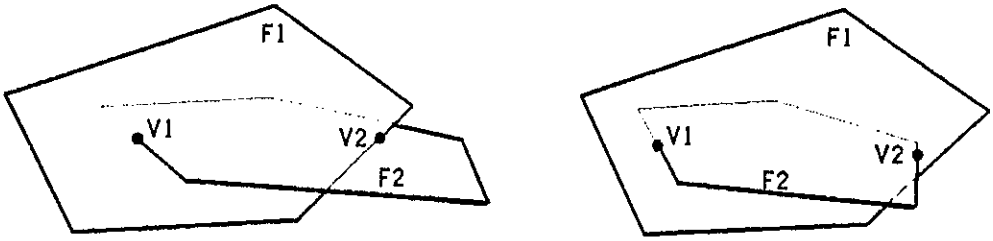
Interior Edges of Intersection.

In similar fashion there are surface vertices and interior vertices of the result. Each face-piercing vertex of the operands has a corresponding surface vertex in the result which is always a trihedral corner. The operand/result correspondence is maintained in a temporary link field named ALT; the alternate vertices and edges that belong to the result are created by two topological trace routines: the make surface, MKSURF routine, which creates surface edges and vertices of the result by tracing surface loops starting from an "un-ALTered" face piercing vertex. At each face-piercing vertex, MKSURF applies the ETRACE routine to the single interior edge of the trihedral corner. ETRACE creates edges and vertices interior to the result by tracing the edge graph bounded by face-piercing vertices. The new result edges are temporarily linked (PFACE and NFACE) to the old

operand faces. The MKSURF and ETRACE routines are followed by three steps that fix up the surface wings, interior wings and face nodes so that a complete winged edge polyhedral result is legally formed.

The interior trace routine is trivial - all the links are readily accessed using the ECCW and OTHER primitives on the operand polyhedra. The surface trace routine is made easy by implementing a procedure, NEXTPV, to retrieve the next face-piercing vertex about a surface loop. The NEXTPV procedure, given below, is based on the obseravtion that the intersection of two convex faces is one line segment and either one face is pierced twice by two different edges of the other face; or each face is pierced once by one edge of the other face, Figure 5.4.

FIGURE 5.4 - FETCH NEXT FACE-PIERCING VERTEX.



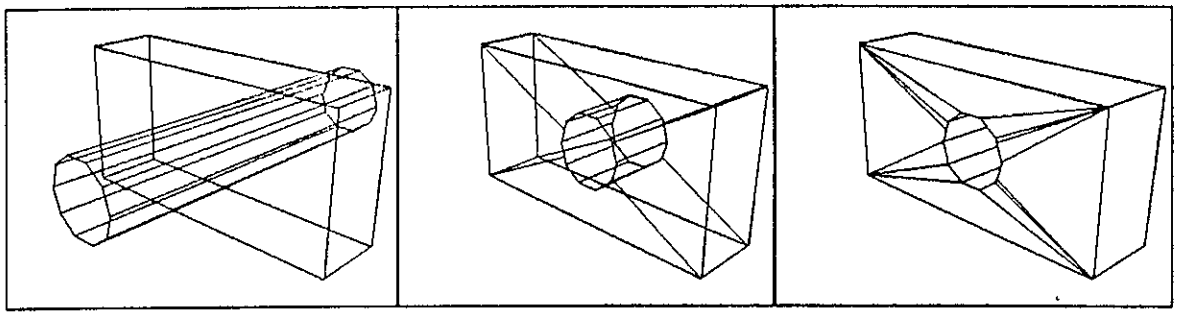
Edge of F1 pierces F2 at V2. Edge of F2 pierces F1 at V2.

```
COMMENT RETURN THE NEXT FACE PIERCING VEXT OF A SURFACE LOOP;
INTEGER PROCEDURE NEXTPV (INTEGER F2,V1);
BEGIN "NEXTPV"
        INTEGER F1,V2;
        F1 ← CW(V1);                    COMMENT FACE PIERCED BY V1;
COMMENT DOES AN EDGE OF F1 PIERCE F2 AT THE OTHER PIERCE-VERTEX V2;
        E ← E0 ← PED(F1);
        DO IF F2 = CW(V2←VCCW(E,F1)) THEN RETURN(V2) UNTIL E0 = (E←ECCW(E,F1));
COMMENT DOES AN EDGE OF F2 PIERCE F1 AT THE OTHER PIERCE-VERTEX V2;
        E ← E0 ← PED(F2);
        DO IF F1 = CW(V2←VCCW(E,F1)) ∧ V2≠V1 THEN RETURN(V2) UNTIL E0 = (E←ECCW(E,F2));
COMMENT FATAL CONSISTENCY ERROR - SOMETHING WRONG IN FACE/EDGE COMPARE PASS;
        RETURN(0);
END "NEXTPV";
```

Fix up step-1 places vertex and wing pointers in all the interior edges. An interior edge is distinguished by its non-zero ALT link. The new vertices are provided with a first edge, PED(VNEW), if it be lacking. Fix up step-2 wings together the surface vertex tridedral corners. Since by good luck

all surface vertices are necessarily trihedral, the edges can be passed to the WING primitive for oriented linking, in any order. The two surface wings of a surface vertex were stored in the NED and PED links by MKSURF; the inward wing can be retrieve as the PED(ALT(U)). Surface vertices are distinguished by their ALT vertex being marked as a piercing vertex. Fix up step-3 replaces the alien faces of the result with native faces. This is done by scanning the edge ring of the body, testing the two faces of each edge to see if they belong to the result body, and if a face doesn't belong it is replaced by a new one. Face replacement, as ususal, requires clocking around a face perimeter and changing the appropriate face link in each edge. A final marking trace assigns one body node to each separate connected graph of faces, edges and vertices.

FIGURE 5.5 - EXAMPLE OF A FACE HOLE FIXUP.



5.3 Special Cases of Intersection.

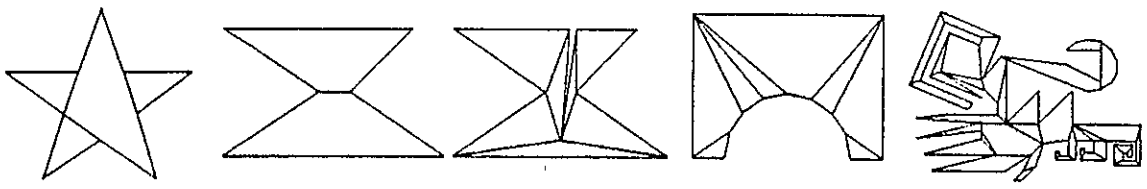
In order of difficulty from easy to hard, the four special cases that must be handled are non-intersection, extremely short edges, face holes and coincident entities. *Non-Intersection.* When the face-edge compare (by recursive window space sort) returns no piercing points, it implies that the surfaces of the given polyhedra do not intersect and that a further test is needed to determine whether the operands are disjoint (and so the intersection be empty) or whether one operand contains the other. *Face Holes.* Because EVERTed solids are allowed, one polyhedron can cut a hole in a face of the other without intersecting any of the edges of that face, which would fool the copy-trace. So as a preliminary step to BIN, all the surface loops are traced and checked to make certain they cross more than one face. If a one face surface-loop is found, the face is pyramided to a vertex interior to the surface-loop. A face hole fix up is illustrated in Figure 5.5, the middle panel of the figure shows

that two faces of the cubic prism were pyramided, the right panel of the figure shows the result after face-convexity coercion. *Short Edges*. An application of BIN can create edges with almost zero length, which require an extra pass to find and delete. *Coincident Entities*. An occasional edge that lies exactly in the plane of a face can be nudged off the plane a little resulting in extremely short edges which are later removed. Although it is meaningful to try to intersect polyhedra which have many faces, edges and vertices that are exactly coincident, the present implementation loses track of interior and exterior when too many nearly zero length edges are made.

5.4 Face Convexity Coercion.

Since, both the body intersecter, BIN, and the hidden line eliminator, OCCULT, are restricted to convex faced polyhdera; it is essential to have a routine that detects and subdivides the concave faces of a given polyhedron. The make convex routine, MKCNVX, reduces the concave faces of a body into reasonably small number of convex faces. The method consists of two steps: first, the face is broken down into triangles and second, the longest unnecessary newly made edges are removed. The reduction to triangles step is recursive: the pointiest extrema vertex of a face, V0, is lopped off, if no other vertices of the face are on the same side of the line segment between V0's immediate neighboring vertices: OTHER(ECCW(V0,F),V0) and OTHER(ECW(V0,F),V0). Otherwise the face is split, MKFE, using the vertex closest to V0 that violates V0's potential lop line. An extrema vertex is one that touchs the smallest circumscribed rectangle whose sides are parellel to the coordinate axes; the pointiest vertex is the one with the largest cosine.

FIGURE 5.6 - EXAMPLES OF FACE CONVEXITY COERCION.



5.5 Body Cutting.

Body cutting is the operation of dividing an arbitrary polyhedron into sets of parts above and

below a given cutting plane, as has already been illustrated in Figure 3.8. Although body cutting might be done by subtracting a very large thin rectangular prism, the process is sufficiently important to merit a separate implementation which nevertheless resembles the subtraction. First, all the edges of the given body are compared with the given cutting plane and piercing vertices are formed in pairs (one vertex for each side of the cut). Between the cutting-plane vertex-pairs are zero length edges which are placed into a special temporary list. Next, pairs of cutting-plane vertices (belonging to the same face and destined to be in the same half-space) are MKFEed together splitting the faces with cutting-plane edge pairs (one edge for each side of the cut). Between the cutting-plane edge-pairs are zero area faces. Finally all the zero length cutting plane edges are KLFEed if their PFACE and NFACE are different or UNGLUEed if their PFACE and NFACE are the same. In this circumstance an edge having the same NFACE and PFACE is a wasp edge. The simplicity of the body cutting implementation is do to the power of the UNGLUE Euler primitive.

5.6 Performance and Related Work.

Curious to relate, I have found no example in the literature of a general polyhedron intersection method. Maruyama's (72) method is a collision detector rather than a intersector, because he does not attempt to generate the polyhedra of intersection; however, his algorithm does resemble the geometric first phase of BIN and might have been extended for generating new solids. The intersection methods of Braid (73) are restricted to particular combinations of six volume elements which comprise a useful subset of cases for mechanical drawing.

The version of BIN is implemented on a PDP-10 (with 2 microsecond core memory) can construct the intersection of simple objects such as a pair of cubes in less than a quarter of a second; the intersection of a couple of twenty sided cylinders in about two seconds; the intersection of two horse silhouette cones takes (chapter 9) about fifteen seconds; and the intersection of two silhouette cone intersections can take up to a minute.

SECTION 6.

COMPUTER VISION THEORY.

6.0 Introduction to Computer Vision Theory.

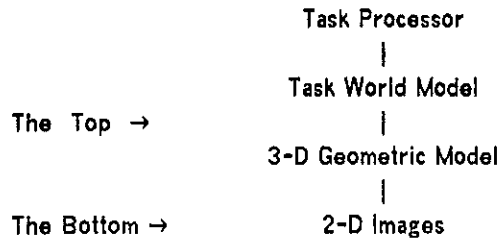
Computer vision concerns programming a computer to do a task that demands the use of an image forming light sensor such as a television camera. The theory I intend to elaborate is that general 3-D vision is a continuous process of keeping an internal visual simulator in sync with perceived images of the external reality, so that vision tasks can be done more by reference to the simulator's model and less by reference to the original images. The word *theory*, as used here, means simply a set of statements presenting a systematic view of a subject; specifically, I wish to exclude the connotation that the theory is a natural theory of vision. Perhaps there can be such a thing as an *artificial theory* which extends from the philosophy thru the design of an artifact.

6.1 A Geometric Feedback Vision System.

Vision systems mediate between images and world models; these two extremes of a vision system are called, in the jargon, the *bottom* and the *top* respectively. In what follows, the word *image* will be used to refer to the notion of a 2-D data structure representing a picture; a picture

being a rectangle taken from the pattern of light formed by a thin lens on the nearly flat photoelectric surface of a television camera's vidicon. On the other hand, a *world model* is a data structure which is supposed to represent the physical world for the purposes of a task processor. In particular, the main point of this thesis concerns isolating a portion of the world model (called the 3-D geometric world model) and placing it below most of the other entities that a task processor has to deal with. The vision hierarchy, so formed, is illustrated in box 6.1.

BOX 6.1 VISION SYSTEM HIERARCHY.



Between the top and the bottom, between images and the task world model, a general vision system has three distinguishable modes of operation: recognition, verification and description. Recognition vision can be characterized as bottom up, what is in the picture is determined by extracting a set of features from the image and by classifing them with respect to prejudices which must be taught. Verification vision is top down or model driven vision, and involves predicting an image followed by comparing the predicted image and a perceived image for differences which are expected but not yet measured. Descriptive vision is bottom up or data driven vision and involves converting the image into a representation that makes it possible (or easier) to do the desired vision task. I would like to call this third kind of vision "revelation vision" at times, although the phrase "descriptive vision" is the term used by most members of the computer vision community.

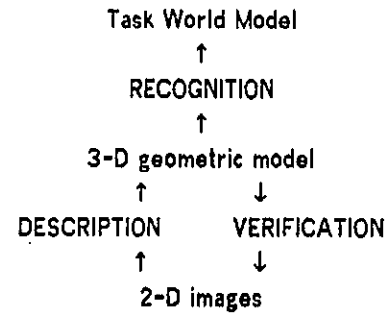
Box 6.2 THREE BASIC MODES OF VISION.

1. Recognition Vision - Feature Classification. (bottom up into a prejudiced top).
2. Verification Vision - Model Driven Vision. (nearly pure top down vision).
3. Descriptive Vision - Data Driven Vision. (nearly pure bottom up vision).

There are now enough concepts to outline a feedback system. By placing a 3-D geometric model between top and bottom; recognition vision can be done mapping 3-D (rather than 2-D) features

into the task world model with descriptive vision and verification vision linking the 2-D and 3-D models in a relatively dumb, mechanical fashion. Previous attempts to use recognition vision, to bridge directly the gap between 2-D images (of 3-D objects) and the task world model, have been frustrated because the characteristic 2-D image features of a 3-D object are very dependent on the 3-D physical processes of occultation, rotation and illumination. It is these processes that will have to be modeled and understood before the features relevant to the task processor can be deduced from the perceived images. The arrangement of these elements is diagramed below.

Box 6.3 BASIC FEEDBACK VISION SYSTEM DESIGN.



The lower part of the above diagram is the feedback loop of the 3-D geometric vision system. Depending on circumstances, the vision system may run almost entirely top-down (verification vision) or bottom-up (revelation vision). Verification vision is all that is required in a well known predictable environment; whereas, revelation vision is required in a brand new (tabula rasa) or rapidly changing environment. Thus revelation and verification form a loop, bottom-up and top-down. First, there is revelation that unprejudically builds a 3-D model; and second, the model is verified by testing image features predicted from the model. This loop like structure has been noted before by others; it is a form of what Tenenbaum (71) called *accommodation* and it is a form of what Falk (69) called *heuristic vision*; however I will go along with what I think is the current majority of vision workers who call it *feedback vision*.

Completing the design, the images and worlds are constructed, manipulated and compared by a variety of processors, the topmost of which is the task processor. Since the task processor is expected to vary with the application, it would be expedient if it could be isolated as a user program that calls

on utility routines of an appropriate vision sub-system. Immediately below the task processor are the 3-D recognition routines and the 3-D modeling routines. The modeling routines underlie most everything because they are used to create, alter and access the models.

Box 6.4 PROCESSORS OF A 3-D VISION SYSTEM.

0. The task processor.
1. 3-D recognition.
2. 3-D modeling routines.
3. Reality simulator.
4. Image analyser.
5. Image synthesizer.
6. Locus solvers.
7. Comparators: 2D and 3D.

The remaining processors include the reality simulator which does mechanics for modeling motion, collision and gravity. Also there are image analyzers, which do image enhancement and conversions such as converting video rasters into line drawings. There is an image synthesizer, which does hidden line and surface elimination, for verification by comparing synthetic images from the model with perceived images of reality. There are three kinds of locus solvers that compute numerical descriptions for cameras, light sources and physical objects. Finally, there is of course a large number of (at least ten) different compare processors for confirming or denying correspondences among entities in each of the different kinds of images and 3-D models.

6.2 Vision Tasks.

The 3-D vision research problem being discussed is that of finding out how to write programs that can see in the real world. Related vision problems include: modeling human perception, solving visual puzzles (non-real world), and developing advanced automation techniques (ad hoc vision). In order to approach the problem, specific programming tasks are proposed and solutions are sought, however a programming task is different than a reseach problem because many vision tasks can be done without vision. The vision solution to be found should be able to deal with real images, should include the continuity of the visual process in time and space, and should be more general purpose and less ad hoc. These three requirements (reality, continuity, and generality) will be developed by surveying six examples of computer vision tasks.

BOX 6.5 SIX EXAMPLES OF COMPUTER VISION TASKS.

Cart Related Tasks.
1. The Chauffeur Task.
2. The Explorer Task.
3. The Soldier Task.

Table Top Related Tasks.
4. Turntable Task.
5. The Blocks Task.
6. Machine Assembly Tasks.

First, there is the robot chauffeur task. In 1969, John McCarthy asked me to consider the vision requirements of a computer controlled car such as he depicted in an unpublished essay. The idea is that a user of such an automatic car would request a destination; the robot would select a route from an internally stored road map; and it would then proceed to its destination using visual data. The problem involves representing the road map in the computer and establishing the correspondence between the map and the appearance of the road as the automatic chauffeur drives the vehicle along the selected route. Lacking a computer controlled car, the problem was abstracted to that of tracing a route along the driveways and parking lots that surround the Stanford A.I. Laboratory using a television camera and transmitter mounted on a radio controlled electric cart. The robot chauffeur task could be solved by non-visual means such as by railroad like guidance or by inertial guidance; to preserve the vision aspect of the problem, no particular artifacts should be required along a route (landmarks must be found, not placed); and the extent of inertial dead reckoning should be noted.

Second, there is the task of a robot explorer. In (McCarthy 1964) there is a description of a robot for exploring Mars. The robot explorer was required to run for long periods of time without human intervention because the signal transmission time to Mars is as great as twenty minutes and because the 23.5 hour Martian day would place the vehicle out of Earth sight for twelve hours at a time. (This latter difficulty could be avoided at the expense of having a set of communication relay satellites in orbit around Mars.) The task of the explorer would be to drive around mapping the surface, looking for interesting features, and doing various experiments. To be prudent, a Mars explorer should be able to navigate without vision; this can be done by driving slowly and by using a tactile collision and crevasse detector. If the television system fails, the core samples and so on can still be collected at different Martian sites without unusual risk to the vehicle due to visual blindness.

The third vision task is that of the robot soldier, tank, sentry, pilot or policeman. The problem has several forms which are quite similar to the chauffeur and the explorer with the additional goal of doing something to coerce an opponent. Although this vision task has not yet been explicitly attempted at Stanford, to the best of my knowledge, the reader should be warned that a thorough solution to any of the other tasks almost assures the Orwellian technology to solve this one.

Fourth, the turntable task is to construct a 3-D model from a sequence of 2-D television images taken of an object rotated on a turntable. The turntable task was selected as a simplification of the explorer task and is an example of a nearly pure descriptive vision task.

Fifth, the classic blocks vision task consists of two parts: first convert a video image into a line drawing; second, make a selection from a set of predefined prototype models of blocks that accounts for the line drawing. In my opinion, this vision task emphasizes three pitfalls: single image vision, line drawings and blocks. The greatest pitfall, in the usual blocks vision task, is the presumption that a single image is to be solved; thus diverting attention away from the two most important depth perception mechanisms which are motion parallax and stereo parallax. The second pitfall is that the usual notion of a perspective line drawing is not a natural intermediate state; but is rather a very sophisticated and platonic geometric idea. The perfect line drawing lacks photometric information; even a line drawing with perfect shadow lines included will not resemble anything that can readily be gotten by processing real television pictures. Curiously, the lack of success in deriving line drawings from real television images of real blocks has not dampened interest in solving the second part of the problem. The perfect line drawing puzzle, was first worked on by Guzman (68) and extended to perfect shadows by Waltz (72); nevertheless, enough remains so that the puzzle will persist on its own merits, without being closely relevant to real world computer vision. Even assuming that imperfect line drawings are given, the blocks themselves, have lead such researchers as Falk (69) and Grape (73) to concentrate on vertex/edge classification schemes which have not been extended beyond the blocks domain. The blocks task could be rehabilitated by concentrating on photometric modeling and the use multiple images for depth perception.

Sixth, the Stanford Artificial Intelligence Laboratory has recently (1974) begun work on a National Science Foundation Grant supporting research in automatic machine assembly. In particular, effort will be directed to developing techniques that can be demonstrated by automatically assembling a chain saw gasoline engine. Two vision questions in such a machine assembly task are, where is the part and where is the hole; these questions will be initially handled by composing ad hoc part and hole detectors for each vision step required for the assembly.

The point of this task survey was to illustrate what is and is not a task requiring real 3-D vision; and to point out that caution has to be taken to preserve the vision aspects of a given task. In the usual course of vision projects, a single task or a single tool unfortunately dominates the research; my work is no exception, the one tool is 3-D modeling, and the task that dominated the formative stages of the research is that of the robot chauffeured cart. A better understanding of the ultimate nature of computer vision can be obtained by keeping the several tasks and the several tools in mind.

6.3 Vision System Design Arguments.

The physical information most directly relevant to vision is the location, extent and light scattering properties of solid opaque objects; the location, orientation and projection of the camera that takes the pictures; and the location and nature of the light that illuminates the world. The transformation rules of the everyday world that a programmer may assume, a priori, are the laws of physics. The arguments against geometric modeling divide into two categories: the reasonable and the intuitive. The reasonable arguments attack 3-D geometric modeling by comparing it to another modeling alternative, some of which are listed in Box 6.6. Actually, the domains of efficiency of the possible kinds of models do not greatly overlap; and an artificial intellect will have some portion of each kind. Nevertheless, I feel that 3-D geometric modeling is superior for the task at hand, and that the other models are less relevant to vision.

BOX 6.6 Alternatives to 3-D Geometric Modeling in a Vision System.

1. Image memory and with only the camera model in 3-D.
2. Statistical world model, e.g. Duda & Hart.
3. Procedural Knowledge, e.g. Hewitt & Winograd.
4. Semantic knowledge, e.g. Wilkes & Shank.
5. Formal Logic models, e.g McCarthy & Hayes.
6. Syntactic models.

Perhaps the best alternative to a 3-D geometric model is to have a library of little 2-D images describing the appearance of various 3-D loci from given directions. The advantage would be that a sophisticated image predictor would not be required; on the other hand the image library is potentially quite large and that even with a huge data base new views and lighting of familiar objects and scenes cannot be anticipated. A second alternative is the statistical world model used in the pattern recognition paradigm. Such modeling might be added to the geometric model; however, alone the statistical abstraction of world features in the presence of occultation, rotation and illumination seems as hopeless as the abstraction of a man's personality from the pattern of tea leaves in his cup.

Procedural knowledge models represent the world in terms of routines (or actors) which either know or can compute the answer to a question about the world. Semantic models represent the world in term of a data structure of conceptual statements; and formal logic models represent the world in terms of first order predicate calculus or in terms of a situation calculus. The procedural, semantic and formal logic world models are all general enough to represent a vision model and in a theoretical sense they are merely other notations for 3-D geometric modeling. However in practice, these three modeling regimes are not efficient holders and handlers of quantitative geometric data; but are rather intended for a higher level of abstract reasoning. Another alleged advantage of these higher models is that they can represent partial knowledge and uncertainty, which in a geometric model is implicit, in that structures are missing or incomplete. For example, McCarthy and Feldman demand that when a robot has only seen the front of an office desk that it should be able to draw inferences from its model about the back of the desk; I feel that this so called advantage is not required by the problem and that basic visual modeling is on a more agnostic level.

The syntactical approach to descriptive vision is that an image is a sentence of a picture grammar and that consequently the image description should be given in terms of a sequence of grammar transformations rules. Again this paradigm is valid in principle but impractical for real images of 3-D objects because simple replacement rules cannot readily express rotation, perspective, and photometric transformations. On the other hand, the syntactical model has been used to describe perfect line drawings of 3-D objects, (Gips 74).

The intuitive arguments include the opinions that geometric modeling is too numerical, too exact, or too non-human to be relevant for computer vision research. Against such intuitions, I wish to pose two fallacies. First, there is the natural mimicry fallacy, which is that it is false to insist that a machine must mimic nature in order to achieve its design goals. Boeing 747's are not covered with feathers; trucks do not have legs; and computer vision need not simulate human vision. The advocates of the uniqueness of natural intelligence and perception will have to come up with a rather unusual uniqueness proof to establish their conjecture. In the meantime, one should be open minded about the potential forms a perceptive consciousness can take.

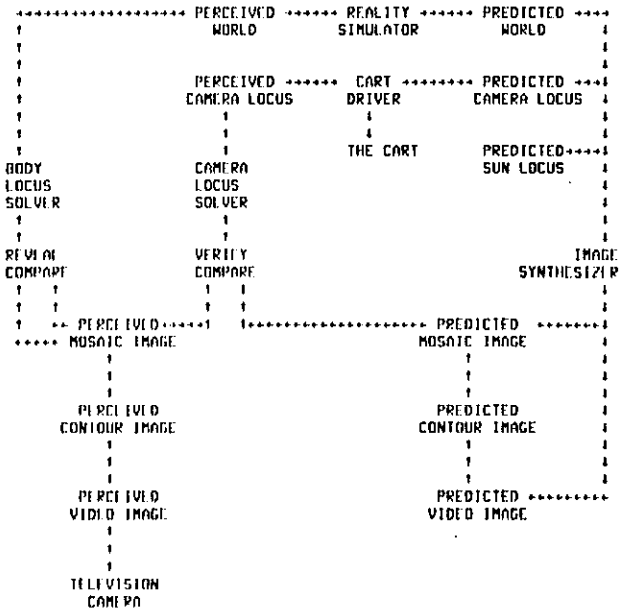
Second, there is the self introspection fallacy, which is that it is false to insist that one's introspections about how he thinks and sees are direct observations of thought and sight. By introspection some conclude that the visual models (even on a low level) are essentially qualitative rather than quantitative. My belief is that the vision processing of the brain is quite quantitative and only passes into qualities at a higher level of processing. In either case, the exact details of human visual processing are inaccessible to conscious self introspection.

Although describing the above two fallacies might soften a person's prejudice against numerical geometric modeling, some important argument or idea is missing that would be convincing short of the final achievement of computer vision. Contrariwise, I have not heard an argument that would change my prejudice in favor of such models. Nevertheless beyond prejudice, my theory would be proved wrong if a really powerful computer vision system is ever built without using any geometric models worth speaking of, perhaps by employing an elaborate stimulus response paradigm.

6.4 Mobile Robot Vision.

The elements discussed so far will now be brought together into a system design for performing mobile robot vision. The proposed system is illustrated below in the block diagram in Box 6.7. (The diagram is called a mandala in that a *mandala* is any circle-like system diagram). Although, the robot chauffeured cart was the main task theme for this research; I have failed to date, August 1974, to achieve the hardware and software required to drive the cart around the laboratory under its own control. Nevertheless, this necessarily theoretical cart system has been of considerable use in developing the visual 3-D modeling routines and theory, which are the subject of this thesis.

BOX 6.7 CART VISION MANDALA.



The robot chauffeur task involves establishing the correspondence between an internal road map and the appearance of the road in order to steer a vehicle along a predefined path. For a first cut, the planned route is assumed to be clear, and the cart and the sun are assumed to be the only movable things in a static world. Dealing with moving obstacles is a second problem, motion thru a static world must be dealt with first.

The cart at the Stanford Artificial Intelligence Laboratory is intended for outdoors use and consists of a piece of plywood, four bicycle wheels, six electric motors, two car batteries, a television camera, a television transmitter, a box of digital logic, a box of relays, and a toy airplane radio receiver. (The vehicle being discussed is not "Shaky", which belongs to the Stanford Research Institute's Artificial Intelligence Group. There are two A.I. labs near Stanford and each has a computer controlled vehicle.) The six possible cart actions are: run forwards, run backwards, steer to the left, steer to the right, pan camera to the left, pan camera to the right. Other than the television camera, there is no telemetry concerning the state of the cart or its immediate environment.

BOX 6.8 A POSSIBLE CART TASK SOLUTION.

1. Predict (or retrieve) 2-D image features.
2. Perceive (take) a television picture and convert into features.
3. Compare (verify) predicted and perceived features.
4. Solve for camera locus.
5. Servo the cart along its intended course.

The solution to the cart problem, begins with the cart at a known starting position with a road map of visual landmarks with known loci. That is, the upper leftmost two rectangles of the cart mandala are initialized so that the perceived cart locus and the perceived world correspond with reality. Flowing across the top of the mandala, the cart driver, blindly moves the cart forward along the desired route by dead reckoning (say the cart moves five feet and stops) and the driver updates the predicted cart locus. The reality simulator is an identity in this simple case because the world is assumed static. Next the image synthesizer uses the predicted world, camera and sun to compute a predicted image containing the landmark features expected to be in view. Now, in the lower left of the mandala, the cart's television camera takes a perceived picture and (flowing upwards) the picture is converted into a form suitable for comparing and matching with the predicted image. Features that are both predicted and perceived and found to match are used by the camera locus solver to compute a new perceived camera locus (from which the cart locus can be deduced). Finally the cart driver compares the perceived and the predicted cart locus and corrects its course and moves the cart again, and so on.

The remaining limb of the cart mandala is invoked in order to turn the chauffeur into an explorer. Perceived images are compared in time by the reveal compare and new features are located by the body locus solver and placed into the world model. The generality and feasibility of such a cart system depends almost entirely on the representation of the world and the representation of image features. (The more general, the less feasible). Four smaller cart systems might be possible using simpler 3-D models.

A first system might consist of a road map, a road model, a road model generator, a solar ephemeris, an image predictor an image comparator, a camera locus solver, and a course servo routine. The roadways and nearby environs are entered into the computer. In fact, real roadways are constructed from a two dimensional (X,Y) allignment map showing where the center of the road goes as a curve composed of line segment and circular arcs; and from a two dimensional (S,Z) elevation diagram, showing the height of the road above sea level as a function of distance along the road in a sequence of linear grades and vertical arcs which (not too surprising) are nearly cubic splines. A second version, might be made like the first except that the road model, road model generator, and image predictor are replaced by a library of road images. In this system the robot vehicle is trained by being driven down the roads it is suppose to follow. A third system also might be made like the first except that the road map is not initially given, and indeed the road is no longer presumed to exist. Part of the problem becomes finding a road, a road in the sense of a clear area; this version yields the cart explorer and if the clear area is found quite rapidly and the world is updated quite frequently, the explorer can be a chauffeur that can handle obstacles and moving objects.

6.5 Summary and Related Vision Work.

To recapitulate, three vision system design requirements were postulated: reality, generality, and continuity. These requiremonts were illustrated by discussing a number of vision related tasks. Next, a vision system was described as mediating between 2-D images and a world model; with the world model being further broken down into a 3-D geometric model and a task world model. Between these entities three basic vision modes were identified: recognition, verification and revelation (description). Finally, the general purpose vision system was depicted as a quantitative and description

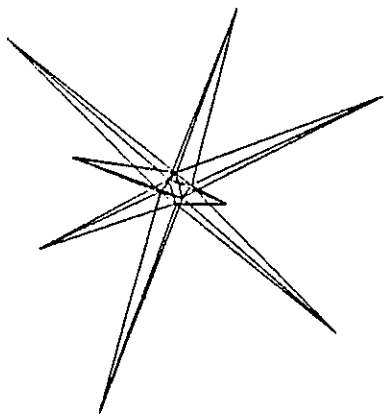
oriented feedback cycle which maintain a 3-D geometric model for the sake of higher qualitative, symbolic, and recognition oriented task processors. Approaching the vision system in greater detail; the role of seven (or so) essential kinds of processors were explained: the task processor, 3-D modeling routines, reality simulator, image analyser, image synthesizer, comparators, and locus solvers. The processors and data types were assembled into a cart chauffeur system.

Larry Roberts is justly credited for doing the seminal work in 3-D Computer Vision; although his thesis (Roberts 63) appeared over ten years ago the subject has languished dependent on and overshadowed by the four areas called: Image Processing, Pattern Recognition, Computer Graphics, and Artificial Intelligence. Outside the computer sciences, workers in psychology, neurology and philosophy also seek a theory of vision.

Image Processing involves the study and development of programs that enhance, transform and compare 2-D images. Nearly all image processing work can eventually be applied to computer vision in various circumstances. A survey of this field can be found in an article by Rosenfeld(69). Image Pattern Recognition involves two steps: feature extraction and classification. A comprehensive text about this field with respect to computer vision, has been written by (Duda and Hart 73). Computer Graphics is the inverse of descriptive computer vision. The problem of computer graphics is to synthesis images from three dimensional models; the problem of descriptive computer vision is to analyze images into three dimensional models. An introductory text book about this field would be that of (Newman and Sproull 73). Finally, there is Artificial Intelligence, which in my opinion is an institution sheltering a heterogenous group of embryonic computer subjects; the biggest of the present day orphans include: robotics, natural language, theorem proving, speech analysis, vision and planning. A more narrow and relevant definition of artificial intelligence is that it concerns the programming of the robot task processor which sits above the vision system.

The related vision work of specific individuals has already been mention in context. To summarize, the present vision work is related to the early work of Roberts(63) and Sutherland; to recent work at Stanford: Falk, Feldman and Paul(67), Tenenbaum(72), Agin(72), Grape(73); to the work at MIT: Guzman, Horn, Waltz, Krakaurer; to the work at the University of Utah: Warnock, Watkins;

and to work at other places: SRI and JPL. Future progress in computer vision will proceed in step with better computer hardware, better computer graphics software, and better world modeling software. Further vision work at Stanford, which is related to the present theory is being done by Lynn Quam and Hans Morevac. The machine assembly task is being pursued both by the Artificial Intelligence Group of the Stanford Research Institute and by the Hand Eye Project at Stanford University. Because the demand for doing practical vision tasks can be satisfied with existing ad hoc methods or by not using a visual sensor at all; little or no theoretical vision progress will necessarily result from the achievement of spectacular robotic industrial assembly demonstations (hire the handicap, blind robots assembles widgets). On the other hand, since the missing ingredient for computer vision is the spatial modeling to which perceive images can be related; I believe that the development of the technology for generating commercial film and television by computer for entertainment might make significant contribution to computer vision.



SECTION 7.

VIDEO IMAGE CONTOURING.

7.0 Introduction to Image Analysis.

Simply put, image analysis is the inverse of image synthesis. From this point of view, the usually difficult question of "analysis into what ?" is answered by the answer to the question "synthesis from what ?". Since a 3-D geometric model is adequate (and necessary) for synthesizing digital television pictures, it is reasonable to suppose that such a model is an appropriate subgoal in the analysis of television pictures. Such an analysis into a 3-D model would provide a useful data reduction as well as a convenient representation for solving robotics problems such as manipulation, navigation and recognition. This approach to image analysis is somewhat heretical, the orthodox method is to extract features from 2-D images, which features are then used directly for the desired task. On the other hand, vision by inverse computer graphics may be viewed as an extreme form of feature finding, involving the extraction of a set of basic geometric features which are combined to form a superfeature, a 3-D model. The rest of this introduction enumerates some of the kinds of information available in a sequence of images and some of the kinds of data structures for representing images. An image is a 2-D data structure representing the contents of a rectangle from the pattern of light formed by a thin lens; a sequence of images in time is called a film.

Three basic kinds of information in an image are photometric information, geometric information, and topological information. Fundamentally, geometry concerns distance measure. The geometry of an image is based on coordinate pairs that are associated with the elements that form the image. From the coordinates such geometric properties as length, area, angle and moments can be computed. Photometry means light measure, although physical measurements of light may include power, hue, saturation, polarization and phase; only the relative power between points of the same image is easily available to a computer using a television camera. The taking of color images is possible at Stanford by means of filters; however, the acquisition of color is inconvenient and has not been seriously pursued in the present work. Finally, topology has to do with neighborhoods, what is next to what; topological data may be explicitly represented by pointers between related entities, or implicitly represented by the format of the data.

Three basic kinds of image data structures are the raster, the contour map and the mosaic. A raster image is a two dimensional integer valued array of pixels; a pixel "picture element", is a single sample position on a vidicon. Although the real shape of a pixel is probably that of a blunt ellipse; the fiction that pixels tesselate the image into little rectangles will be adopted. For other theoretical purposes the array is assumed to be formed by sampling and truncating a two dimensional, smooth, infinitely differentiable real valued function. A contour image is like a geodesic contour map, no two contours ever cross and all the contours close. A mosaic image (or tesselation) is like a ceramic tile mosaic, no two regions ever overlap and the whole image is completely covered with tiles. Further useful restrictions might be made concerning whether it is permitted to have tiles with holes surrounding smaller tiles or whether it is permitted for a tile to have points that are thinner than a single pixel.

Given a raster image, the usual visual analysis approach is to find the features. One canonical geometric image feature is called the *edge* and the places where edges are not found are called *regions*. For a naive start, an edge can be defined as a locus of change in the image function. Edges and regions are complementary sides of the same slippery concept; the concept is slippery because although a continuous function of two variables and a graph of edges are each well known mathematical

objects the conversion of one into the other is a poorly understood process that depends greatly on ones motives and resources. A computational definition of the region/edge notion would include a procedure for converting a raster approximation of an image function into a region/edge representation based on parameters which are conceptually elegant.

7.1 CRE - An Image Processing Sub-System.

The acronym CRE stands for "Contour, Region, Edge". CRE is a solution to the problem of finding contour edges in a sequence of television pictures and of linking corresponding edges and polygons from one picture to the next. The process is automatic and is intended to run without human intervention. Furthermore, the process is bottom up; there are no inputs that anticipate the content of the given television images. The output of CRE is a 2-D contour map data structure which is suitable input to the 3-D modeling program, GEOMED. Five design choices that determine the character of CRE are listed in Box 7.1. The design choices are ordered from the more strategic to the more tactical; the first three choices being research strategies, the latter two choices being programming tactics. Adopting these design choices lead to image contouring and contour map structures similar to those of Krakauer (71) and Zahn (66).

BOX 7.1 CRE DESIGN CHOICES

1. Dumb vision rather than model driven vision.
2. Multi image analysis rather than single image analysis.
3. Total image structure imposed on edge finding; rather than separate edge finder and image analyzer.
4. Automatic rather than interactive.
5. Machine language rather than higher level language.

The first design choice does not refer to the issue of how model dependent a finished general vision system will be (it will be quite model dependent), but rather to the issue of how one should begin building such a system. The best starting points are at the two apparent extremes of nearly total knowledge of a particular visual world or nearly total ignorance. The first extreme involves synthesis (by computer graphics) of a predicted 2-D image, followed by comparing the predicted and a perceived image for slight differences which are expected but not yet measured. The second extreme involves analyzing perceived images into structures which can be readily compared for near equality

and measured for slight differences; followed by the construction of a 3-D geometric model of the perceived world. The point is that in both cases images are compared, and in both cases the 3-D model initially (or finally) contains specific numerical data on the geometry and physics of the particular world being looked at.

The second design choice, of multi image analysis rather than single image analysis, provides a basis for solving for camera positions and feature depths. The third design choice solves (or rather avoids) the problem of integrating an edge finder's results into an image. By using a very simple edge finder, and by accepting all the edges found, the image structure is never lost. This design postpones the problem of interpreting photometric edges as physical edges. The fourth choice is a resolution to write an image processor that does not require operator assistance or manual parameter tuning. The final design choice of using machine language was for the sake of implementing node link data structures that are processed one hundred times faster than LEAP, ten times faster than compiled LISP and that require significantly less memory than similar structures in either LISP or LEAP. Furthermore machine code assembles and loads faster than higher level languages; and machine code can be extensively fixed and altered without recompiling.

It is my impression that CRE itself does not raise any really new scientific problems; nor does it have any really new solutions to the old problems; rather CRE is another competent video region edge finding program with its own set of tricks. However, it is further my impression that the particular tricks for nesting and comparing polygons in CRE are original programming techniques. As a part of the larger feedback system, CRE is a necessary, but not entirely satisfactory implementation of pure bottom up image analysis.

CRE consists of five steps: thresholding, contouring, nesting, smoothing and comparing. Thresholding, contouring and smoothing perform conversions between two different kinds of images. Nesting and contouring compute topological relationships within a given image representation. In summary the major operations and operands are as listed in Box 7.2; the VIC-Images are Video Intesity Contour Images and the ARC-images are contours that have been smoothed.

BOX 7.2 CRE DATA TRANSFORMATIONS.

MAJOR OPERATION	OPERAND	RESULT.
1. THRESHOLDING:	6-BIT-IMAGE,	1-BIT-IMAGES.
2. CONTOURING:	1-BIT-IMAGES,	VIC-IMAGE.
3. NESTING:	VIC-IMAGE,	NESTED-VIC-IMAGE.
4. SMOOTHING:	VIC-IMAGE,	ARC-IMAGE.
5. COMPARING:	IMAGE & FILM,	FILM.

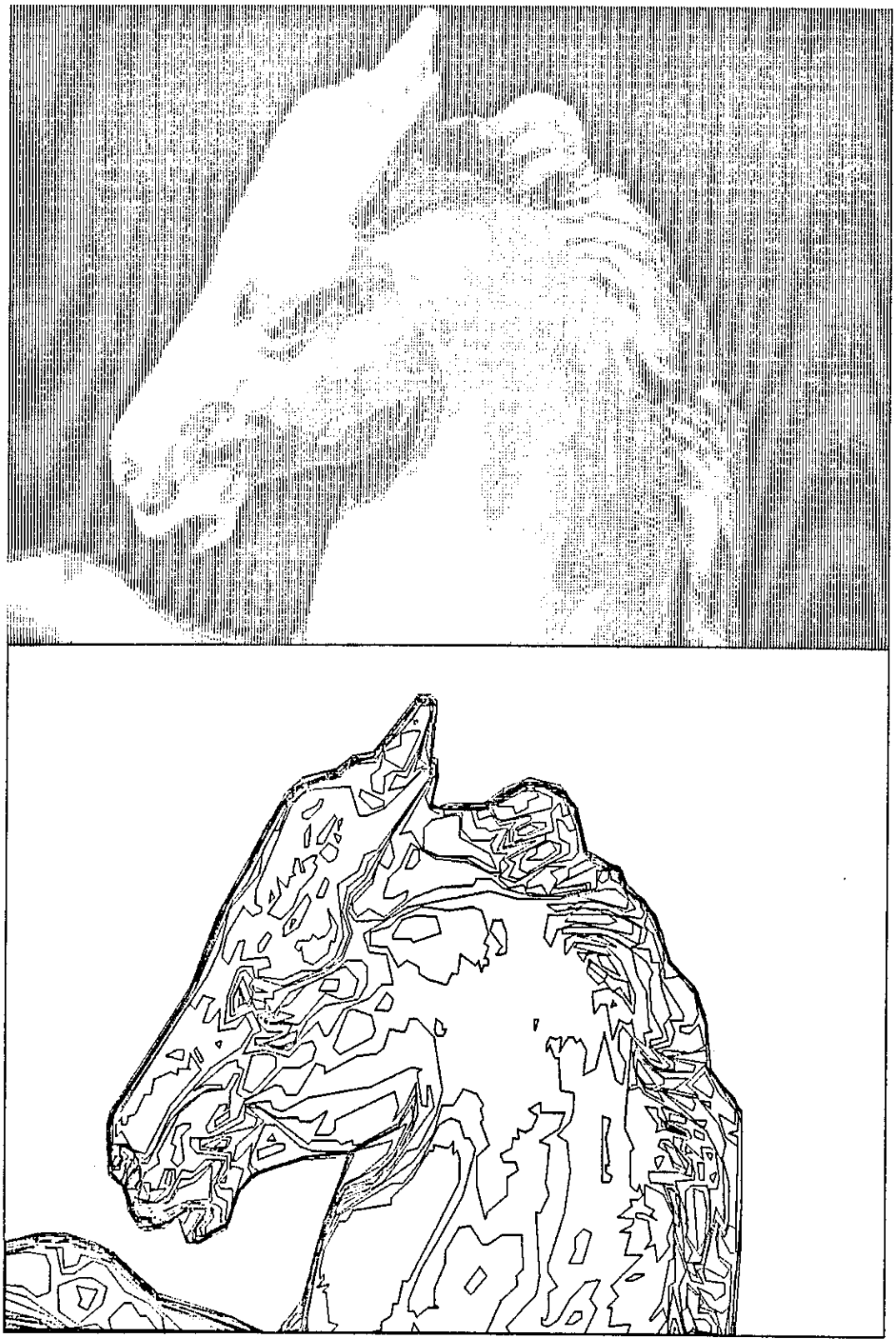
The initial operand is a 6-bit video raster, which in the present implementation is coerced into a window of 216 row by 288 columns; intermediate operands consist of 1-bit rasters named PAC, VSEG and HSEG which are explained below, as well as a raster of links named SKY which is used to perform the polygon nesting. The magic window size 216 by 288 was derive by considering the largest product of powers of two and three that would fit within a video image. The final result is a node/link structure composed of several kinds of nodes: vectors, arcs, polygons, lamtens (lamina inertia tensors) levels, images and the film node.

Although the natural order of operations is sequential from image thresholding to image comparing; in order to keep memory size down, the first four steps are applied one intensity level at a time from the darkest cut to the lightest cut (only nesting depends on this sequential cut order); and comparing is applied to whole images. Figure 7.1 illustrates an initial video image and its corresponding contour image. The contoured image consists of thirteen intensity levels and took 45 seconds to compute (on a PDP-10, two microsecond memory). The final CRE data structure was composed of 1996 nodes.

7.2 Thresholding.

Thresholding, the first and easiest step of CRE, consists of two subroutines, called THRESH and PACXOR. THRESH converts a 6-bit image into a 1-bit image with respect to a given threshold cut level between zero for black and sixty-three for light. All pixels equal to or greater than the cut, map into a one; all the pixels less than the cut, map into zero. The resulting 1-bit image is stored in a bit array of 216 rows by 288 columns (1728 words, 36 bits per word) called the PAC (picture accumulator) which was named in memory of McCormick's ILLIAC-III. After THRESH, the PAC contains blobs of bits.

FIGURE 7.1 - VIDEO IMAGE AND CONTOUR IMAGE.



A blob is defined as "rook's move" connected; that is every bit of a blob can be reached by horizontal or vertical moves from any other bit without having to cross a zero bit or having to make a diagonal (bishop's) move. Blobs may of course have holes. Or equivalently a blob always has one outer perimeter polygon, and may have one, several or no inner perimeter polygons. This blob and hole topology is recoverable from the CRE data structure and is built by the nesting step.

Next, PACXOR copies the PAC into two slightly larger bit arrays named HSEG and VSEG. Then the PAC is shifted down one row and exclusive ORed into the HSEG array; and the PAC is shifted right one column and exclusive ORed into the VSEG array to compute the horizontal and vertical border bits of the PAC blobs. Notice, that technically this is the very heart of the edge finder of CRE; namely, PACXOR is the mechanism that converts regions into edges. Edge tracing is the only operation CRE performs on the 1-bit rasters; although Boolean image processing has caught the eye of many vision programmers (perhaps because it resembles an array automata or the game Life) one rapidly discovers that raster operations alone are too weak to do anything interesting that can't already be done better analytically in a raster of numbers or topologically in a node/link data structure.

7.3 Contouring.

Contouring, converts the bit arrays HSEG and VSEG into vectors and polygons. The contouring itself, is done by a single subroutine named MKPGON, make polygon. When MKPGON is called, it looks for the upper most left non-zero bit in the VSEG array. If the VSEG array is empty, MKPGON returns a NIL. However, when the bit is found, MKPGON traces and erases the polygonal outline to which that bit belongs and returns a polygon node with a ring of vectors. The MKPGON trace can go in four directions: north and south along vertical columns of bits in the VSEG array, or east and west along horizontal rows of the HSEG array. The trace starts by heading south until it hits a turn; while heading south MKPGON must check for first a turn to the east (indicated by a bit in HSEG); next for no turn (continue south); and last for a turn to the west. When a turn is encountered MKPGON creates a vector node representing the run of bits between the previous turn and the present turn. The trace always ends heading west bound. The outline so traced can be either the edge of a blob or a hole, the two cases are distinguished by looking at the VIC-polygon's uppermost left pixel in the PAC bit array.

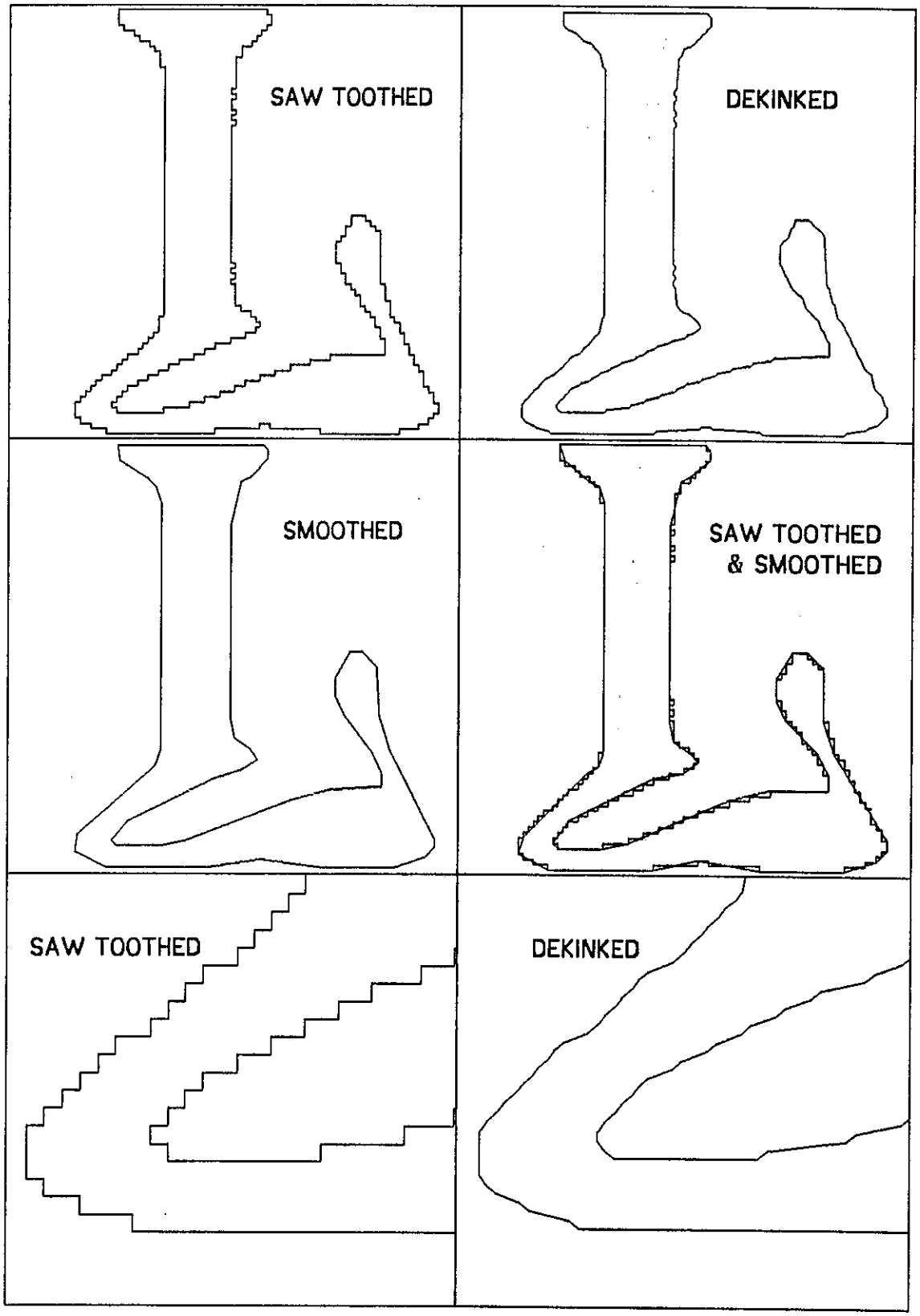
There are two complexities: contrast accumulation and dekinking. The contrast of a vector is defined as (QUOTIENT (DIFFERENCE (Sum of pixel values on one side of the vector)(Sum of pixel values on the other side of the vector)) (length of the vector in pixels)). Since vectors are always either horizontal or vertical and are construed as being on the cracks between pixels; the specified summations refer to the pixels immediately to either side of the vector. Notice that this definition of contrast will always give a positive contrast for vectors of a blob and negative contrast for the vectors of a hole.

The contours that have just been traced would appear "sawtoothed" or "kinky"; the terms "kink", "sawtooth" and "jaggy" are used to express what seems to be wrong about the lowermost left polygon in Figure 7.2. The problem involves doing something to a rectilinear quantized set of segments, to make its continuous nature more evident. In CRE, the jaggies solution (in the subroutine MKPGON) merely positions the turning locus diagonally off its grid point a little in the direction (northeast, northwest, southwest or southeast) that bisects the turn's right angle. The distance of dekink vernier positioning is always less than half a pixel; but greater for brighter cuts and less for the darker cuts; in order to preserve the nesting of contours. The sawtoothed and the dekinked versions of a polygon have the same number of vectors. I am very fond of this dekinking algorithm because of its incredible efficiency; given that you have a north, south, east, west polygon trace routine (which handles image coordinates packed row, column into one word); then dekinking requires only one more ADD instruction execution per vector !

7.4 Polygon Nesting.

The nesting problem is to decide whether one contour polygon is within another. Although easy in the two polygon case; solving the nesting of many polygons with respect to each other becomes n-squared expensive in either compute time or in memory space. The nesting solution in CRE sacrifices memory for the sake of greater speed and requires a 31K array, called the SKY. CRE's accumulation of a properly nested tree of polygons depends on the order of threshold cutting going from dark to light. For each polygon there are two nesting steps: first, the polygon is placed in the

FIGURE 7.2 - SAW TOOTH DEKINKING ILLUSTRATED.



tree of nested polygons by the subroutine INTREE; second, the polygon is placed in the SKY array by the subroutine named INSKY.

The SKY array is 216 rows of 289 columns of 18-bit pointers. The name "SKY" came about because, the array use to represent the farthest away regions or background, which in the case of a robot vehicle is the real sky blue. The sky contains vector pointers; and would be more efficient on a virtual memory machine that didn't allocate unused pages of its address space. Whereas most computers have more memory containers than address space; computer graphics and vision might be easier to program in a memory with more address space than physical space; i.e. an almost empty virtual memory.

The first part of the INTREE routine finds the surrounder of a given polygon by scanning the SKY due east from the uppermost left pixel of the given polygon. The SON of a polygon is always its uppermost left vector. After INTREE, the INSKY routine places pointers to the vertical vectors of the given polygon into the sky array. The second part of the INTREE routine checks for and fixes up the case where the new polygon captures a polygon that is already enclaved. This only happens when two or more levels of the image have blobs that have holes. The next paragraph explains the arcane details of fixing up the tree links of multi level hole polygons; and may be skipped by everyone but those who might wish to implement a polygon nester.

Let the given polygon be named Poly; and let the surrounder of Poly be called Exopoly; and assume that Exopoly surrounds several enclaved polygons called "endo's", which are already in the nested polygon tree. Also, there are two kinds of temporary lists named the PLIST and the NLIST. There is one PLIST which is initially a list of all the ENDO polygons on Exopoly's ENDO ring. Each endo in turn has an NLIST which is initially empty. The subroutine INTREE re-scans the sky array for the polygon due east (to the left) of the uppermost left vector of each endo polygon on the PLIST, (Exopoly's ENDO ring). On such re-scanning, (on behalf of say an Endo1), there are four cases: *No change*; the scan returns Exopoly; which is Endo1's original EXO. *Poly captures Endo1*; the scan returns Poly indicating that endo1 has been captured by Poly. *My brothers fate*; the scan hits an endo2 which is not on the PLIST; which means that endo2's EXO is valid and is the valid EXO of endo1.

My fate delayed; the scan hits an endo2 which is still on the PLIST; which means that endo2's EXO is not yet valid but when discovered it will also be Endo1's EXO; so Endo1 is CONSed into Endo2's NLIST. When an endo polygon's EXO has been rediscovered, then all the polygons on that endo's NLIST are also placed into the polygon tree at that place. All of this link crunching machinery takes half a page of code and is not frequently executed.

7.5 Contour Segmentation.

In CRE the term *segmenting* refers to the problem of breaking a 1-D manifold (a polygon) into simple functions (arcs). The segmenting step, converts the polygons of vertical and horizontal vectors into polygons of arcs. For the present the term "arc" means "linear arc" which is a line segment. Fancier arcs: circular and cubic spline were implemented and thrown out mostly because they were of no use to higher processes such as the polygon compare which would have to break the fancy arcs back down into linear vectors for computing areas, inertia tensors or mere display buffers.

Segmenting is applied to each polygon of a level. To start, a ring of two arcs is formed (a bi-gon) with one arc at the uppermost left and the other at the lowermost right of the given vector polygon. Next a recursive make arc operation, MKARC, is appled to the two initial arcs. Since the arc given to MKARC is in a one to one correspondence with a doubly linked list of vectors; MKARC checks to see whether each point on the list of vectors is close enough to the approximating arc. MKARC returns the given arc as good enough when all the sub vectors fall within a given width; otherwise MKARC splits the arc in two and places a new arc vertex on the vector vertex that was farthest away from the original arc.

The two large images in Figure 7.3, illustrate a polygon smoothed with arc width tolerances set at two different widths in order to show one recursion of MKARC. The eight smaller images illustrate the results of setting the arc width tolerance over a range of values. Because of the dekinking mentioned earlier the arc width tolerance can be equal to or less than one pixel and still yield a substantial reduction in the number of vectors it takes to describe a contour polygon.

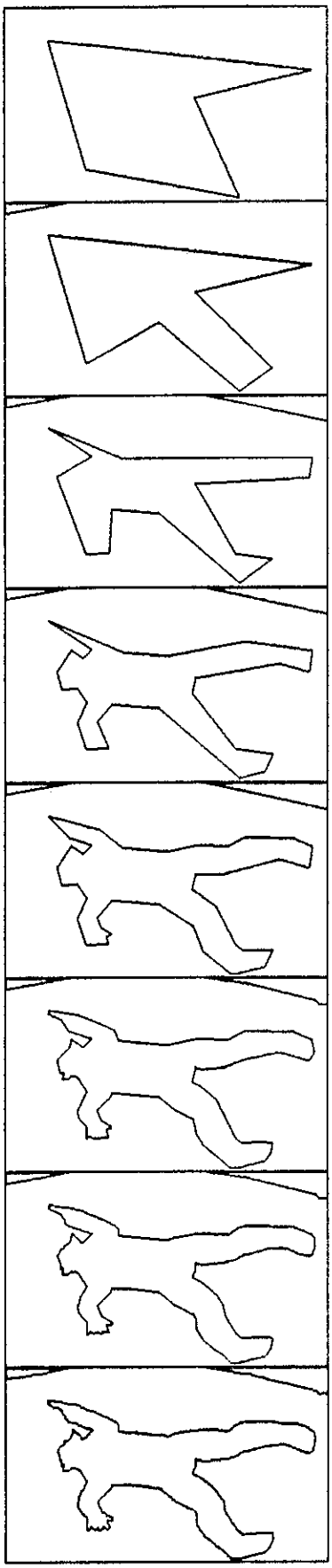
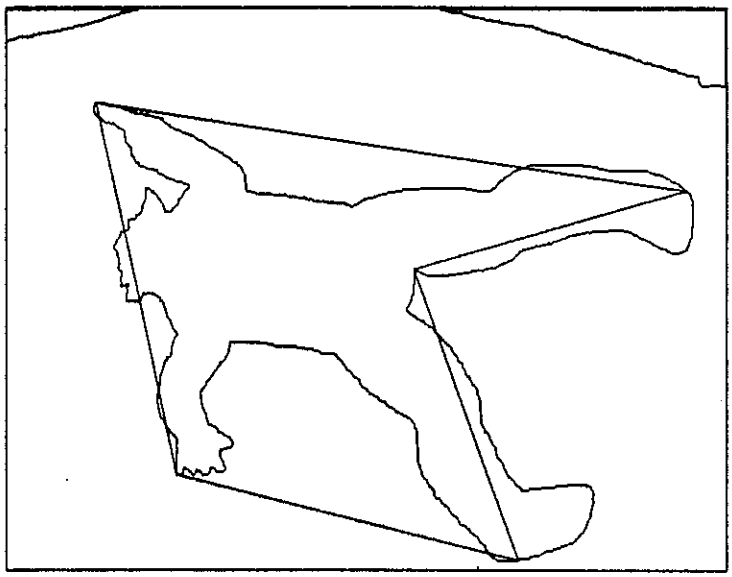
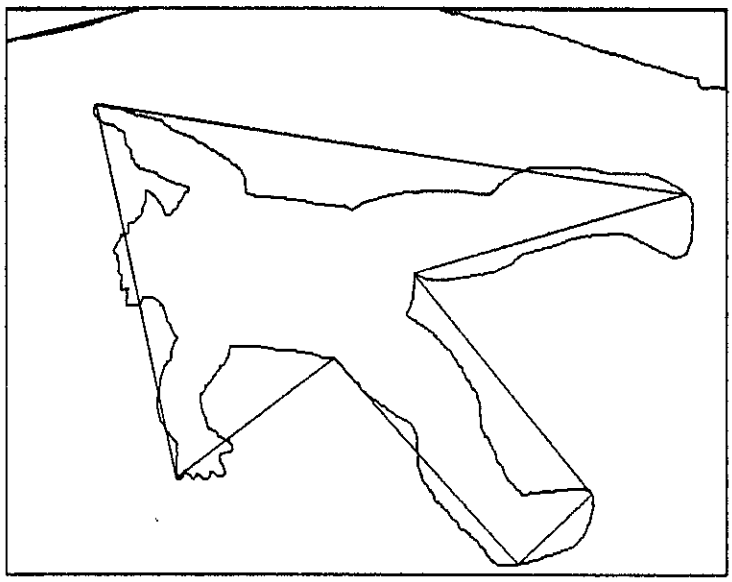


FIGURE 7.3 - CONTOUR SEGMENTATION.





A final important detail is that the arc width tolerance is actually taken as a function of the highest contrast vector found along the arc; so that high contrast arcs are smoothed with much smaller arc width tolerances than are low contrast arcs. After smoothing, the contrast across each arc is computed and the ring of arcs replaces the ring of vectors of the given polygon. (Polygons that would be expressed as only two arcs are deleted).

7.6 Related and Future Image Analysis.

In general, robotic image analysis should consist of three steps: first, high quality pictures are taken continuously in time and space; second, several low level bulk operations (such as correlation, filtering, histogramming and thresholding) are applied to each image and to pairs of images; third, the rasters are converted into linked 2-D structures which are further amalgamated into connected 3-D models. It is clear to me that my present implementation only has fragile toy routines where rugged tools are needed. Eventually, more kinds of image features and larger coherent structures must be included. In particular, the contour maps should be bundled into regional mosaics and more features should be woven into the node/link structure.

Contour image processing is effectively surveyed by Freeman (74) who gives the erroneous impression that contour images are the best image representation (rasters and mosaics are equally important). Contours are applied to recognition of silhouettes by Dudani (70) using moments similar to those explained in the next chapter. Finally, my own acquaintance with the contour image representation was initially derived from papers by Zahn (66) and Krakauer (71).

SECTION 8.

IMAGE COMPARING.

8.0 Introduction to Image Comparing.

The image compare process is both the "*keystone of the arch*" as well as the "*weakest link of the chain*". By comparing images, the 3-D modeling and the 2-D image processing are finally linked, however as will be apparent the implementation to date demonstrates only a small part of what is possible. In the feedback perception design, there are three classes of compare operations: verification, revelation and recognition which may be applied to each of the three kinds of image data structures: raster, contour and mosaic. The verify compare finds the corresponding entities between a predicted image and a perceived image for the sake of calibration measurement and for the sake of eliminating already known features from further consideration. In vision for industrial machine assembly, calibration measurements suddenly seems to be the only kind of vision necessary in a relatively constrained factory situation. The reveal compare involves finding the corresponding entities in two perceived images, so that the location and extent of new objects can be solved. Finally, the recognition compare involves matching a perceived entity with one of a set of prototype entities.

8.3 Compare by Recursive Windowing.

The final step in the CRE polygon match (Section 8.1) is to link the corresponding vertices between two geometrically normalized polygons (or sets of polygons) using a nearest neighbor criterion. The nearest neighbors are found by recursive windowing, initially all the vertices are pushed into one large window which is subsequently split until there were few enough vertices contained in the window to allow exhaustive comparing. To make this windowing technique applicable to the nearest neighbor problem a distance criterion, *delta*, has to be declared so that the windows overlap by that amount. Consequently the windows are no longer disjoint rectangles, but are rather boxes with rounded corners, the smallest possible window being a circle with radius, *delta*. The recursive windowing technique is essentially a two dimensional partition sort, the technique can be generalized for comparing edges and other entities in 2-D or higher dimensions.

8.4 Related Work and Work Yet To Be Done.

To complete the visual feedback system, there remains yet to be written an image compare that uses both raster based and polygon based techniques. The two kinds of compares are symbiotic in that the polygon compare could aim the raster correlator alleviating the need to do bulk correlation over wide areas, and the raster correlator could verify and improve the measurement of corresponding vertex loci. At Stanford, image comparison by raster correlation techniques is begin worked on by Quam(71), Hannah and Morevac. Another approach to comparing polygons is to examine their curvature, the curvature of a polygon can be expressed as a parametric function of arc length; two such functions can be normalized and alligned and differenced using statistical techniques (Zahn 66).

SECTION 9.

CAMERA AND FEATURE LOCUS SOLVING.

9.0 Introduction to Locus Solving.

There are three kinds of locus solving problems in computer vision: camera locus solving, feature locus solving and sun locus solving. Camera solving is routinely attempted in two ways: using one image the 2-D image loci of a set of already known 3-D world loci (perhaps points on a calibration object) are measured and a camera model is computed; or using two or more images a set of corresponding landmark feature points are found among the images and the whole system is solved relative to itself. After the camera positions are known, the location and extent of the objects composing the scene can be found using parallax (motion parallax and stereo parallax). Parallax is the principal means of depth perception and is the alchemist for converting 2-D images into 3-D models. After the camera and object positions are known to some accuracy, the nature and location of light sources might potentially be deduced from the shines and shadows in the images. However, in outdoor situations the primary light source is the sun, whose position in the sky can be computed from the time, date and latitude by means of a simple solar ephemeris routine.

9.1 An Eight Parameter Camera Model.

In GEOMED and CRE the basic camera model is specified by eight parameters. There are three parameters for the lens center location of the camera: CX, CY, CZ; three parameters for the orientation: WX, WY, WZ; and two parameters for the projection ratios: the aspect ratio, AR; and the focal ratio, FR. The location is given in world coordinates and the orientation is specified by a rotation vector whose direction gives an axis and whose magnitude gives rotation which when applied to a set of three axes unit vectors yields a set of unit vectors that determines the camera's coordinate system. By convention the principal ray of the camera is parallel to the Z axis unit vector and is negatively directed. The camera raster is alligned such that the rows (vidicon scan lines) are parallel to the X unit vector and the columns are parallel to the Y unit vector.

The aspect ratio, AR, is the ratio of width, PDX, to height, PDY, of a single vidicon sample point called a pixel: AR = PDX/PDY. The focal ratio, FR, is the ratio of the focal plane distance to the height of a single pixel: FR = FOCAL/PDY. The typical value of the aspect ratio is about one, and the typical value of the focal ratio runs from 300 to 3000.

The actual physical size of the digital raster of a television vidicon is on the order of 12 millimeters wide by 8 millimeters high with approximately 512 lines of potentially 512 pixels per line. However, a standard television scans its raster in two phases (odd rows in one phase, even rows in the next) so that a one-phase pixel is approximately 40 microns by 40 microns (rather than 20 by 20). By contrast, the cones and rods in a human eye are 1 and 2 microns in diameter respectively.

The aspect ratio and the focal ratio can be measured individually using a spherical calibration object. I have used plastic toy balls and billiard balls, billiard ball radius RBB=2.125". The perspective projection of a sphere is an ellipse and the ratio of the apparent width to height of the ellipse of a sphere that nearly fills the viewing screen is the aspect ratio. To measure the focal ratio, mount the sphere on a stick and measure its apparent radii (r1 and r2) at two positions that are approximately along the camera's principal axis a measured distance, DZ, apart. Then then the focal ratio FR =

DZ*r1*r2/(R*(r1-r2)) which can be thought of as the FOCAL plane distance in pixels. The beauty of this is that a naive measuring method yields results as good as measurements obtained by more elaborate methods such as principal axis relaxation of a camera model in numerous variables (Sobel 70) and Pingle unpublished.

Camera Resolution. The focal ratio description allows one to have a firm numerical intuition of camera's spatial resolution in the object space. The smallest distance interval, DELTA, a camera can measure at a given range, RNG, is merely the ratio of range to FR: DELTA=RNG/FR. The arctan of the reciprocal of the focal ratio ARCTAN(1/FR) is the angle subtended by a single pixel.

Lens Center Irrelevancy Theorem. The actual location of the principal axis of the lens in the image plane is irrelevant because the effect of deviation from the true center is equivalent to rotating the camera Many camera modelists worry needlessly about the exact location of the camera lens center; the angular error, ANGERR, of a pixel X units from the center of the image of a camera modeled with a lens center that is wrong in the X direction by Q pixels is given by the following expression:

ANGERR = ARCTAN(X/FR) - ARCTAN((X+Q)/FR) - ARCTAN(Q/FR)

Which for the physical parameters of the television hardware at Stanford in 1974; means that the lens center can be allowed to wander by tens of pixels from its true position without causing a pixel of error at the edge of the image, (allowing that one camera model is alligned on the same feature by rotation as the camera that defines a good lens center).

9.2 Camera Locus Solving: One View of Three Points - The Iron Triangle Camera Solving Method.

A mobile robot having only visual perception must determine where it is going by what it sees. Specifically, the position of the robot is found relative to the position of the lens center of its camera. The following algorithm is a geometric method for computing the locus of a camera's lens center from three landmark points.

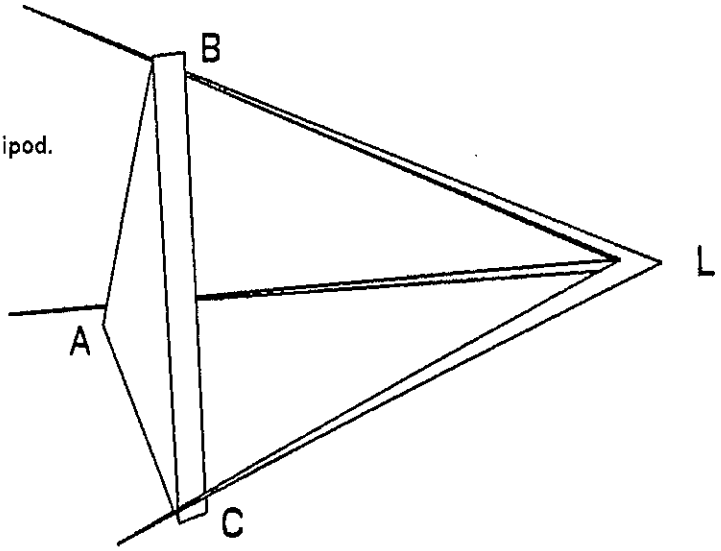
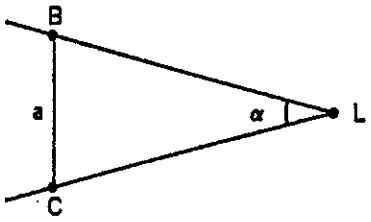


FIGURE 9.1
The Iron Triangle and Tripod.

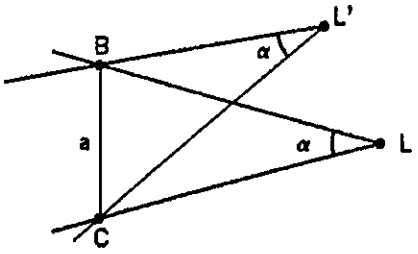
Consider four non-coplanar points A, B, C and L. Let L be the unknown camera's lens center, also called the camera locus. Let A, B and C be the landmark points whose loci either are given on a map or are found by stereo from two already known viewing positions. Assuming for the moment an ideal camera which can see all 4π steradians at once, the camera can measure the angles formed by the rays from the camera locus to the landmark points. Let these angles be called α, β and γ where α is the angle BLC, β is the angle ALC and γ is the angle ALB. The camera also measures whether the landmarks appear to be in clockwise or counter clockwise order as seen from L. If the landmarks are counterclockwise then B is swaped with C and β with γ. A mechanical analog of the problem would be to position a rigid triangular piece of sheet metal between the legs of a tripod so that its corners touch each leg. The metal triangle is the same size as the triangle ABC and the legs of the tripod are rigidly held forming the angles α, β and γ. The algorithm was developed by thinking in terms of this analogy.

FIGURE 9.2 - FIVE IRON TRIANGLE DIAGRAMS.

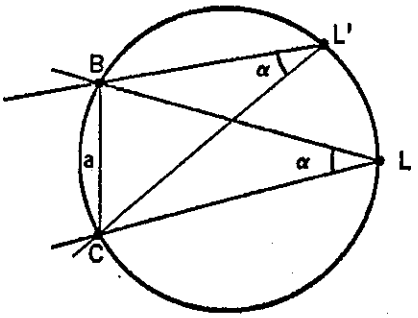
In order to put the iron triangle into the tripod, the edge BC is first placed between the tripod legs of angle α. Let a be the length of BC, likewise b and c are the lengths of AC and AB.



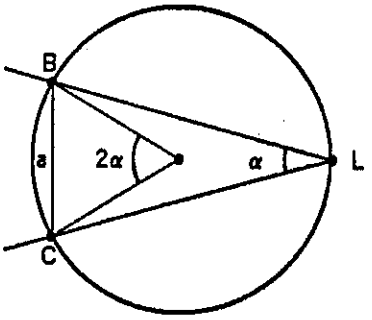
Restricting attention to the plane LBC, consider the locus of points L' arrived at by sliding the tripod and maintaining contacts at B and C.



Remembering that in a circle, a chord subtends equal angles at all points of each arc on either side of the chord; it can be seen that the set of possible L' points lie on a circular arc. Let this arc be called L's arc, which is part of L's circle.



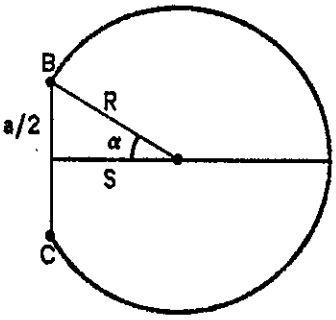
Also in a circle the angle at the center is double the angle at the circumference, when the rays forming the angles meet the circumference in the same two points.



And the perpendicular bisector of a chord passes thru the center of the chord's circle bisecting the central angle. Let S be the distance between the center of the circle and the chord BC. So by trigonometric definitions:

$$R = a / 2\sin(\alpha)$$

$$S = R\cos(\alpha)$$



The position of L on its arc in the plane BLC can be expressed in terms of one parametric variable omega ω, where ω is the counter clockwise angular displacement of L from the perpendicular bisector such that for $\omega=\pi-\alpha$, L is at B and for $\omega=\alpha-\pi$, L is at C. By spinning the iron triangle about the axis BC, the vertex A sweeps a circle. Let H be the radius of A's circle and let D be the directed distance between the center of A's circle and the midpoint of BC. By Trigonometric relations on the triangle ABC:

```
COS(ACB) = (a↑2 + b↑2 - c↑2)/2ab
SIN(ACB) = SQRT(1 - COS(C)↑2)
H = b SIN(ACB)
D = b COS(ACB) - a/2
```

Now consider the third leg of the tripod which forms the angles β and γ. The third leg intersects the BLC plane at point L and extends into the appropriate halfspace so that the landmark points appear to be in clockwise order as seen from L. Let A' be the third leg's point of intersection with the plane containing A's circle. Let the distance between the point A' and the center of A's circle less the radius H of A's circle be called "The Gap". The gap's value is negative if A' falls within A's circle. By constructing an expression for the value of the Gap as a function of the parametric variable ω, a root solving routine can find the ω for which the gap is zero thus determining the orientation of the triangle with respect to the tripod and in turn the locus of the point L in space.

Using vector geometry, place an origin at the midpoint of BC, establish the unit y-vector j pointing towards the vertex B, let the plane BCL be the x-y plane and orient the unit x-vector i pointing into L's halfplane. For right handedness, set the unit z-vector k to i cross j. In the newly defined coordinates points B, C, and L become the vectors:

```
B = (-s, +a/2, 0);
C = (-s, -a/2, 0)
L = (R cos(w), R sin(w), 0)
```

Introducing two unknowns xx and zz the locus of point A' as a vector is:

```
A' = (xx, D, zz)
```

The vectors corresponding to the legs of the tripod are:

LB = B - L = (-s-Rcos(w), +a/2-Rsin(w), 0)
LC = C - L = (-s-Rcos(w), -a/2-Rsin(w), 0)
LA = A'- L = (xx-Rcos(w), D-Rsin(w), zz)

Since the third leg forms the angles β and γ:

LA . LC = |LA| |LC| cos(β)
LA . LB = |LA| |LB| cos(γ)

Solving each equation for |LA| yields:

|LA| = (LA . LC)/|LC|cos(β) = (LA . LB)/|LB|cos(γ)

Multiplying by |LB| |LC| cos (β) cos (γ) gives:

(LA . LC)|LB| cos(γ) = (LA . LB)|LC| cos(β)

Expressing the vector quantites in terms of their components:

|LB| = sqrt((-S-Rcos(w))↑2 + (+a/2-Rsin(w))↑2)
|LC| = sqrt((-S-Rcos(w))↑2 + (-a/2-Rsin(w))↑2)
LA . LC = (xx-Rcos(w))(-s-Rcos(w)) + (D-Rsin(w))(-a/2-Rsin(w))
LA . LC = (xx-Rcos(w))(-s-Rcos(w)) + (D-Rsin(w))(+a/2-Rsin(w))

Substituting:

((xx-Rcos(w))(-s-Rcos(w)) + (D-Rsin(w))(-a/2-Rsin(w))) |LB|cos(γ)
= ((xx-Rcos(w))(-s-Rcos(w)) + (D-Rsin(w))(+a/2-Rsin(w))) |LC|cos(β)

The previous equation is linear in xx, so solving for xx:

xx = P/Q + Rcos(w)

where

P = (-s-Rcos(w))(|LB|cos(γ) - |LC|cos(β))
Q = (D-Rsin(w))((+a/2-Rsin(w))|LC|cos(β)
- (-a/2-Rsin(w))|LB|cos(γ))

The unknown zz can be found from the definition of |LA|

|LA| = sqrt((xx-Rcos(w))↑2 + (D-Rsin(w))↑2 + zz↑2)
so zz = sqrt(|LA|↑2 - (P/Q)↑2 - (D-Rsin(w))↑2)

and since:

|LA| = (LA . LC) / |LC|cos(β)

The negative values of zz are precluded by the clockwise ordering of the landmark points. Thus the expression for the Gap can be formed:

GAP = sqrt((XX+S)↑2 + zz↑2) - H

As mentioned above, when the gap is zero the problem is solved since the locus of A' then must be on A's circle, so the triangle touches the third leg. The gap function looks like a cubic on its interval $[\alpha-\pi,\pi-\alpha]$ which almost always has just one zero crossing.

Having found the locus of L in the specially defined coordinate system all that remains to do is to solve for the components of L in the coordinate system that A, B and C were given. This can be done by considering three vector expressions which are not dependent on the frame of reference and do not have second order L terms, namely: (CA dot CL); (CB dot CL); and ((CA x CB) dot CL). Let the locus of L in the given frame of reference be (X,Y,Z) and let the components of the points A, B and C be (XA,YA,ZA), (XB,YB,ZB) and (XC,YC,ZC) respectively. Listing all four points in both frames of reference:

```
A  = (xx,    D,   zz)    =  (XA, YA, ZA)
B  = (-s, +a/2,   0)     =  (XB, YA, ZA)
C  = (-s, -a/2,   0)     =  (XC, YC, ZC)
L  = (Rcos(w),Rsin(w),0)  =  ( X,  Y,  Z)
```

Evaluating the vector expressions which are invariant:

```
CA = A - C                          =  (XA-XC. YA-YC, ZA-ZC)
CB = B - C = (0, a, 0)              =  (XB-XC, YB-YC, ZB-ZC)
CL = L - C = (Rcos(w)+s,Rsin(w)+a/2,0)  =  ( X-XC,  Y-YC,  Z-ZC)
```

The dot products are:

```
CA . CL = (xx+S)(Rcos(w)+s)+(D+a/2)(Rsin(w)+A/2)
   = (XA-XC)(X-XC) + (YA-YC)(Y-YC) + (ZA-ZC)(Z-ZC)
CB . CL  = a(Rsin(w) + a/2)
   = (XB-XC)(X-XC) + (YB-YC)(Y-YC) + (ZB-ZC)(Z-ZC)
```

The cross product is:

```
(CA x CB) . CL = -a zz(Rcos(w) + s)
      = ((YA-YC)(ZB-ZC) - (ZA-ZC)(YB-YC)) (X-XC)
        - ((XA-XC)(ZB-ZC) - (ZA-ZC)(XB-XC)) (Y-YC)
        + ((XA-XC)(YB-YC) - (YA-YC)(XB-XC)) (Z-ZC)
```

The last three equations are linear equations in the three unknowns X, Y and Z which are readily isolated by Cramer's Rule. The whole method has been implement in auxiliary programs LS1V3P and QBALL which calibrate a camera with respect to a turntable for the sake of the silhouette cone intersection demonstration in Section 9.3.

9.3 Object Locus Solving: Silhouette Cone Intersection.

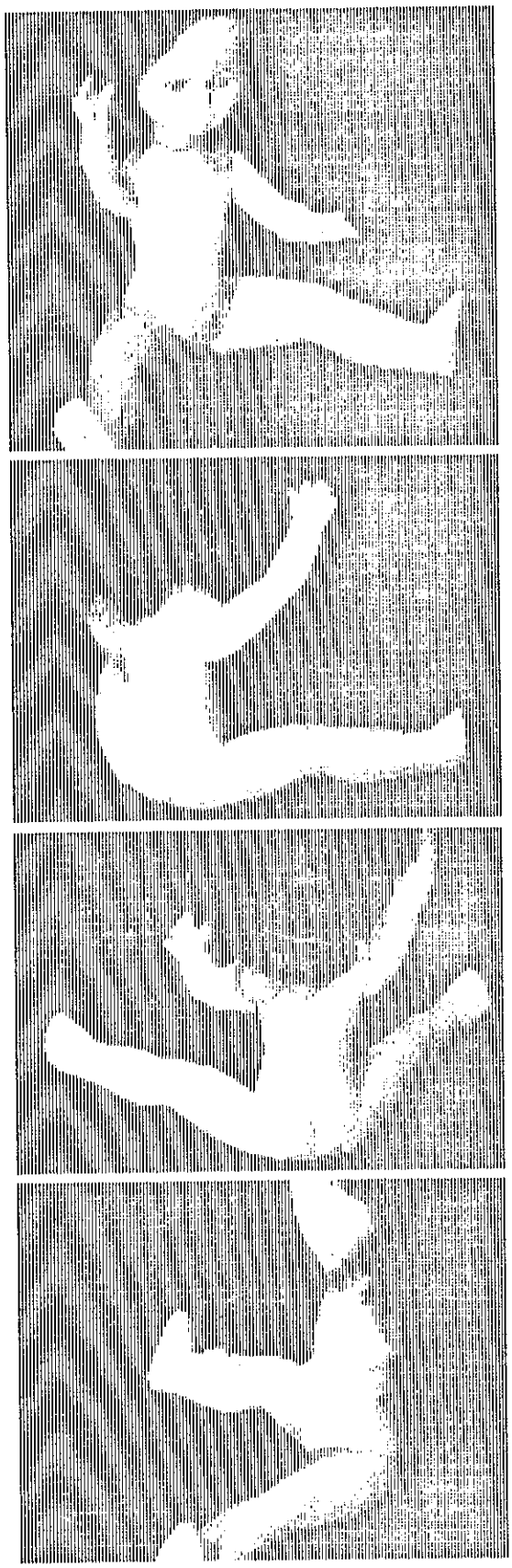
After the camera location, orientation and projection are known; 3-D object models can be constructed. The silhouette cone intersection method is a conceptually simple form of wide angle, stereo reconstruction. The idea arose out of an original intention to do "blob" oriented visual model acquisition, however a 2-D blob came to be represented by a silhouette polygon and a 3-D blob consequently came to be represented by a polyhedron. The present implementation requires a very favorably arranged viewing environment (white objects on dark backgrounds or vice versa); application to more natural situations might be possible if the necessary hardware (and software) were available for extracting depth discontinuities by bulk correlation. Furthermore, the restriction to turntable rotation is for the sake of easy camera solving; this restriction could be lifted by providing stronger feature tracking for camera calibration.

Figure 9.3 shows four video images and the corresponding silhouette contours of a baby doll on a turn table. Figure 9.4 is an overhead view of the four silhouette cones that were swept from the contours, the circle in the middle of Figure 9.4 is the turntable. Figure 9.5 gives three views (cross eyed stereo pairs) of the polyhedron that resulted by taking the intersection of the four silhouette cones. Like in the joke about carving a statue by cutting away everything that does not look like the subject, the approximate shape of the doll is hewed out of 3-D space by cutting away everything that falls outside of the silhouettes. A second example of silhouette cone intersection is depicted in Figure 9.6; the model was made from three silhouettes of the horse facing to the left which can be compared with an initial video image and a final view of the result of the horse facing to the right - a plausible (maximal) backside has been constructed that is consistent with the front views.

The silhouette cone intersection method does indeed construct concave objects and even objects with holes in them - what are missed are concavities with a full rim, that is points on the surface of the object whose tangent plane cuts the surface in a loop that encloses the point.

FIGURE 9.3 - FOUR VIEWS OF A BABY DOLL.

video images



silhouette contours

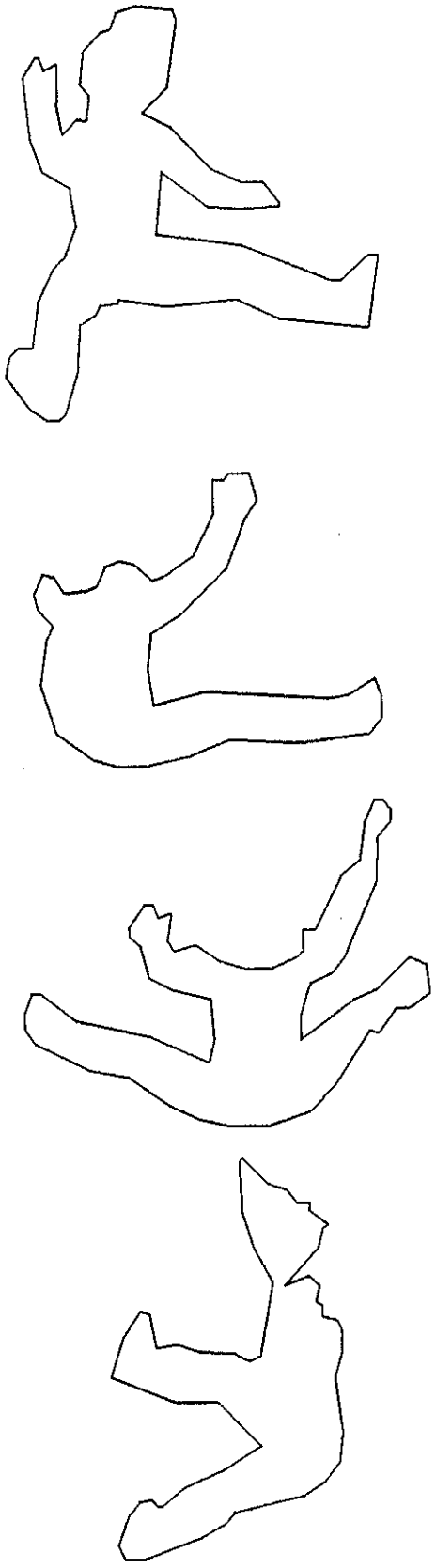


FIGURE 9.4 - FOUR TURNTABLE SILHOUETTE CONES.
...as viewed from above.

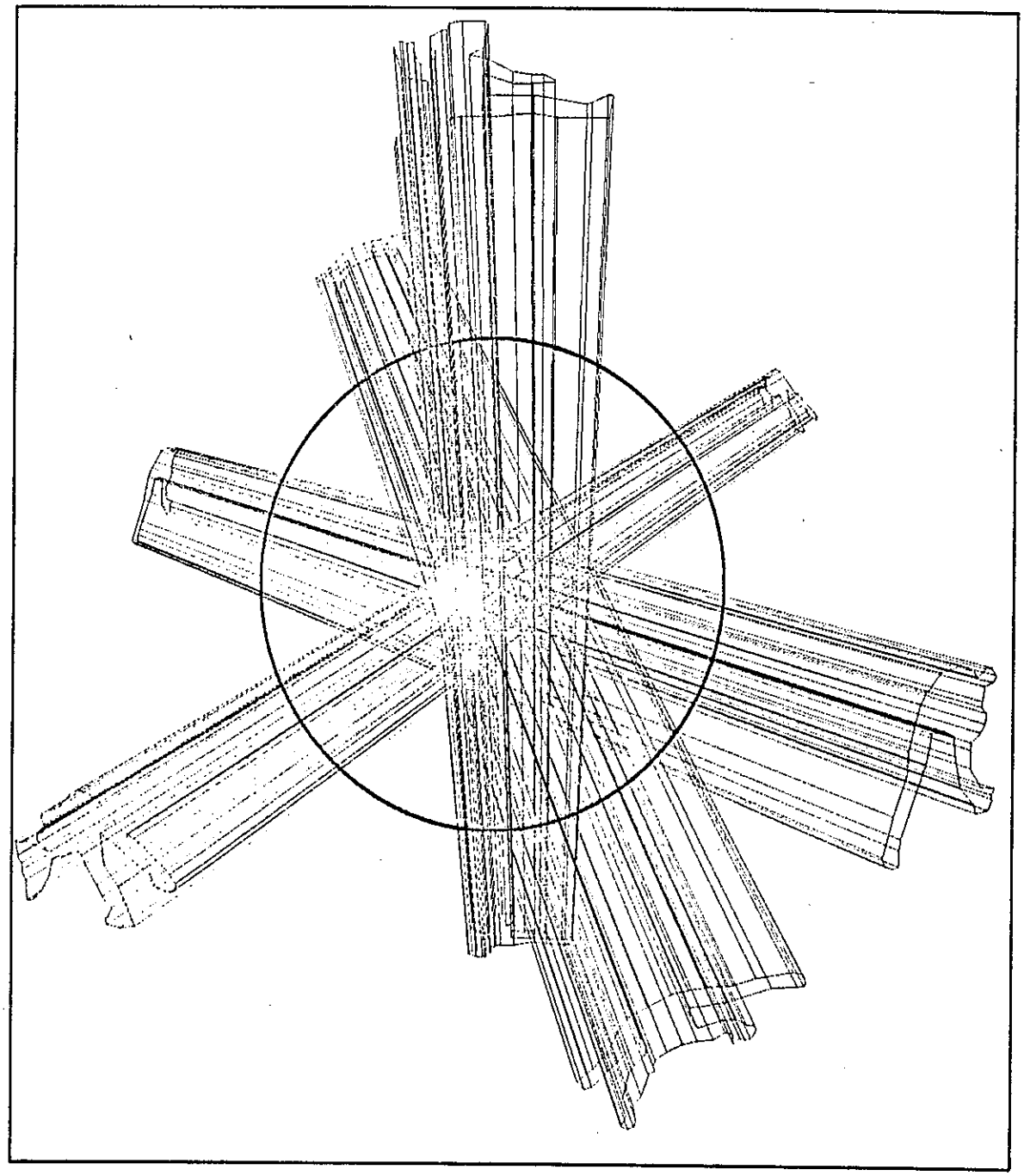
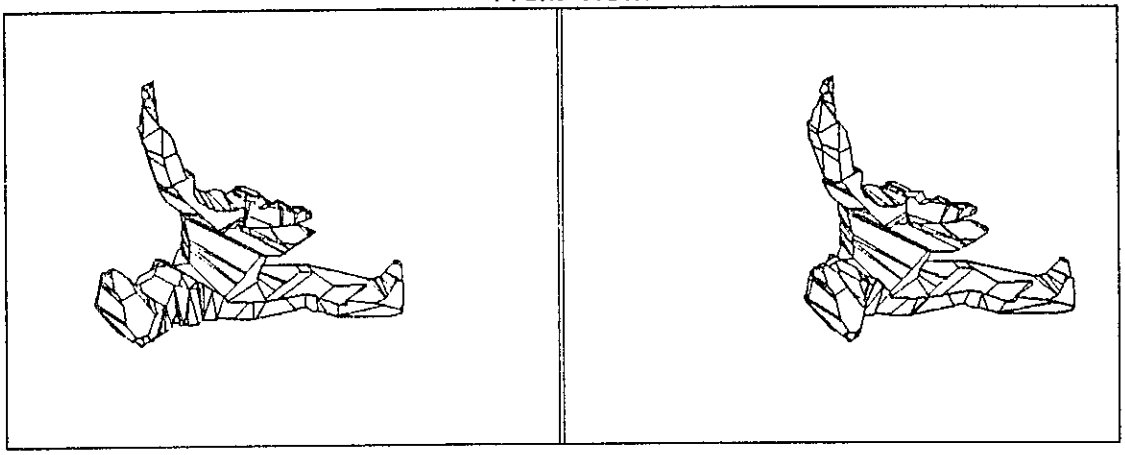
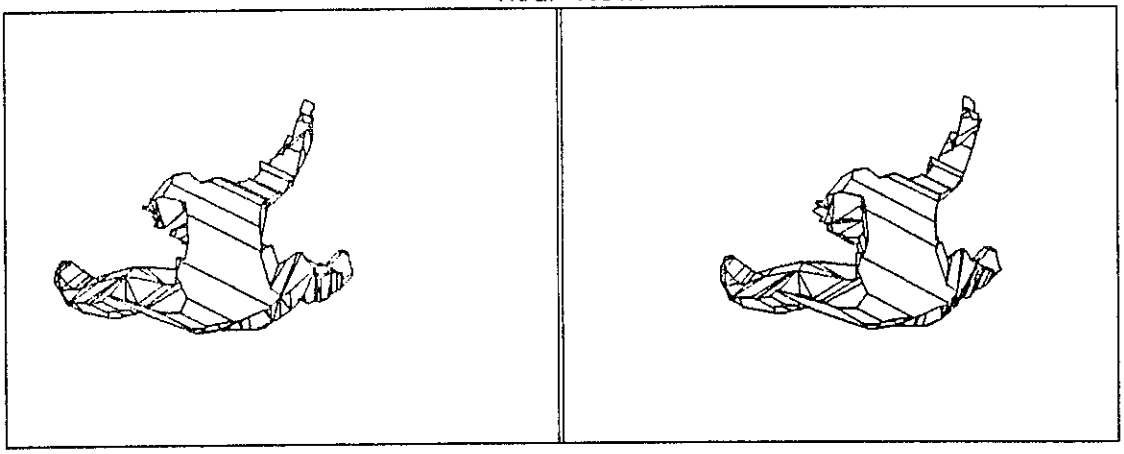


FIGURE 9.5 - RESULTS OF SILHOUETTE CONE INTERSECTION

Front View.



Rear View.



Top View.

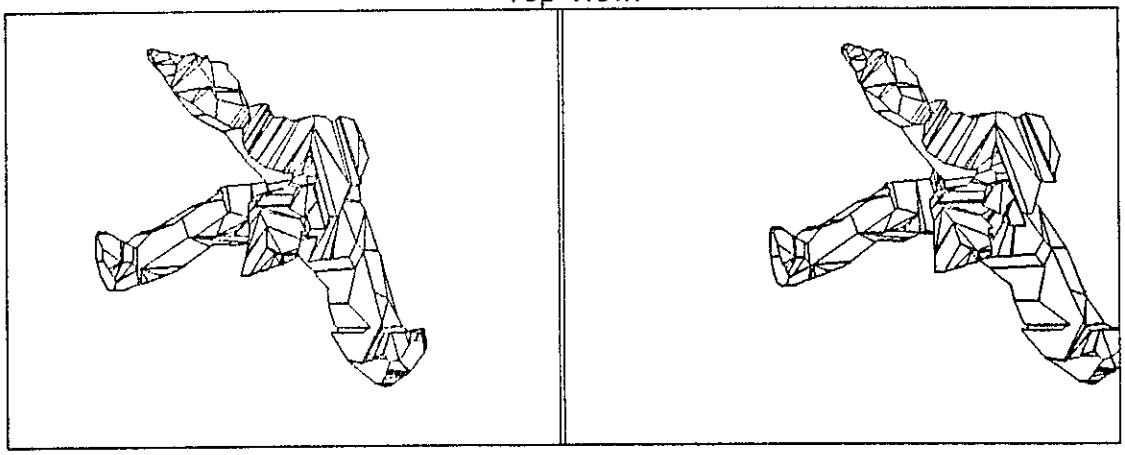
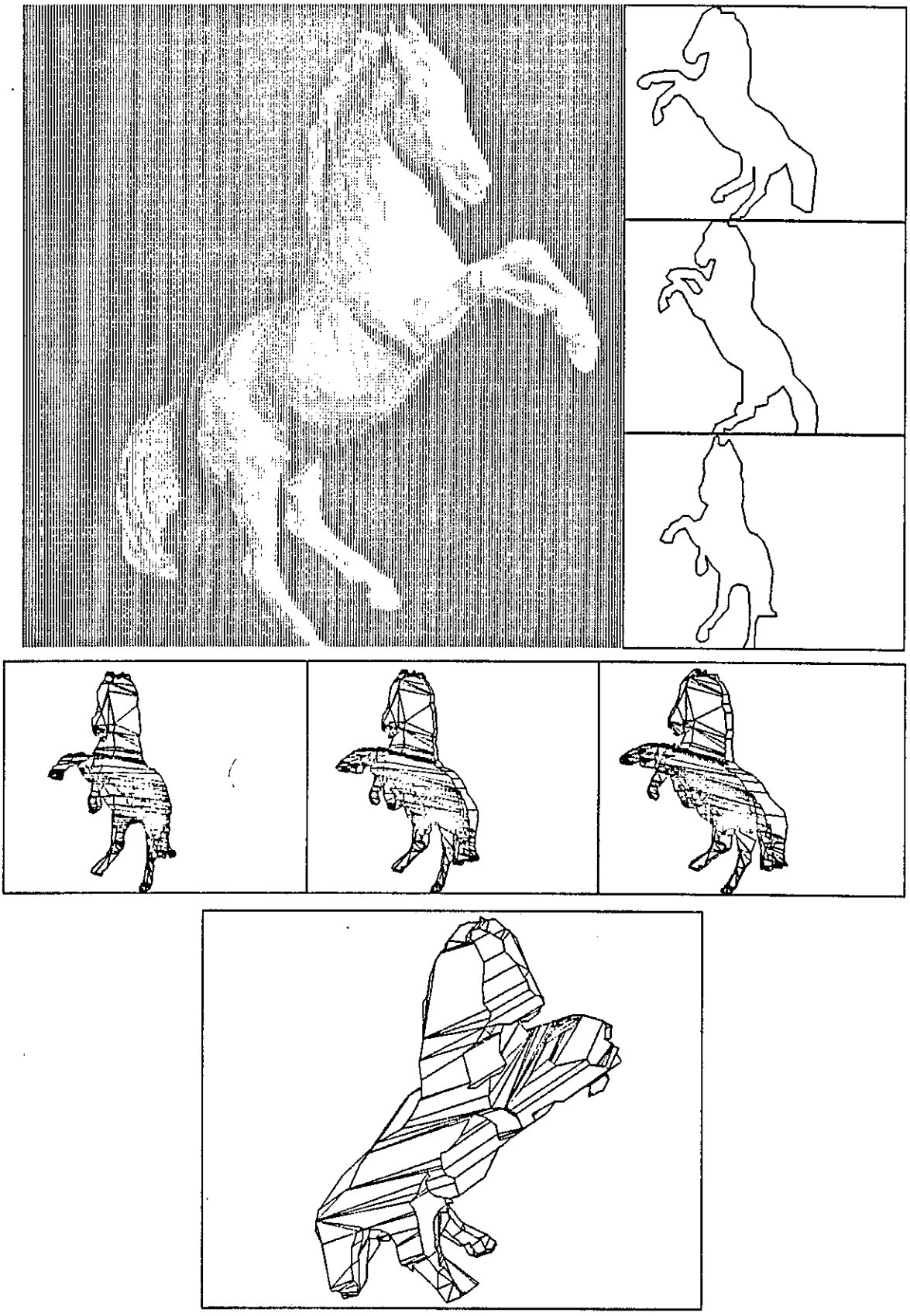


FIGURE 9.6 - HIGH HORSE SILHOUETTE CONE INTERSECTION



9.4 Sun Locus Solving: A Simple Solar Ephemeris.

The location of the sun is useful to a robot vehicle vision system both for sophisticated scene interpretation and for avoiding the blunder of burning holes in the television vidicon. The approximate position of the sun in the sky is readily computed from the time, date and latitude using circular approximations. The longitude is implicitly used to compute Local Solar Time, since the Stanford A.I. Lab is 122 degrees 10 minutes west of the Greenwich meridian, Local Solar time is 8 minutes, 44 seconds earlier than Pacific Standard Time (120 degrees west). The orientation of the earth with respect to the sun follows from remembering that the sun is highest at noon. The tilt of the earth with respect to its orbit is 23.45 degrees, so in earth centered coordinates the sun appears to circle the earth counterclockwise crossing the plane of the equator from south to north on the spring equinox, March 21. The SUNLOCUS procedure given below computes the local azimuth and altitude of the sun in the sky, given the number of days since March 21, the time in seconds since midnight and the latitude in radians.

```
PROCEDURE SUNLOCUS (REAL DAY,TIME,LAT; REFERENCE REAL SUNAZM,SUNALT);
BEGIN
        REAL RHO,PHI,TMP,ECLIPTIC,NORTH,EAST,ZENITH;
COMMENT POSITION OF THE SUN ON THE ECLIPTIC IN THE CELESTIAL SPHERE;
        ECLIPTIC←       ((23+27/60)*PI);
        RHO     ←       2*PI*DAY/365.25;
        EAST    ←       SIN(RHO)*COS(ECLIPTIC);
        NORTH   ←       SIN(RHO)*SIN(ECLIPTIC);
        ZENITH  ←       COS(RHO);
COMMENT LOCAL SOLAR TIME, OVER THE MAST AT NOON;
        TIME    ←       TIME - (8*60 + 44);
        PHI     ←       PI*(1-TIME/(12*3600)) - ATAN2(EAST,ZENITH);
        TMP     ←       ZENITH*COS(PHI) - SIN(PHI)*EAST;
        EAST    ←       EAST*COS(PHI) + SIN(PHI)*ZENITH;
        ZENITH  ←       TMP;
COMMENT ROTATE CLOCKWISE IN THE NORTH/ZENITH PLANE TO LOCAL LATITUDE;
        TMP     ←       COS(LAT)*ZENITH + SIN(LAT)*NORTH;
        NORTH   ←       COS(LAT)*NORTH  - SIN(LAT)*ZENITH;
        ZENITH  ←       TMP;
CONVERT TO ANGULAR MEASURES;
        SUNAZM  ←       ATAN2(NORTH,EAST);      COMMENT AZIMUTH FROM DUE EAST;
        SUNALT  ←       PI/2 - ACOS(ZENITH);    COMMENT ALTITUDE ABOVE HORIZON;
END "SUNLOCUS";
```

9.5 Related and Future Locus Solving Work.

The camera solving problem is discussed in Roberts (63), Sobel (70) and Quam (71). I have always disliked the many dimensional hill climbing approach to camera solving and have sought more geometric and intuitive solutions to the problem. Although the bulk of this chapter concerned camera solving using one view of three points the multi view camera calibration is probably more important to continuous image processing.

SECTION 10.

RESULTS AND CONCLUSIONS.

10.1 Results: Accomplishments and Original Contributions.

As a regular feature in a Ph.D. dessertation, it is required to state explicitly what has been accomplished and what is original. Some of what has been accomplished is itemized in box 10.1; with the so called *original contributions* marked by asterisks. Each of the accomplishments has been elaborated in the indicated chapter.

BOX 10.1 ACCOMPLISHMENTS AND ORIGINAL CONTRIBUTIONS.

0. The Geometric Feedback Vision Theory	Chapter 6.
*1. The Winged Edge Polyhedron Representation	Chapter 2.
*2. The Euler Primitives for Polyhedron Construction	Chapter 3.
3. The Iron Triangle Camera Locus Algorithm	Chapter 9.
*4. The OCCULT hidden line elimination algorithm	Chapter 4.
*5. The Polygon Nesting Algorithm	Chapter 7.
*6. The Polygon Dekinking Method	Chapter 7.
7. The Polygon Segmenting Method	Chapter 7.
8. The Polygon Comparing Method	Chapter 8.
*9. Silhouette Cone Intersection	Chapters 5 and 9.

As a whole, the system described in this thesis is the third of its kind, succeeding the systems of Roberts (1963) and Falk (1970). Although, the modeling routines of the present system are considerably more sophisticated than were those of its predecessors; improvement in the visual analysis routines is less dramatic and more open to question. The present image analysis differs from

the earlier systems in that emphasis is placed on the use of multiple images for the sake of parallax depth perception and in that several spatially connected image representations are combined (contour image, mosaic image and raster image) to preserve the structure of the scene through feature extraction rather than following the earlier paradigm of extracting features from the image piecemeal and attempting to splice them together afterwards.

As a design theory, the present work can be compared with earlier work by comparing the block diagrams. The charcteristically circular feedback vision mandala like diagrams appear in (Falk) Figure 4-7, page 78; (Grape) Figure 12.1, page 242; (Tenenbaum) Figure 1.13, page 43; as well as in this work Figure 6.1, page 70. The feedback mandala is conspicuously absent in the best of the stimulus-response visual parsing work, (Waltz), as well as in statistical recognition work, (Duda and Hart). The important ideas depicted in the feedback vision mandala are the duality of the simulated and physical worlds, the duality of description and verification, the dualism of camera and body locus solving, and the dual opposing flows of predicted and perceived images along a hieracry of commensurate abstractions. Tenenbaum's figure illustrates the basic feedback loop in the immediate vicinity of the visual sensor. The diagrams of Falk and Grape are similar mirrors of the overall system design of the Stanford Hand/Eye group (1969 to 1973) under the leadership of Professor Jerome Feldman. The two diagrams depict an array of relevant boxes (camera solver, edge finder, world modeler and so on) all sending messages to each other under the benign direction of a box labeled "general strategist".

Among the elements composing the GEOMED/CRE system, the most original accomplishment is the winged edge polyhedron representation. In computer graphics models are based on face perimeter lists (or arrays), with an awareness that more topological relations exist but with no realization that a substantial improvement in surface topology modeling is feasible using approximately the same resources.

Another accomplishment, the Euler primitives was based on a constructive proof of the Euler relation from (Coxeter 61). Other graphics systems lack this level of abstraction that falls between the level of node/link operations and operations with solids. The Euler primitives were useful in

implementing OCCULT and GEOMED sweep and glue operations, but they were less useful in implementing the body intersector, BIN.

A pre-computer form of the Iron Triangle camera solving method appears in a paper by Berkay (59). Berkay described the method as an analog procedure to be performed with paper, ruler and afew other photogrammetric hand tools. (The existence of this paper was pointed out to me by Irwin Sobel).

The original accomplishment of the hidden line eliminator, OCCULT lies in its unification of several methods and in its exploitation of object and image coherence made possible by the Euler primitives and the Winged Edge Representation.

The last five accomplishments listed in box 10.1 are related to vision. The nesting and dekinking problems have been stated and solved by others, the present solutions are original only in technical detail: the nesting for its use of memory to avoid a N-squared number of compares and the dekinking for its achievement of good results with almost no effort. The recursive polygon segmentation and the polygon compare idea were accomplishments that were compatible with the contour image approach but are not necessarily original ideas.

10.2 Critique: Errors and Omissions.

The major weakness in the existing modeling system is that it lacks overall unity - the modeling and image anaylsis are not yet sufficiently well integrated. The second major weakness is that the essential subsystems involving comparing, locus solving and recognition are still in a primitive condition. Consequently, an unambiguous objective demonstation of the relevance of 3-D modeling to computer vision is missing; the particular demonstration which I had in mind was to have a robot vehicle drive outside around the laboratory visually servoing along a trajectory given in advance.

In the course of this work, technical failures have included the attempt to use Euler primitives to implement body intersection, the attempt to bundle contour images into mosiac images, as well as

attempts to make the Euler kill primitives logically air tight without time consuming model checking. However, the worst errors are of the form of misallocated effort; more time might have been spent on image analysis and less on image synthesis and so forth. The research suffers from not having a criterion for deciding which objectives deserves the most immediate effort.

A final barrier to progress in computer vision is the inadequacy of the hardware. It may be true that "It is a poor workman who blames his tools"; but for me the greatest source of personal frustration has been the television cameras, the cart and the turntable. At Stanford, these devices have not been implemented or maintained with sufficient care to make them convenient to use.

10.3 Suggestions for Future Work.

Box 10.2 SUGGESTIONS FOR FUTURE WORK.

SPATIAL MODELING WORK.
1. Combination Geometric Models - Converters.
2. Cellular Space Modeling - Tetrahedral Simplices.
3. Spatial Simulation: Collision Avoidance Problem.
4. Higher Dimensionality, 4-D GEOMED.

SIMULATIONS.
5. Mechanical Simulation.
6. Creature Simulations.
7. Geometric Task Planning.
8. Geometric/Semantics Modeling.

MATHEMATICALLY ORIENTED PROBLEMS.
9. The Manifold Resurfacing Problem.
10. The Curved Patches Problem.
11. Prove the Correctness of a Hidden Line Eliminator.

GET RICH QUICK APPLICATIONS.
12. Automatic Machine Shop.
13. Animation for Entertainment Industry.

SYSTEMS SOFTWARE AND VISION HARDWARE WORK.
14. Better Loader and/or Incremental Assembler.
15. Better Cameras.
16. Image Oriented Number Crunching Computer Hardware.
17. Better Robot Vehicles.

The application of geometric modeling to vision and robotics raises numerous interesting ideas and problems, box 10.3. Future development of *Combination Geometric Models* may begin by writing converters between geometric representations. For example, there is a need to convert polyhedra

into spine cross sections, space points into polyhedra, contour maps into faceted surfaces and so on. Extramural combination models include *Geometric/Semantic Modeling* which will be needed to cover the gulf between Minsky's (1974) notion of a visual frame-system (e.g. the expectation of a room interior) and a geometric prediction of the features to be found in the image. Although the Minsky Frame-System theory does not explicitly reveal the crucial interface between numerical geometric modeling and symbolic abstractions, that nexus is a central part of the frame-system idea.

The *Cellular Space Modeling* idea is that both space and objects should be modeled using a space filling tesselation of cells; perhaps using the tetrahedral 3-simplex. The difficulty lies in getting the Euclidean primitives to update the geometry and topology of empty space as an object moves and rotates. The rewards might include an elegant approach to collision avoidance problems in vehicle navigation and arm trajectory planning. Other approaches to *spatial simulation* and *collision avoidance problems* that might be pursued include the use of simulated viewpoints to see obstacle free trajectories by means of hidden line elimination, this method is suggested in (Sutherland 69).

In several recent Stanford dissertations, (Falk, Yakimofsky, Grape, and so on) the authors conclude with the prediction that their essentially 2-D techniques can readily be extended to 3-D in future work. In my turn, I seriously wish to propose that my essentially 3-D techniques can be extended to 4-D. The resulting models could be applied to Regge Calculus for computing the general relativistic geometric models of such systems as two or three colliding blackholes or on a less cosmic level a 4-D GEOMED could be of service for planning sequences of arm manipulations viewing time as a spatial dimension. Collision of 3-D polyhdera moving in time can be described as a static intersection of 4-D polytopes.

Geometric modeling is also applicable to future work in simulation. *Mechanical Simulation* involves computing the Newtonian mechanics of everyday objects, problems which are immediately approachable from a GEOMED foundation include simulated object collision, statics, and pseudo friction. For example, consider what is needed to predict the outcome of setting one more block at a given place on an existing tower or of throwing one block into a tower of other blocks. *Geometric Task Planning* problems include the old A.I. favorite of block stacking as well as the newer research

problems related to industrial assembly. Existing solutions to geometric tasks are notoriously restricted, for example I know of no blocks stacking program that handles arbitrary rotations, all blocks to date are piled on the square.

Although, it has been recognized (early and often) that the programming of numerically controled machine tools should be automated, the actual implementation of a system that builds artifacts directly from a geometric model still lies in the future. As a start, someone at any of the research labs with a general purpose manipulator could begin by carving models out of soap or other soft material with a rotating cutting tool.

Advanced mechanical simulations as well as *Animation for Entertainment* quickly run into the problem of *Creature Simulation* - given a multilegged bug, what control program is required to make the bug walk through a building. Barring the darkness of war, it is likely that the greatest potential future users of robotic simulation will not be found in government, universities, or manufacturing industries but rather in the entertainment industry. When it becomes economically feasible to create realistic (and surrealistic) animation by computer graphics, great progress will be made in simulating visual reality and in representing mundane situations in a computer.

Theoretical work in geometric modeling will continue to pursue curved representations. Two problems that I would especially like to see solved involve fitting curved surfaces to form a smooth object, (a manifold), as well as resurfacing an existing manifold representation. Both problems I believe are more a question of automatic segmentation rather than automatic smoothing. It is easy to fit functions to facial patches of an object, it is hard to subdivide an object into the proper set of patches. In terms of analysis of algorithms and the mathematical theory of computation, the one geometric algorithm that seems most ripe for future quantative study and logical analysis is the hidden line elimination process. There is a wealth of different techniques to be compared and the inputs and outputs seem to be sufficiently well defined for formal axiomatizing.

Finally progress in computer vision and geometric modeling requires progress in systems software and computer systems. In my opinion, recent university based research in programming

languages is over concentrated in very high level language theory and automatic programming. Future language and systems work should include developing an incremental loader, assembler, debugger and editor that can handle algebraic expressions, block structure, node/link storage notation as well as unvarnished machine instructions. Although special purpose image processing hardware has earned a bad reputation (starting with the Illiac-III); in my opinion a real vision system will be composed of a large array of computer like elements (4096 by 4096) that pipeline a stream of images into structured image representations. The perceived images are then compared with predicted images and a detailed 3-D model is altered or constructed in real time (24 images per second) using a small number of computers (32 or less) which by the standards of our day (1974) would be very large and very fast (ten megawords main memory and ten megahertz instruction execution). Assuming the continuation of civilization with a growing technology over the next one hundred to a thousand years, developments in Computer Vision and Artificial Intellegence could lead to robots, androids and cyborgs which will be able to see, to think and to feel conscious.

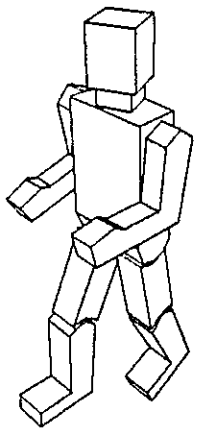
10.4 Conclusions.

The particular technical conclusions of this work include the methods, system designs and data structures for geometric modeling which have already been elaborated. Based on the details, one could make such generalized observations as that: recursive windowing is a good technique for spatial sorting, simple geometric representations fall into space oriented and object oriented classes, the essence of an object representation is its coherence under various operators and that the power of a vision system might be enhanced by application of 3-D modeling techniques. However in closing, I would like to draw three rather more general conclusions, conclusions which by contrast to the technical ones might be construed as scientific conclusions.

1. *The Nature of Perception.* Perception is essential to intelligence as it is the process which converts external sensations into internal thoughts. There are two kinds of simple perception systems: stimulus-response and prediction-correction feedback; together they explain perception.

2. *The Necessity to Experiment.* Robotic hardware is essential to Artificial Intelligence as an experimental science. It is misleading to study only theoretical robotics of plausible abstractions, mathematics, puzzles, games and simulations. The real physical world is the best test of adaptive general intelligence. The complexity and subtlety of real world situations, even of a situation as seemingly finite as a digital television picture, can not be anticipated from a philosopher's armchair or from a programmer's console.

3. *The Necessity to Simulate Visual Reality.* Modeling is essential to prediction-correction feedback perception. Although simulated robot environments should not be used in place of the external physical reality, such environmental simulations are an essential part of a robot's internal mental reality. In the particular case of vision, geometric models should be easy to adapt to the basic mental abilities of present day computer hardware. To conclude, perception requires two worlds one that is the external physical reality and the other which is the internal mental reality.



SECTION 11.
ADDENDA

11.1 References.

Agin (1972)
Gerald Jacob Agin; "Representation and Description of Curved Objects"; Ph.D. Thesis, Computer Science Department, Stanford Artificial Intelligence Laboratory, Memo no. AIM-173, Stanford University, October 1972.

Archuleta (1972)
Michael Archuleta; "Hidden Surface Line Drainwg Algorithm"; University of Utah, Technical Report UTEC-CSc-72-121; Salt Lake City, Utah; June 1972.

Baumgart (1972)
Bruce G. Baumgart; "Winged Edge Polyhderon Representation"; Stanford Artificial Intelligence Laboratory, Memo no. AIM-179, Stanford University, October 1972.

Baumgart (1973)
Bruce G. Baumgart; "Image Contouring and Comparing"; Stanford Artificial Intelligence Laboratory, Memo no. AIM-199, Stanford University, October 1973.

Baumgart (1974)
Bruce G. Baumgart; "GEOMED - A Geometric Editor"; Stanford Artificial Intelligence Laboratory, Memo no. AIM-232, Stanford University, May 1974.

Berkay (1958)
Nedret Berkay; "Determination of Space Coordinates of Photographic Exposures by a Semi-Graphic Method"; Brausch & Lomb Photogrammetry Yearbook; 1958.

Coons (1967)
Steve A. Coons; "Surface for Computer Aided Design of Space Forms"; Project MAC Technical Report, MAC-TR-41, Massachusetts Institute of Technology, Cambridge, Massachusetts; June 1967.

Coxeter (1961)
Harrold S. M. Coxeter; Introduction to Geometry; John Wiley & Sons, New York, 1961.

Coxeter (1963)
Harrold S. M. Coxeter; Regular Polytopes; Macmillan, New York, 1963.

Duda (1973)
Richard Duda and Peter Hart; Pattern Classification and Scene Analysis; John Wiley & Sons, New York, 1973.

Dudani (1970)
Sahibsingh Amulsingh Dudani; "An Experimental Study of Moment Methods for Automatic Identification of Three Dimensional Objects from Television Images."; Ph.D. Thesis, Department of Electrical Engineering; Communication and Control Systems Laboratory, Ohio State University; Columbus, Ohio; August 1970.

Eves (1965)
Howard Eves;A Survey of Geometry;Allyn and Bacon, Boston, 1965.

Falk (1970)
Gilbert Falk; "Computer Interpretation of Imperfect Line Data as a Three Dimensional Scene"; Ph.D. Thesis, Computer Science Department, Stanford Artificial Intelligence Laboratory, Memo no. AIM-132, August 1970.

Feldman (1969)
Jerome Feldman, Gilbert Falk and Lou Paul; "Computer Representation of Simply Described Scenes"; Stanford Artificial Intelligence Laboratory, SAILON-52; Stanford University, 1969.

Feynman (1963)
Richard P. Feynman, Robert B. Leighton, Matthew Sands;
The Feynman Lectures on Physics; Addison-Wesley; Reading, Massachusetts; 1963.

Freeman (1974)
Herbert Freeman; "Computer Processing of Line Drawings"; ACM Computing Surveys, volume 6, number 1; March 1974.

Gardner (1959)
Martin Gardner;
The Scientific American Book of Mathematical Puzzles and Diversions;
Simon and Schuster; New York; 1959.

Gardner (1961)
Martin Gardner;
The 2nd Scientific American Book of Mathematical Puzzles and Diversions;
Simon and Schuster; New York; 1959.

Gill (1972)
Aharon Gill; "Visual Feedback and Related Problems in Computer Controlled Hand Eye Coordination"; Ph.D. Thesis, Computer Science Department, Stanford Artificial Intelligence Laboratory, Memo no. AIM-178, Stanford University, October 1972.

Gips (1974)
James Gips; "Shape Grammars and their Uses"; Ph.D. Thesis, Computer Science Department, Stanford Artificial Intelligence Laboratory, Memo no. AIM-231, Stanford University, May 1974.

Goldstein (1950)
Herbert Goldstein; Classical Mechanics; Addison-Wesley; Reading, Massachusetts; 1950.

Gouraud (1971)
Henri Gouraud; "Computer Display of Curved Surfaces"; Ph.D. Thesis, Department of Computer Science, University of Utah, Technical Report UTEC-CSc-71-113; Salt Lake City, Utah; June 1971.

Grape (1973)
Gunnar R. Grape; "Model Based (Intermediate-Level) Computer Vision"; Ph.D. Thesis, Computer Science Department, Stanford Artificial Intelligence Laboratory, Memo no. AIM-201, Stanford University, May 1973.

Graustein (1935)
William C. Graustein; Differential Geometry; Macmillan; New York; 1935.

Guzman (1968)
Adolfo Guzman; "Computer Recognition of Three Dimensional Objects in a Visual Scene"; Ph.D. Thesis, Department of Electrical Engineering, Project MAC Technical Report, MAC-TR-59, Massachusetts Institute of Technology, Cambridge, Massachusetts; December 1968.

Hilbert (1952)
David Hilbert and S. Cohn-Vossen; translated by Nemenyi, P.;
Geometry and the Imagination; Chelsea Publishing Company; New York; 1952.

Knuth (1968)
Donald Ervin Knuth; The Art of Computer Programming; Addison-Wesley; Reading,Massachusetts; 1968.

Krakauer (1971)
Lawrence J. Krakauer; "Computer Analysis of Visual Properties of Curved Objects"; Project MAC Technical Report, MAC-TR-82, Massachusetts Institute of Technology, Cambridge, Massachusetts; May 1971.

Luzadder (1971)
Warren J. Luzadder; Fundamentals of Engineering Drawing; Printice Hall; Englewood Cliffs, New Jersey; 1971.

Maruyama (1972)
Kiyoshi Maruyama; "A Procedure to Determine Intersections Between Objects"; International Journal of Computer and Information Sciences, volume 1, number 3, 1972.

McCarthy (1964)
John McCarthy; "Computer Control of a Machine for Exploring Mars'; Stanford Artificial Intelligence Laboratory, Memo no. AIM-14, Stanford University, June 1964.

McCarthy (1968)
John McCarthy and Patrick Hayes; "Some Philosophical Problems from the Standpoint of Artificial Intelligence"; Stanford Artificial Intelligence Laboratory, Memo no. AIM-73, Stanford University, November 1968.

Minsky (1974)
Marvin Minsky; "Frame-Systems"; Unpublished Paper, MIT-AI LAB 1974; (cf. draft version of 27 February 1974; SAIL internal document).

Muller (1967)
Edward J. Muller; Architectural Drawing *and Light Construction*; Printice-Hall; Englewood Cliffs, New Jersey; 1967.

Nevetia (1974)
Ramakant Nevetia; "Structured Descriptions of Complex Objects for Recognition and Visual Memory"; Ph.D. Thesis, Computer Science Department, Stanford University, (Forthcoming) 1974.

Newman and Sproull (1973)
William M. Newman and Robert F. Sproull;
Principles of Interactive Computer Graphics; McGraw-Hill; New York; 1973.

Parke (1972)
Frederic Ira Parke; "Computer Generated Animation of Faces"; Ph.D. Thesis, Department of Electrical Engineering, University of Utah, Technical Report UTEC-CSc-72-123; Salt Lake City, Utah; June 1972.

Paul (1969)
Richard Paul, Gilbert Falk and Jerrome A. Feldman; "The Computer Representation of Simply Described Scenes"; Stanford Artificial Intelligence Laboratory, Memo no. AIM-101, Stanford University, October 1969.

Paul (1972)
Richard Paul; "Modelling, Trajectory Calculation and Servoing of a Computer Controlled Arm"; Ph.D. Thesis, Computer Science Department, Stanford Artificial Intelligence Laboratory, Memo no. AIM-177, Stanford University, November 1972.

Quam (1971)
Lynn H. Quam; "Computer Comparison of Pictures"; Ph.D. Thesis, Computer science Department, Stanford Artificial Intelligence Laboratory, Memo no. AIM-144, Stanford University, May 1971.

Quam et al (1972)
Lynn H. Quam, Sidney Liebes, Robert B. Tucker, Botond G. Eross and Marsha Jo Hannah; "Computer Comparison of Pictures"; Stanford Artificial Intelligence Laboratory, Memo no. AIM-166, Stanford University, April 1972.

Roberts (1963)
Larry G. Roberts; "Machine Perception of Three Dimensional Solids"; Lincoln Laboratory Technical Report no. 315; Lexington, Massachusetts; May 1963.

Rosenfeld (1969)
Azriel Rosenfeld; "Picture Processing by Computer"; ACM Computer Surveys, volume 1, number 3; September 1969;

Schmidt (1971)
Rodney A. Schmidt; "A Study of the Real-Time Control of a Computer Driven Vehicle"; Ph.D. Thesis, Department of Electrical Engineering; Stanford Artificial Intelligence Laboratory, Memo no. AIM-149, Stanford University, May 1971.

Snyder (1914)
Virgil Snyder and C.H.Sisam; Analytic Geometry of Space; Henry Holt and Company; New York; 1914.

Sobel (1970)
Irwin Sobel; "Camera Models and Machine Perception"; Ph.D. Thesis, Department of Electrical Engineering; Stanford Artificial Intelligence Laboratory, Memo no. AIM-121, Stanford University, May 1970.

Stewart (1970)
Bonnie Stewart; Adventures Among the Toroids; Okemos, Michigan; 1970.

Sutherland, Sproull and Schumacker(1973)
Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker; "A Characterization of Ten Hidden-Surface Algorithms"; Evans & Sutherland Computer Corporation, Salt Lake City, Utah; 1973. (also published in: ACM Computing Surveys; volume 6, number 1; March 1974).

Sutherland (1969)
Ivan E. Sutherland; draft copy of "A Method for Solving Arbitrary-wall mazes by Computer"; which later appeared in the IEEE transactions on Computers, 1969.

Sutherland (1970)
Ivan E. Sutherland; "Computer Displays"; Scientific American, volume 222, number 6; June 1970.

Sutro and Kilmer(1969)
Louis L. Sutro and William L. Kilmer; "Assembly of Computers to Command and Control a Robot"; Instrumentation Laboratory, Report number R-582; Massachusetts Institute of Technology, Cambridge, Massachusetts; February 1969.

Symon(1953)
Keith R. Symon; Mechanics; Addison-Wesley; Reading, Massachusetts; 1953.

Tenenbaum (1970)
Jay Martin Tenenbaum; "Accommodation in Computer Vision"; Ph.D. Thesis, Department of Electrical Engineering, Stanford Artificial Intelligence Laboratory, Memo no. AIM-134, Stanford University, October 1970.

Waltz (1972)
David L. Waltz; "Generating Semantic Descriptions from Drawings of Scenes with Shadows"; MIT Aritificial Intelligence Laboratory, Technical Report, AI-TR-271, Massachusetts Institute of Technology, Cambridge, Massachusetts; November 1972.

Warnock (1968)
John E. Warnock; "A Hidden-Line Algorithm for Halftone Picture Representation," Technical Report 4-5, Department of Computer Science, University of Utah, Salt Lake City, Utah; May 1968.

Warnock (1969)
John E. Warnock; "A Hidden-Surface Algorithm for Computer Generated Halftone Pictures"; Technical Report 4-15, Department of Computer Science, University of Utah, Salt Lake City, Utah; June 1969.

Watkins (1970)
G. S. Watkins; A Real-Time Visible Surface Algorithm"; University of Utah, Technical Report UTEC-CSc-70-101; Salt Lake City, Utah; June 1970.

Winograd (1971)
Terry Winograd; "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language"; Ph.D. Thesis, Department of Mathematics; MIT Aritificial Intelligence Laboratory, Technical Report, AI-TR-17 or MAC-TR-84, Massachusetts Institute of Technology, Cambridge, Massachusetts; January 1971.

Winograd (1974)
Terry Winograd; "Frame Representations and the Declarative/Procedural Controversy"; (forthcoming), 1974.

Yakimovsky (1973)
Yoram Yakimovsky; "Scene Analysis Using a Semantic Base for Region Growing"; Ph.D. Thesis, Computer Science Department, Stanford Artificial Intelligence Laboratory, Memo no. AIM-209, Stanford University, June 1973.

Zahn (1966)
Charles T. Zahn; "Two-Dimensional Pattern Description and Recognition via Curvaturepoints"; Stanford Linear Accelerator Center, SLAC Report no. 70, Stanford University, December 1966.

11.2 GEOMED Node Formats.

The latest (June 1974), public implementation of GEOMED distinguishes sixteen different node formats at the user level: Tram, Empty, Universe, Sun, Camera, World, Window, Image, Text, Xnode, Ynode, Znode, Body, Face, Edge and Vertex. Of the sixteen nodes, five are unimplemented, open ended or trivial and so will not be exhibited: Empty, Text, Xnode, Ynode and Znode. The empty node contains all zeroes except for a one in the status word and a free list pointer in the PFACE field. The Text nodes were implemented in 1973 by Tovar Mock and were taken out. The X, Y and Z nodes are used for miscellaneous things such as beads, one-word atoms and inertia tensors. Field names printed in capital letters indicate that the contents of that field have one standard intrepretation; lower case field names are temporary intrepretations. The machine address of a node points to word zero of the format diagrams.

TRAM NODE-0 FORMAT

The tram node, explained in Section 3.3, represents both Cartesian coordinate systems and Euclidean transformation. Although the status bits contain data, TRAM nodes are can be distinguished from other nodes because bits 0 and 9 are either different or the word is all zeroes in the PDP-10 floating number format.

-3	XWC	Location of TRAM origin
-2	YWC	or Vector of TRAM translation.
-1	ZWC	
0	IX	X-axis unit vector
1	IY	or 3 by 3 rotation matrix.
2	IZ	
3	JX	Y-axis unit vector
4	JY	
5	JZ	
6	KX	Z-axis unit vector
7	KY	
8	KZ	

UNIVERSE NODE-2 FORMAT

The Universe node is the unique root of the data structure and represents the universe of discourse. Directly accessible from the universe node are the free storage list, the world ring and the display ring. The world ring and display rings are headless so two pointers are kept one indicating a "now" entity and the other indicating the "first" made entity.

-3			
-2			
-1			
0	STATUS BITS		
1		AVAIL	Free Storage List of Nodes.
2			
3			
4	NWRLD	PWRLD	Now World, First World.
5			
6			
7	NDPY	PDPY	Now Display Ring, First Display Ring.
8			

SUN NODE-3 FORMAT

The sun node represents a very distant point light source. The sun belongs to a ring of suns that belongs to a world, although handling of multiple light sources is quite premature. The location and orientation of the sun is carried by a TRAM pointed to by the TRAM field.

-3			
-2			
-1			
0	STATUS BITS		
1			
2			
3			
4		PWRLD	World containing this sun.
5	BRO	SIS	Ring of Suns.
6	alt	TRAM	Location/Orientation of Sun.
7			
8	nlnk	plnk	User links.

CAMERA NODE-4 FORMAT

The camera node contains the scale constants of projection, the physical pixel size, PDX and PDY; the logical image size, LDX and LDY; and the focal plane distance FOCAL.

-3	scalex = -focal/pdx		Perspective Projection Scales.
-2	scaley = -focal/pdy		
-1	scalez = -focal/pdz		
0	STATUS BITS		
1	PDX	LDX	Physical Pixel Size
2	PDY	LDY	and Logical image size.
3	FOCAL		Focal Plane distance.
4		PWRLD	World of Camera.
5	BRO	SIS	Camera Ring.
6	alt	TRAM	Camera location/orientation.
7	SIMAG	PIMAG	Simulated and Perceived Image Rings.
8	nlnk	plnk	User links.

WORLD NODE-5 FORMAT

The world node has a ring of bodies, a ring of cameras, and a ring of suns which comprise the totality of existence for image simulation. One world is the reality world whose cameras correspond to actual video hardware and whoes bodies correspond to the physical objects actually in the proximity of the cameras. Other worlds are fantasy worlds for planning and learning.

-3	time and date		Simulated World Time.
-2	PNAME1		Print Name of World.
-1	PNAME2		
0	STATUS BITS		
1	nface	pface	Potentially visible face list.
2	ned	ped	Potentially visible edge list.
3			
4	NCAMR	PCAMR	Now camera and First camera.
5	BRO	SIS	World Ring.
6	NSUN	TRAM	Sun Ring and World Coordinates.
7	CW	CCW	Head links of Body Ring of World.
8	nlnk	plnk	User links.

WINDOW NODE-6 FORMAT

The display window node represents a mapping from a camera's image coordinates (source image) to a display device's screen coordinates (object image). Window mappings can be composed. The mapped window is clipped to a border XL, XH, YL, YH in object coordinates after being dilated by the scale factor MAG. The windows are organized into a ring of displays which each consists of a ring of windows.

-3	SX	SY	Locus of center of Source Image.
-2	OX	OY	Locus of center of Object Image.
-1	MAG		Magnification of Window Mapping.
0	STATUS BITS		
1	XL	XH	Object Image Clipping Border.
2	YL	YH	
3			
4	NCAMR		Now Camera of Window.
5	BRO	SIS	Window ring of a display.
6			
7	CW	CCW	Display ring of window rings.
8	nlnk	plnk	User Links.

IMAGE NODE-7 FORMAT

Image nodes represent either perceived contour images created by input from CRE or simulated mosiac images created by the hidden line eliminator, OCCULT. Like a world, images carry a list of bodies and a time representing when the image was taken. Image nodes also carry a pointer to a copy of the camera and sun under which they were made.

-3			
-2	PNAME1		Corresponding Video image file name.
-1	PNAME2		
0	STATUS BITS		
1			
2			
3			
4	NCAMR	PWRLD	Camera Copy and World of this image.
5	NTIME	PTIME	Image ring links to form a film.
6	ALT		Corresponding image.
7	CW	CCW	Head links of image body ring.
8	nlnk	plnk	User Links.

BODY NODE-14 FORMAT

The body node is the head of the face, edge and vertex rings which use word 1, 2, and 3. The body node carries a parts tree structure in word 4 and 5. There is a print name of up to ten characters carried in words -2 an -1. The links of the eighth word are always left free for linkage to user data structures.

-3			
-2	PNAME1		Ten character print name.
-1	PNAME2		
0	STATUS BITS		
1	NFACE	PFACE	Face ring.
2	NED	PED	Edge ring.
3	NVT	PVT	Vertex ring.
4	DAD	SON	Parts Tree links: up and down tree.
5	BRO	SIS	Parts Tree links: ring of siblings.
6	alt	TRAM	Body coordinate system TRAM.
7	CW	CCW	Body ring of world.
8	nlnk	plnk	User links.

FACE NODE-15 FORMAT

The face node carries a normalized face normal vector in AA, BB, and CC; the negative distance of the face plane from the orgin, KK; photometric parameters are kept in words 4, 5 and 7.

-3	AA		Face plane normal vector.
-2	BB		
-1	CC		
0	STATUS BITS		
1	NFACE	PFACE	Face ring of a body.
2	Ncnt	PED	Edge count and first edge.
3	KK		Distance of face plane from origin.
4	red grn blu wht		Reflectivities under four filters.
5	Lr Lg Lb Lb Sm Sn		Luminosities and Spectral constants.
6	alt	alt2	Temporaries.
7	QQ		Video Intensity under four filters.
8	nlnk	plnk	User Links.

EDGE NODE-16 FORMAT

The important fields of the winged edge node are explained in Chapter 2. The negative three words are used for edge coefficients and for clipped display coordinates of the edge. The alt, alt2 and cw field are used as temporary fields in OCCULT, BIN and so on. The CCW field points at body of edge and expedites BGET. The nlnk and plnk fields are kept empty for developmental work.

-3	x1dc	AA	y1dc	Clipped Display Coordinates or
-2	x2dc	BB	y2dc	2-D Edge Coefficients or
-1		CC		3-D line Cosines.
0		STATUS BITS		
1	NFACE		PFACE	Two faces of the edge.
2	NED		PED	Edge ring of the body.
3	NVT		PVT	Two vertices of the edge.
4	NCW		PCW	Wings: neighboring edges in PFACE and
5	NCCW		PCCW	Neighboring edges in NFACE.
6	alt		alt2	Temporaries.
7	cw		ccw	Temporary and Body Link.
8	nlnk		plnk	User links.

VERTEX NODE-17 FORMAT

The vertex node carries a point's locus in three coordinate systems: world coordinates, perspective projected coordinates and display coordinates. The first edge of a vertex perimeter is contained in the PED field. The alt, alt2, cw, ccw and Tjoint fields are used as temporaries.

-3		XWC		World Locus
-2		YWC		
-1		ZWC		
0		STATUS BITS		
1	XDC		YDC	Display Screen Locus.
2	Tjoint		PED	Temporary and First Edge.
3	NVT		PVT	Vertex ring of the body.
4		XPP		Perspective Projected Locus.
5		YPP		
6	alt	ZPP	alt2	...also used for temporaries.
7	cw		ccw	temporaries.
8	nlnk		plnk	User links.