STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-179

STAN-CS-320

WINGED EDGE POLYHEDRON REPRESENTATION

BY

BRUCE G. BAUMGART

SUPPORTED BY

ADVANCED RESEARCH PROJECTS AGENCY

ARPA ORDER NO. 457

OCTOBER 1972

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

STANFORD ARTIFICIAL INTELLIGENCE PROJECT OCTOBER 1972
MEMO AIM-179

COMPUTER SCIENCE DEPARTMENT REPORT
NO, CS-320

WINGED EDGE POLYHEDRON REPRESENTATION,

Bruce G, Baumgart

Abstract: A winged edge Polyhedron representation is stated and a set of primitives that preserve Euler's F-E+V = 2 equation are explained, Present use of this representation in Artificial Intelligence for computer graphics and world modeling is illustrated and its Intended future application to computer vision is discribed,

CONTENTS

I, INTRODUCTION,
A, Introduotlon to World Modeling,
B, Introduction to a Camera Model,
C, Introduction to Body, Face, Edge, Vertex Modeling,

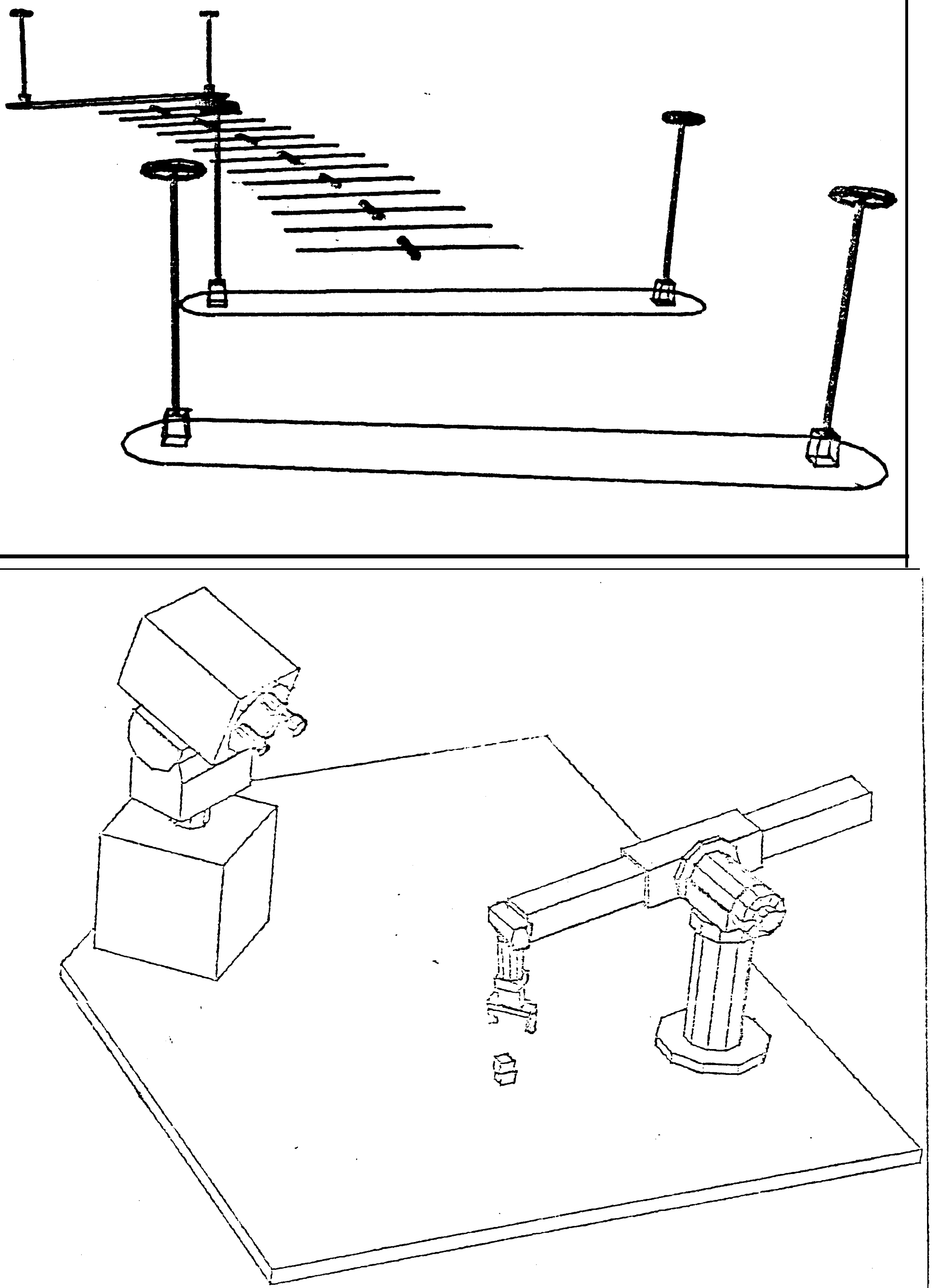
II, DATA STRUCTURE of Winged Edge Polyhedra,
A, Winged Edge Structure,
B, Winged Edge Operations,
C, Elaborations,

III, PRIMITIVES on Polyhedra,
A, Euler Primitives,
B, Solid Primitives,
C, Geometric Primitives,
D, Image Forming Primitives,

IV, APPLICATIONS,
A, Modeling: GEOMED - a 3D drawlng editor,
B, Graphics: OCCULT - a hidden line eliminator,
C, Vision: CAREYE - a video region-edge finder,

This research was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defence under contract SD-183,

FIGURE 1.1 - Examples of World Model Scenes.



I. INTRODUCTION.

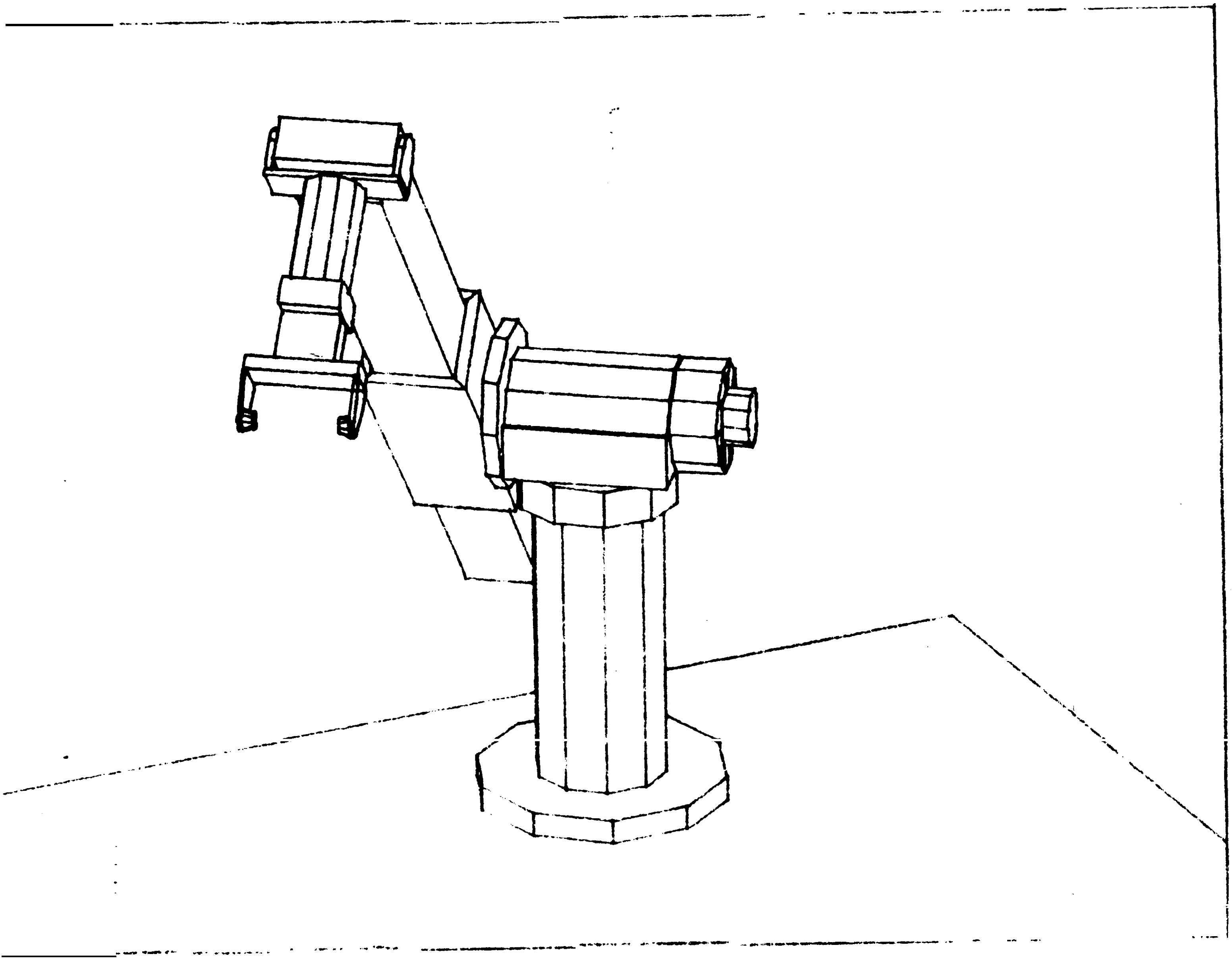
In order to get a computer to deal with the physical world it must have a data representation on which computations involving space, time, shape, size, and the appearance of things can be done. It is my current prejudice that polyhedra provide the proper starting point for building such a physical world representation. At Stanford Artificial Intelligence, Binford and Agin have started instead with spine-cross section models as an alternate approach to the same problems [reference 1]. Other researchers with somewhat different goals, are attempting to build semantic, predicate calculus, problem solving, or startegy planning World models. In any event, this paper is about a body, face, edge, vertex polyhedron model that is for modeling objects and scenes of objects for the sake of computer vision.

Although the data structure to be discussed is not language dependent, the terminlogy and examples will follow ALGOL and LISP. Also, the reader is assumed to have some acquaintance with the ideas associated with the following technical terms:

A: block, node, item, element, atom,
R: link, pointer, address, reference.
C: datum, content, value,
D: list, ring, stack, pdl, tree,
E: dynamic free storage & memory allocation,

A thorough presentation of these terms and ideas can be found in chapter two of volume one of Knuth's cookbook, 'The Art of Computer Programming' [Reference 7]. The word "ring" used informally in this paper will always mean a double pointer ring with a head; and as in LISP, words of memory happen to be able to hold two pointers.

FIGURE 1.2 - A Polyhedron Model of a Mechanical Arm.



I, A, Introduction to World Modeling.

I will introduce my requirements for a computer model of the physical world In terms of its role as a memory. As a memory, a world model has contents and an addressing mechanism, The kinds of data that I wish to hold in my world model are:

CONTENT REQUIREMENTS

1, Topological data,
2, Geometric data.
3, Photometric data,
4, Parts tree data,

Topological data has to do with the notion of neighborhood; a world modal has data on what is next to what. A face, edge, vertex model is essentially dedicated to surface topology; matters of volume topology are not stored but rather must be computed. Geometric data has to do with notions such as locus, length, area and volume. Photometric data includes the locus and nature of light sources, as well as data on how surfaces reflect, absorb and scatter light, Parts tree data has to do with the notion that objects are composed of Darts, which I construe as data on the structure of the physical world rather than as purely an artifact of having structured world data; that is, I prefer to have the specification of how an entity is broken into parts be external to my world model, The kinds of data not Included are semantic data (other than body names); physical data such as mass, inertia tensors, electrical properties and so on; and cultural data such as whether an object Is a toy, tool, or weapon; with any artistic, religious or market value.

Next the kinds of addressing mechanisms I wish to have, (or equivalently the input-output modes of the model) are:

ACCESSING REQUIREMENTS

1, Appearance - given a camera, return an image of what the world would look like from that camera.
2, Recognition - given an image, return the objects from the world model that appear in that image.
3, Camera Solution - given a recognized image, find the location 6 orientation of the camera:
4, Perception - given images, from solved cameras, place new bodies into the model for portions of the images that have not Yet been recognized.
5, Spatial Accessing - given a locus and radius, return the Portions of objects In that sphere,

Clearly, these are the high level accessing requirements whfoh are the reasons for having a world model and the design goals for model building,

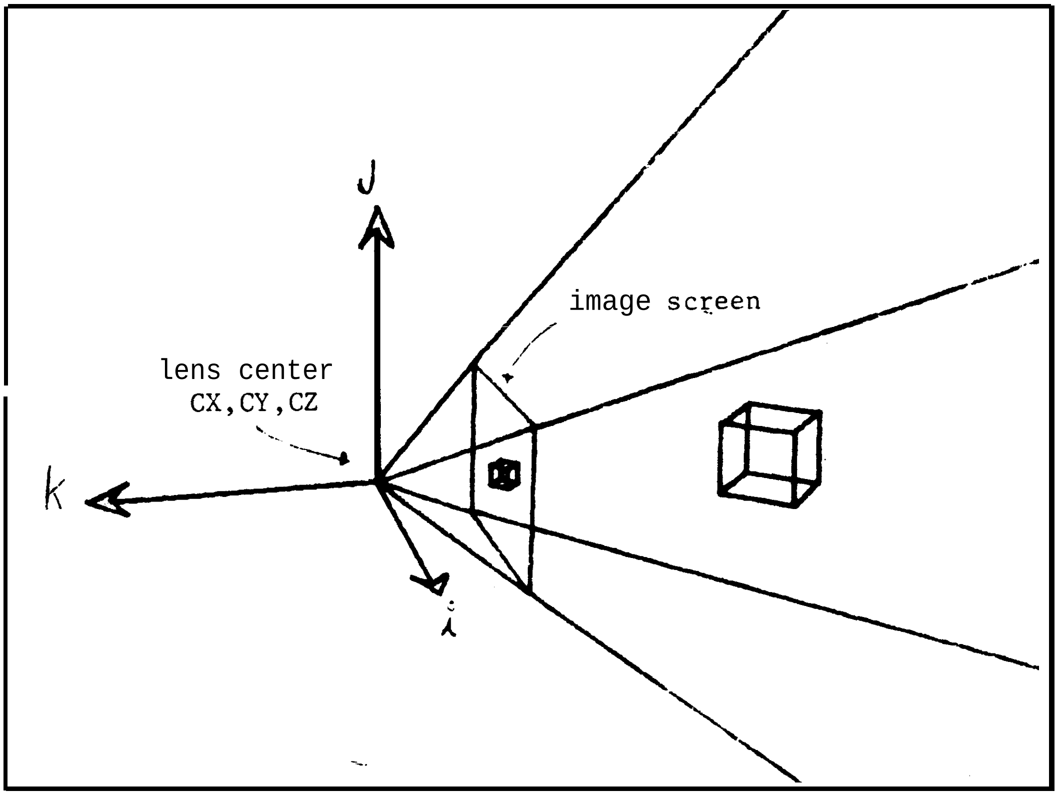
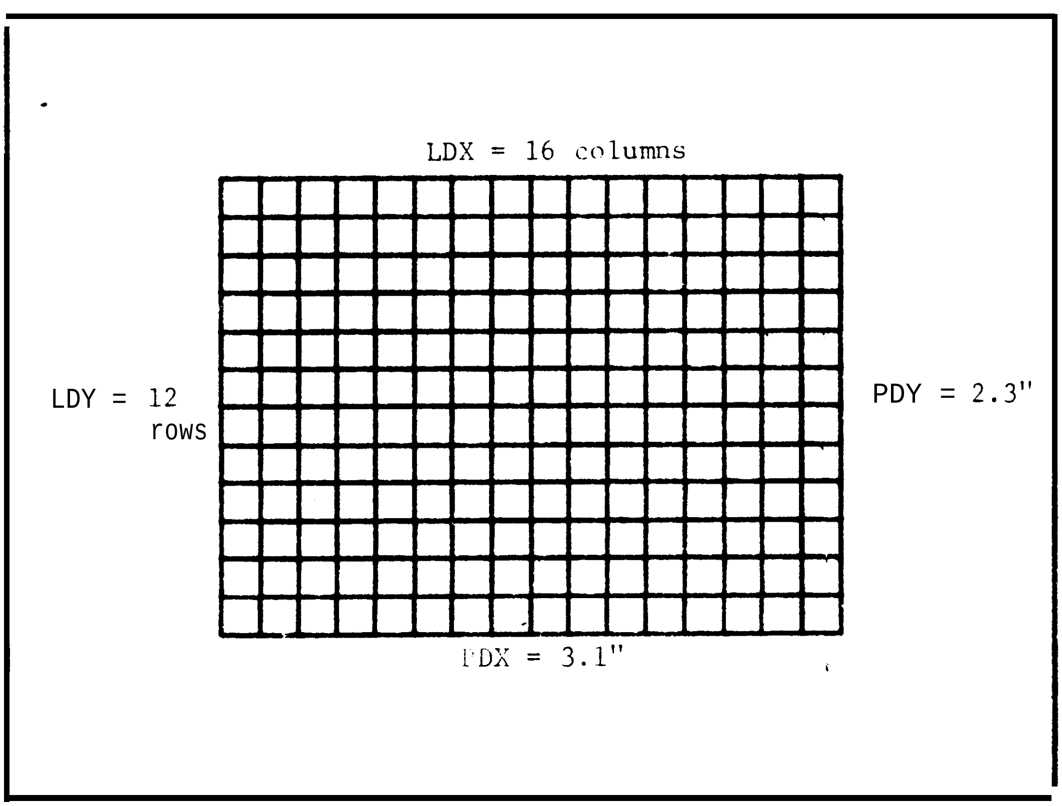


FIGURE 1.3 - A Camera Model.

FIGURE 1.4 - Logical and Physical Raster Sizes.



I. B. Introduction to a Camera Model.

As the accessing requirements imply, a world model requires a special entity called **a camera** which is used to model image formation. Although the camera model is important here for a complete specification of the data, It may be skipped on a first reading. The particular camera model I have been using lately, Is expressed by eighteen real numbers involving nine degrees of freedom. First there is the camera lens center locus:

CX, CY, CZ, In world coordinates,

Affixed to the lens center Is the camera frame of reference with unit vectors I, J and k, When the image formed by the camera is placed in correspondence to a display screen, as illustrated In figure 1.3, the unit vector I maps into the rightward positive x of the display screen, and the unit vector J maps into the upward positive y of the display screen, and the unit vector k comes out of the display screen to form a right handed coordinate system. Together the three unit vectors of the camera are the three by three rotation matrix:

```
IX, IY, IZ
JX, JY, JZ       In world coordinates,
KX, KY, KZ
```

Next, there are three scales which determine the correspondence between world size and image size. Observe that the world is measured In physical units of length like meters or feet while computer images come In integral sizes like 1024 by 1024 or 480 rows by 512 columns, thus the conversion scales must be In terms of logical image units per physical world units. In an actual television camera a minute image (say 9mm by 12mm) Is formed on a vidicon tube and that image has a particular number of rows and columns. It is the little image on the vidicon that we pretend to model by the six parameters:

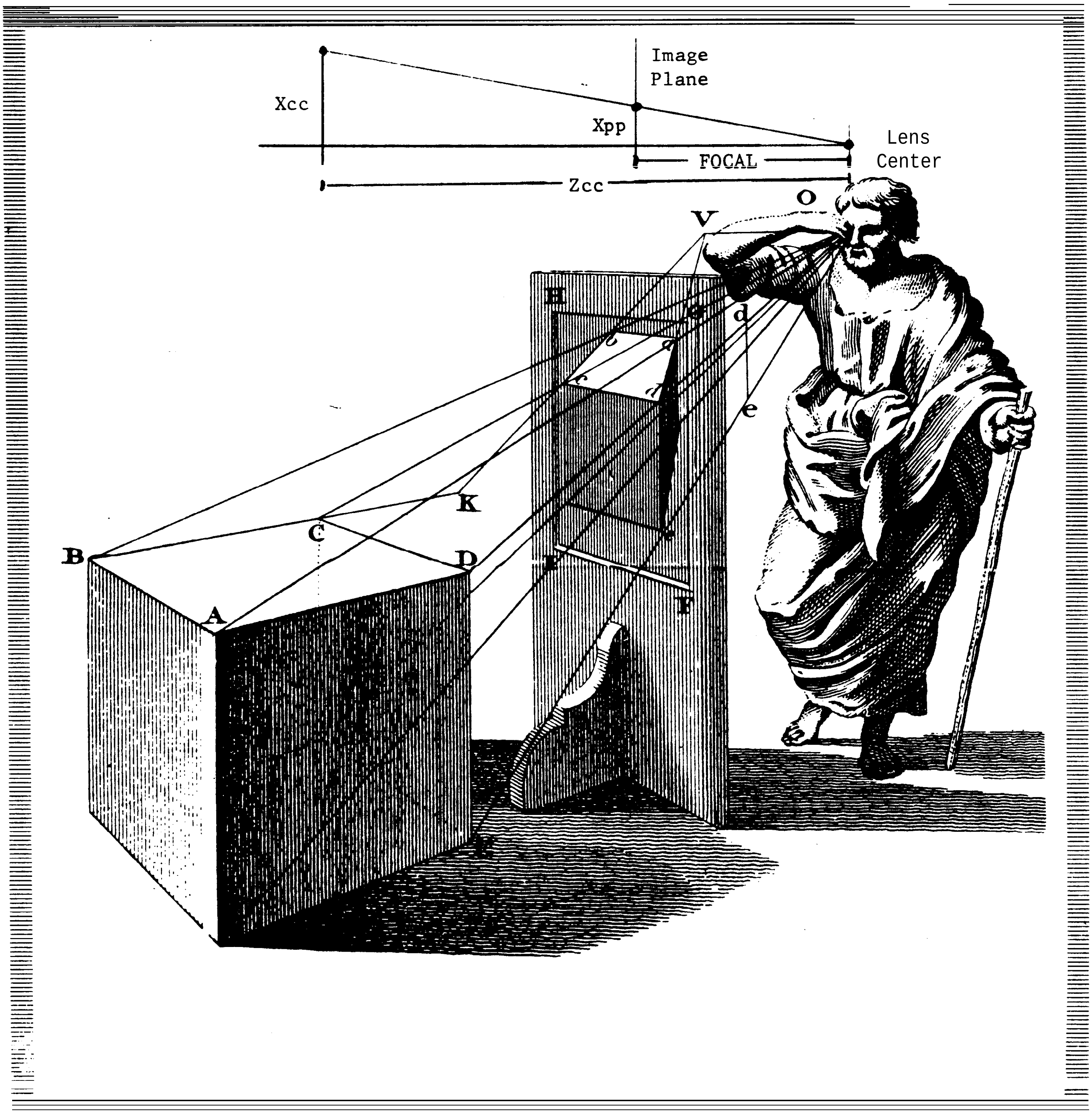
```
LDX, LDY, LDZ          Logical roster size.
PDX, PDY, FOCAL        Physical raster size.
```

Where the number named FOCAL, Is the focal plane distance which in this model (with distant objects) can safely be equated with the lens focal length and can be given In millimeters (conventional lens run 12.5mm to 75mm for 1" TV). The integer LDZ Is an artifact so that the units come out correctly in the Z dimension. Thus the scales factors are defined:

```
SCALEX ← -FOCAL*LDX/PDX;
SCALEY ← -FOCAL*LDY/PDY;
SCALEZ ←  FOCAL*LDZ;
```

This simple camera model is used to compute vertex image coordinates. A more elaborate physical camera model can *be found* In Sobel [reference 9].

FIGURE 1.5 - A Renaissance Camera Model.



I, C, Introduction to Body, Face, Edge, Vertex (BFEV) Modeling,

This introduotfon to BFEV modeling will be informal and specific to the winged edge model presented in Part-11 of this paper. Many of the basic numerical relations which make BFEV models important are stated in ALGOL notation without proof,

The Vertex,

A vertex is an instance of a point In a Euclidean three space, The Important thing about a vertex is its world locus (with component names XWC,YWC,ZWC standing for world-coordinates), Vertex locii are the basic geometric data from which length, area, volume, face vector 3 and image positions can be computed, Also a vertex may exist simultaneously in one or more image spaces, A n image space (with oomponent names XPP,YPP,ZPP standing for perspective-projected) is always three dimensional and is determined with respect to a given camera centered coordinate system (with comobnent names XCC,YCC,ZCC standing for camera-coordinates), The third image component, ZPP, is taken inversely proportional to the distance of the vertex from the cameraImage plane, ZCC, Using the camera of the previous section, The transformation of a vertex world locus to a camera centered locus is;

```
x ← xwc - CX;
Y ← YWC - CY;
Z ← ZWC - CZ;

xcc ← IX*X + IY*Y + IZ*Z;
YCC ← JX*X + JY*Y + JZ*Z;
ZCC ← KX*X + KY*Y + KZ*Z;
```

The first three assignment statements are the translation to the camera frame's origin, the second three assignments are the rotation to the camera frame's orientation, Next the perspective projection transformation IS computed(

```
XPP . SCALEX*XCC/ZCC;
YPP ← SCALEY*YCC/ZCC;
ZPP ← SCALEZ    /ZCC;
```

The XPP and YPP assignments are derived by means of similar triangles, as ising done by the man In figure 1,5; the ZPP assignment is for preserving the depth information and the colinearity of the world in the perspective projected image space, As given, the PP frame is right handed and vertices in front of the camera's image plane will have negative ZPP;ZPP values near -FOCAL are close to the camera and values approaching Zero are far away,

A final matter with respect to vertices is their valence. The valence Of a vertex is the number of edges that meet at the vertex, A vertex valence of three, for example, indicates a trihedral corner,

I. C. Introduction to BFEV Modeling,(continued).

The Edge.

For a start, the structure of an edge need be thought of as little more than two vertices; the topological subtlety of edges will be explained later, However, two vertices do define the important geometric edge data called the 2D line coefficients. Named PA, BB, cc; these coefficients are computed from the perspective locus of the edge's endpoints as follows:

$$
\begin{aligned}
AA &\leftarrow Y1 - Y2;\\
55 &\leftarrow x2 - X1;\\
cc &\leftarrow X1*Y2 - X2*Y1;
\end{aligned}
$$

These coefficients appear in the 2D equation of the line that contains the edge:

$$0 = AA*X + BB*Y + CC;$$

When the edge coefficients are normalized:

$$
\begin{aligned}
L &\leftarrow SQRT(AA\uparrow 2+BB\uparrow 2);\\
AA &\leftarrow AA/L;\\
55 &\leftarrow BB/L;\\
cc &\leftarrow CC/L;
\end{aligned}
$$

the line equation gives the distance, of a point X,Y from the line:

$$Q \leftarrow AA*X + BB*Y + CC;$$

The distance is actually ABS(Q), since Q is negative on one side side cf the line; also if one were standing on the plane at point X1,Y1 facing X2,Y2 the Q positive half-plane would be on your left and the Q negative half plane would be on your right.

An important operation on two edges is to detect whether or rot they intersect: this can be decided by checking first whether the endpoints of one edge are in the opposite half planes of the other edge, and second whether the endpoints of the latter edge are in the opposite half planes of the first. When both conditions obtain, then the intersection point can be found:

$$
\begin{aligned}
T &\leftarrow (A1*B2 - A2*B1);\\
x &\leftarrow (B1*C2 - B2*C1)/T;\\
Y &\leftarrow (A2*C1 - A1*C2)/T;
\end{aligned}
$$

An actual compare for- Intersection should initially check for the identity case, and for edges with a vertex in common.

I.C. Introduction to BFEV Modeling, (Continued),

The Face.

A face is a, finite region of plane enclosed by straight lines. A safe formal face structure could be built by defining a triangle as three non-colinear vertices and then insisting that all faces be triangle interiors. Unhappily, BFEV faces are usually represented as a list of vertices and edges (or by something nearly equivalent) for the sake of saving memory space. Such 'list' faces are not monolithic but tend to suffer special cases and pathologies such as:

Coincident or crossing edges,
Holes and Disjointness,
Concavity (& Convexity),
Non-coplanarity.

Like edges, faces have characteristic coefficients. Face coefficients satisfy the equation of a plane in which the face is embedded:

```
AA*X + BB*Y + CC*Z = KK.
```

The equation could be divided by KK, but that is undesirable because the AA, BB, CC are more useful as a unit normal vector, in which case KK is the distance of the origin from the plane. Given the locii of three non-colinear vertices, the coefficients of a plane can be computed by Kramer's rule as follows:

```
KK   ←   X1*(Z2*Y3-Y2*Z3)
       + Y1*(X2*Z3-Z2*X3)
       + Z1*(Y2*X3-X2*Y3);
AA   ← (Z1*(Y2-Y3) + Z2*(Y3-Y1)  +  Z3*(Y1-Y2));
BB   ← (X1*(Z2-Z3) + X2*(Z3-Z1)  +  X3*(Z1-Z2));
CC   ← (X1*(Y3-Y2) + X2*(Y1-Y3)  +  X3*(Y2-Y1));
```

and normalized:

```
ABC  ← SQRT(AA↑2 + BB↑2 + CC↑2);
AA   ← AA/ABC;
BB   ← BB/ABC;
CC   ← CC/ABC;
KK   ← KK/ABC;
```

If the given vertices V1, V2, V3 had been taken going counter clockwise about the face as viewed from the exterior of the solid, then the following relations obtain:

```
AA*X + BB*Y + CC*Z < KK implies X,Y,Z above the plane.
AA*X + BB*Y + CC*Z = KK implies X,Y,Z in the plane.
AA*X + BB*Y + CC*Z > KK implies X,Y,Z below the plane.
```

Face coefficients prove useful in both world and image coordinates.

I. C. Introduction to BFEV Modeling, (continued),

POLYHEDRA, BODIES and OBJECTS,

In elementary geometry8 a polyhedron is said to be a solid formed (or bounded) by plane faces; the word "polyhedron" literally meaning "many-faced", Topologically, simple polyhedra satisfy Euler's F-E+V=2 equation; where F, E and V are the number of faces, edges and vertices of the polyhedron respectively, This equation was known to Descartes in 1640, but the first proof Wasn't given Until 1752, when Euler proved the relation by considering the graph corresponding to the edges of polyhedra, A simple polyhedron is one homeomorphic to a sphere, The rigorous development of volume measure, and in turn 'solid' Polyhedra, is not simple; thus it has been easier to take the topological notion F-E+V=2 as the more primitive definition of a polyhedron on which to base a data structure and to proceed towards the appearance of 'solidness' which IS a more complicated notion. --

Counter to the usual usuage, I define the word "body" to mean an entity more speciffic than a polyhedron; the idea being that a polyhedron is represented by the whole structure of bodies, faces, edges and vertices. Bodies may have location, orientation and volume in space, Bodies may be conected to faces, edges and vertices, which may or may not form a complete polyhedron, It is typical to have only one body to a polyhedron when representing a rigid object like a sledge hammer and several bodies to a polyhedron when representing a flexible object like a man, Furthermore, the body concept is used to handle the notion of parts and abstract regional objects such as a parking lot, For example, the Stanford AI Parking Lot is represented by a body that has three parts: the Near, Mid and Far Lots, The Near Lot then has aisles and lanes and lamp Islands; a lamp Island has a curb and two lamps; a lamb has a base, stem and too, This parts structure is carried in body nodes, Finally, the word "object" will be used to refer to physical objects such as a redwood-tree, building, or roadway,

Figure 1.6
FACE PLRIMETER - a face is surrounded by edges and vertices.

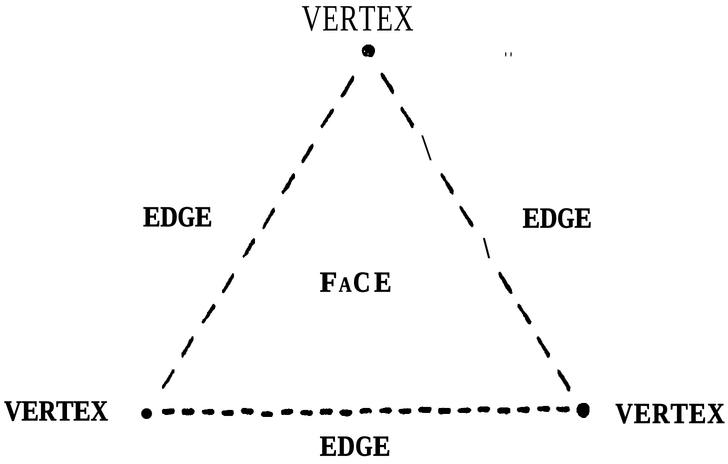


Figure 1.7
VERTEX PER IMETER' - a vertex is surrounded by edges and faces.

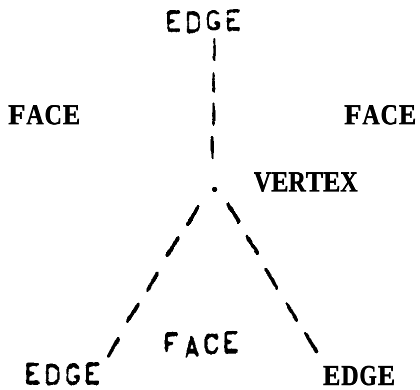
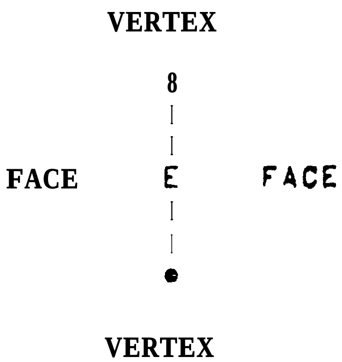


Figure 1.8
EDGE PERIMETER - an edge is surrounded by 2 faces & 2 vertices.



1. C. Introduction to BFEV Modeling, (continued).

FOUR KINDS OF BFEV ACCESSING.

1. Accessing by name and serial number.
2. Parts-Tree Accessing.
3. FEV Sequential Accessing.
4. FEV Perimeter Accessing,

A BFEV model has four kinds of accessing, The most conventional BFEV access is retrieval by symbolic name which requires a symbol table. Next, between bodies there is Parts-Tree accessing, At the top of the Parts-Tree is a special body named the world to which all the other bodies are attached; thus the world body serves as an OBLIST node, Given a particular body, a list of its sub-parts can be retrieved as well as its supra-part or "supart". A supart is the whole entity to which a part belongs, the world being its own supart.

Within each body there is face, edge and vertex sequential accessing. Given a body, all its faces, or edges, or vertices need to be readily available since perspective projection loops thtu all the vertices, and the process of display clipping loops thru all the edges, and the act of checking for body intersection loops thru all the faces. In LISP, one might provide FEV sequential accessing by placing a list of faces, a list of edges and a list of vertices on the property list of each body, so that a cube night be represented as:

```
(DEFPROP CUBE (F1 F2 F3 F4 F5 F6) FACES)
(DEFPROP CUBE (E1 E2 E3 E4 E5 E6 E7 E8 E9 E10 E11 E12)EDGES)
(DEFPROP CUBE (V1 V2 V3 V4 V5 V6 V7 V8) VERTICES)
```

Finally, among the faces, edges and vertices of a body there is perimeter accessing, Faces have a perimeter of edges and vertices [figure 1.6]; less commonly used, vertices have a perimeter of edges and faces [figure 1.7]; and of particular note, edges have a perimeter always formed by two faces and two vertices, [figure 1.8]. Perimeter accessing requires that given a face, edge or vertex, that the perimeter of that entity be readily accessible, Since the surface of a polyhedron is orientable, that is has a well defined inside and outside, (Klein bottles with their crosscaps will not be modeled), such perimeter lists can be ordered (say clockwise) with respect to the exterior of the polyhedron. Perimeter accessing is mentioned in Guzman [reference 6] and Falk [reference 4] and is the underlying basis of part-II of this paper which presents a polyhedron model built for accessing and altering face, edge and vertex perimeters,

Figure 2.1 - BASIC NODE STRUCTURE.

BODY-BLOCK	FACE-BLOCK	EDGE-BLOCK	VERTEX-BLOCK
-3, part,copart	-3,	-3,	-3, XWC
-2,	-2,	-2,	-2, YWC
-1,	-1,	-1,	-1, ZWC
Ø, type	Ø, type	Ø, type	Ø, type
+1, nface,pface	+1, nface,pface	+1, nface,pface	+1,
+2, ned,ped	+2, ped	+2, ned,ped	+2, ped
+3, nvt,pvt	+3,	+3, nvt,pvt	+3, nvt,pvt
+4,	+4,	+4, ncw,pcw	+4,
+5,	+5,	+5, nccw,pccw	+5,
+6,	+6,	+6,	+6,
5 words	2 words	6 words	5 words

Figure 2.2 - THE WINGED EDGE.
(As viewed from the exterior of a solid).

```
             \                 /
     NCCW(E)  \               / PCW(E)
               \             /
                \           /
                 \         /
                  \       /
                   . PVT(E)
                   |
                   |
  NFACE(E)         E             PFACE(E)
                   |
                   |
           NVT(E)  .
                  / \
                 /   \
                /     \
       NCW(E)  /       \  PCCW(E)
              /         \
```

Figure 2.3 - AN ACTUAL NODE STRUCTURE - SEPTEMBER 1972,

BODY-BLOCK	FACE-BLOCK	EDGE-BLOCK	VERTEX-BLOCK
-3, part,copart	-3, AA	-3, AA	-3, XWC
-2, locor	-2, BB	-2, BB	-2, YWC
-1, pname,	-1, CC	-1, CC	-1, ZWC
Ø, type,serial	0, type,serial	0, type,serial	Ø, type,seria
+1, nface,pface	+1, nface,pface	+1, nface,pface	+1, XDC,TJoint
+2, ned,ped	+2, ped	+2, ned,ped	+2, YDC,ped
+3, nvt,pvt	+3, QQ	+3, nvt,pvt	+3, nvt,pvt
+4, Fcnt,Vcnt	+4, KK	+4, ncw, pcw	+4, XPP
+5, Ecnt,Pcnt	+5,	+5, nccw,pccw	+5, YPP
+6, nbody,pbody	+6, alt,	+6, alt,pbody	+6, Zpp

PART-II. THE WINGED EDGE DATA STRUCTURE.

II, A. Winged Edge Data Structure.

Bodies, Faces, Edges and Vertices are represented by blocks of contiguously addressed Words, A single block size of ten words is adequate, A single word, like a LISP node, can hold two addresses of a floating point number, The BFEV blocks are pointed at by the address of their word numbered zero which contains control bits indicating whether the block is a body, face, edge or vertex, Figure 2,1 illustrates the block format that is being presented as an example of a winged edge data structure; a minimal number of words for each block is indicated.

The basic geometric datum is the vertex locus, which is stored in three Words of each vertex block at positions -3, -2, -1; these positions are named XWC, YWC, ZWC respectively; the letters "WC" standing for "world coordinates".

The basic topological data are the three rings of the body; (a ring of faces, a ring of edges, and a ring of vertices) and the winged edge pointers (eight such pointers in each edge block, and one such pointer in each face and vertex block), The face, edge and vertex ring pointers are stored at positions +1, +2, +3; each position has two names: NFACE, NED, NVT for the left pointers respectively; and PFACE, PED, PVT for the right, A face, edge or vertex can only belong to one body and so there is only one body node in a given face, edge or vertex ring: and that body node serves as the head of the ring, The reason for double pointer rings is for the sake of rapid deletion; other minor advantages would not justify the use of double rings,

The eight WINGED pointers of an edge block include: two pointers to the faces of that edge, two pointers to the vertices of that edge, and four pointers to the next edges clockwise and counter clockwise in each of that edge's two faces; these last four pointers are called the wings of that edge, As figure 2.2 suggests, four of these eight pointers are stored in the same positions and referred to by the same names as the face and vertex ring pointers; namely the NFACE, PFACE, NVT and PVT pointers, There are four ways in which a Pair of faces and a pair of vertices can be placed Into the two face positions and two Vertex positions of an edge; by constraining these choices two bits are implicitly encoded, one bit is called the edge parity, and the other Is called the surface parity; these bits are explained later, Finally, the single winged edge pointer found in faces and vertices is kept in the position named PED and it points to one of the edges belonging to that face or vertex.

Although the vertices in figure 2.2 are shown with three edges, vertices may have any number of edges; those Other potential edges Would not be directly connected to E and so were not shown.

A SUMMARY OF WINGED EDGE OPERATIONS.

DYNAMIC STORAGE ALLOCATION,

```
1.      Q ← GETBLK(SIZE);
2       RELBLK(Q,SIZE);
```

BFEV MAKE 6 KILL OPERATIONS,

```
1.      BNEW ← MKB(B);    KLB(BNEW);
2.      FNEW ← MKF(B);    KLF(B,FNEW);
3.      ENEW ← MKE(B);    KLE(B,ENEW);
4.      VNEW ← MKV(B);    KLV(B,VNEW);
```

FETCH LINK AND STORE LINK OPERATIONS,

```
1.      F ← NFACE(Q); F ← PFACE(Q);   NFACE.(F,Q); PFACE.(F,Q)
2.      E ← NED(Q);   E ← NED(Q);     NED.(E,Q);   PED.(E,Q);
3.      v ← NVT(Q);   V ← NVT(Q);     NVT.(V,Q);   PVT.(V,Q);
4.      A ← NCW(E);   A ← PCW(E);     NCW.(A,E);   PCW.(A,E);
5.      A -- NCCW(E); A ← PCCW(E);    NCCW.(A,E);  PCCW.(A,E);
```

WING LINK OPERATIONS,

```
1.      WING(E1,E2);
2.      INVERT(E);
```

PERIMETER FETCH OPERATIONS.

```
1.      E ← ECW(E,Q);
2.      E ← ECCW(E,Q);
3.      F ← FCW(E,V);
4.      F ← FCCW(E,V);
5.      V ← VCW(E,F);
6.      v ← VCCW(E,F);
7.      Q ← OTHER(E,Q);
```

PARTS TREE OPERATIONS,

```
1.      B ← PART(B);      B ← COPART(B);
2.      B ← BODY(Q);      B ← SUPART(B);
3.      ATT(B1,B2);       ATTACH(B1,B2);
4.      DET(B);           DETACH(B);
```

II. B. The Winged Edge Operations.

Dynamic Storage Allocation.

At the very bottom, of what is becoming a rather deep nest of primitives within primitives, are the two dynamic storage allocation functions GETBLK and RELBLK. GETBLK allocates from 1 to 4K words of memory space in a contiguous block and returns the machine address of the first word of that block. RELBLK releases the indicated block to the available free memory space. (It is sad that the machines of our day do not come with dynamic free storage). A good reference for implementing such dynamic storage, mentioned earlier, is Knuth [reference 7]. Although a fixed block size of ten or fewer words can be made to handle the BFEV entities, grandiose and fickle research applications (as well as memory use optimization) demand the flexibility of a variable block size.

BFEV Make & Kill Operations.

Just above the free storage routines are the four pairs of make and kill operations. The MKB operation creates a body block and attaches it as a sub-part of the given body. The world body always exists so that MKB(WORLD) will make a body attached to the world. In this paper, the terms 'attach' and 'detach' refer to operations on the Darts-tree linkages. The FEV make operations: MKF, MKE, MKV create the corresponding FEV entities and place them in their respective FEV rings of the given body. In the current implementation, the FEV makers set the type bits of the entity, and increment the proper total FEV counter, as well as the proper body FEV counter in the given body's node. (the Fcnt, Ecnt, Vcnt node positions are shown in figure 2.3). The kill operations: KLB, KLF, KLE, and KLV; delete the entity from its ring (or remove it from the Darts-tree), release its space by calling RELBLK, and then decrement the appropriate counters. The body of the entity is needed by the kill primitives and can be provide directly as an argument or if missing, will be found in the data by the primitive itself.

Fetch Link and Store Link Operations.

Each of the fetch link and store link operations named in the summary is a single machine instruction that accesses the corresponding link position in a node. Once BFEV nodes exist, with their rings and parts-tree already in place; the fetch and store link operations are used to construct or modify a polyhedron's surface. At this lowest level, constructing a polyhedron requires three steps: first the two vertex and two face pointers are placed into each edge in counter clockwise order as they appear when that edge is viewed from the exterior of the solid; second an edge pointer is placed in each face and vertex, so that one can later get from a given face or vertex to one of its edges; and third the edge wings are linked so that all the ordered perimeter accessing operations described below will work. Winglinking is facilitated by the WING operation.

FIGURE 2 . 4 - MIDPOINT EXAMPLE (gee text on page 2Ø).

```
                 \  pvt  /
                  \     /
        nccw       \   /    pcw
                    \ /
                  v  •
                     |
             ENEW  I
                     I nvt
             VNEW  .
                     | pvt
                 E   |
                     |
                     •
                    / \
        ncw        /   \    pccw
                  /     \
                 /  nvt  \
```

```
INTEGER PROCEDURE MIDPOINT (INTEGER E);
BEGIN "MIDPOINT"
        INTEGER B,ENEW,VNEW,V1,V2;

α CREATE A NEW EDGE AND VERTEX;
        B ← BODY(E);
        VNEW ← MKV(B);
        ENEW ← MKE(B);

α GET VERTICES AND FACES CONNECTED TO EDGES;
        PVT,(PVT(E),ENEW);
        PVT,(VNEW,E);
        NVT,(VNEW,ENEW);
        PFACE,(PFACE(E),ENEW);
        NFACE,(NFACE(E),ENEW);

α GET EDGES CONNECTED TO VERTICES;
      IF PED(V)=E THEN PED,(ENEW,V);
      PED,(ENEW,VNEW);

α LINK THE WINGS TOGETHER;
        WING(NCCW(E),ENEW);WING(PCW(E),ENEW);
        NCW,(E,ENEW);PCCW.(E,ENEW);
        PCW,(ENEW,E);NCCW,(ENEW,E);

α PLACE VNEW AT MIDPOINT POSITION;
        V1 ← PVT(ENEW); V2 ← NVT(E);
        XWC(VNEW) ← (XWC(V1)+XWC(V2)) * 0.5;
        YWC(VNEW) ← (YWC(V1)+YWC(V2)) * Ø.5;
        ZWC(VNEW) ← (ZWC(V1)+ZWC(V2)) * Ø.5;
        RETURN(VNEW);
END "MIDPOINT";
```

The Wing Link Operation.

The WING operation stores edge pointers into edges so that the face perimeters and vertex perimeters are made; and so that surface parity is preserved, Given two edges which have a vertex and a face in common, the WING operation places the first edge in the proper relationship (PCW, NCCW, NCW, or PCCW) with respect to the second, and the second in the proper relationship with respect to the first, The INVERT operation swaps the vertex, face, clockwise wing, and counter clockwise wing pointers of an edge. INVERT preserves surface parity, but flips edge parity.

The Midpoint Example.

In figure 2.4 an example of how the operations given so far could be used to code a midpoint primitive is shown, The example midpoint primitive takes an edge argument and splits it in two by making a new edge and a new vertex and by placing them into the polyhedron with the topology shown in the diagram, Then the midpoint locus is computed and the new vertex is return. The ALGOL notation used is SAIL, which allows defining the character "α" as a COMMENT delimiter and allows defining XWC as a real number from the special array named MEMORY. The MEMORY array in SAIL is the job's actual machine memory space and gives the user the freedom of accessing any word in his core image,

The Parts-Tree Operations.

As shown in figure 2.1, each body node has two parts-tree links named PART and COPART. The PART link is the head of a list of sub-parts of the body. When a body has no sub-parts the PART link is the negative of that body's pointer; that is the body points at itself, When a body has parts, the first part is pointed at by PART and the second is pointed at by the COPART link of the first and so on until a negative pointer is retrieved which indicates the end of the parts list, The negative pointer at the end of a parts list points back to the orginal body, which is the supra-part or "supart" of all those bodies in that list.

The parts may be accessed by its link names PART and COPART. Also the SUPART of a body returns the (positive) pointer to the supart of a body, The BODY operation returns the body to which a face edge or vertex belongs: this might be found by CDR'ing a FEV ring until a body node is reached, but for the sake of speed each edge (as shown in figure 2.3) has a PBODY link which points back to the body to which the edge belongs, and since each face and vertex points at an edge, the body of an FEV entity can be retrieved by fetching only one or two links.

Part Tree Operations (continued).

The parts-tree is altered by the DET(B) operation which removes a body B from Its supart and leaves it hanging free; and the ATT(B1,B2) operation which places a free body B1 into the parts list of a body B2, Since bodies are made attached to the world body and generally kept attached to something, two further parts-tree operations are provided, compounding the first two in the necessary manner, The DETACH(B) operation DET's B from Its current owner and ATT'S it to the world; and the ATTACH(B1,B2) operation will I DET B1 from its supart and attach it to a new supart. In normal (one world) circumstances one Only needs to use ATTACH to build things.

Perimeter Fetch and Store Operations.

There are seven perimeter fetch primitives, which when given an edge an3 one of it3 links will I fetch another link in a certain fashion, Using t-he winged edge data Structure these primitives are easily implemented in a few machine instructions which test the type bits and typically do one or two compares, Clockwise and counter clockwise are always determined from the outside of a polyhedron looking down on a particular face, edge or vertex, I apologize for the high redundancy on the next page, but felt that it was necessary to make the explanations independent for reference,

FIGURE 2.5 - Face Perimeter Accessing with respect to edge E.

```
VCCW(E,F) *--------E--------* VCW(E,F)
            \               /
             \      F      /
         ECCW(E,F)   ECW(E,F)
               \         /
                \       /
                 \     /
                  \   /
                    *
```

FIGURE 2.6 - Vertex Perimeter Accessing with respect to edge E.

```
                     E
                     |
                     |
      FCCW(E,V)      |      FCW(E,V)
                     * V
                    / \
                   /   \
                  /     \
                 /       \
      ECCW(E,V)           ECW(E,V)
```

The Perimeter Fetch Operations.

E ← ECW(E,F); Get Edge Clockwise from E about F's perimeter.
E ← ECCW(E,F); Get Edge Counter Clockwise from E about F's perimeter.

Given an edge and a face belonging to that edge, the ECW fetch primitive returns the next edge clockwise belonging to the given face's perimeter and the ECCW fetch primitive returns the next edge counter clockwise belonging to 'the given face's perimeter,

E ← ECW(E,V); Get Edge Clockwise from E about V's perimeter.
E ← ECCW(E,V); Got Edge Counter Clockwise from E about V's perimeter.

Given an edge and a vertex belonging to that edge, the ECW fetch primitive returns the next edge clockwise belonging to the given vertex's perimeter and the ECCW fetch primitive returns the next edge counter clockwise belonging to tne given vertex's perimeter.

f ← FCW(E,V); Get the face clockwise from E about V.
F ← FCCW(E,V); Get the face counter clockwise from E about V.

Given an edge and a vertex belonging to that edge, the FCW fetch primitive returns the face clockwise from the given edge about the given vertex and the FCCW fetch primitive returns the face counter clockwise from the given edge about the given vertex,

V ← VCW(E,F); Get the vertex clockwise from E about F.
V ← VCCW(E,F); Get the vertex counter Clockwise from E about F,

Given an edge and a face belonging to that edge, the VCW fetch primitive returns the vertex clockwise from the given edge about the given face and the VCCW fetch primitive returns the vertex counter clockwise from the given edge about the given face,

F ← OTHER(E,F); Get the other face of an edge.
v ← OTHER(E,V); Get the other vertex of an edge.

Given an edge and one face of that edge the OTHER fetch primitive returns the other face belonging to that edge, Given an edge and one vertex of that edge the OTHER fetch primitive returns the other vertex belonging to that edge,

II. C. Elaborations on Winged Edge Structure.

In this section, some variations on the basic winged edge structure are given. These variations arise as adaptations for my application, and as unimplemented ideas for improvements. The adaptations, shown in figure 2.3, include adding serial numbers and ALT links to all the faces, edges and vertices. The serial numbers provide another way of addressing and are especially useful during input and output. The ALT link is used for pointing to additional but temporary data; the most elaborate ALT data has to do with edges during a hidden line elimination. Sacrificing memory space for speed and flexibility, the face and edge coefficients are stored in each node, and the image coordinate (XPP,YPP,ZPP) and display coordinates (XDC,YDC) are added to each vertex. In elaborate systems, the image coordinates model a camera and the display coordinates refer to location on a display console. Having two tiers of image coordinates allows scrolling about the modeled image without changing the camera (or heaven forbidden, having to redo a hidden line elimination). The remaining so far unmentioned names include: the Tjoint link in vertices which is for shadow and hidden line operations, the the PD word in faces which contains photometric data, and the LOCOR and PNAME links of a body node, which point to a location-orientation matrix and an ASCII print name respectively.

Sacrificing speed for the sake of memory, the effect of having most of the extra data mentioned above can be achieved by recomputing it rather than fetching it. Furthermore, the winged data structure can be made slightly smaller by eliminating the face and vertex rings. Face and vertex sequential accessing can still be done by having two marking bits in each face and vertex, and by then going thru the edge ring looking at the two faces and two vertices of each edge for ones that are not freshly marked. It would be nice if such economizing could be done below the level of the operations.

Besides optimizations, the next improvement idea I would like to attempt would be to split the notion of a body into the two notions of a "part" and a "cell". Parts would have the parts tree and names that bodies now have, whereas a cell would have volume and face structure. In this hypothetical Cell, Face, Edge, Vertex (CFEV) model, each face could point to a cell on either side of it, the cell with the lower serial number (or something) being construed as exterior. Cell number zero would be the infinite void of three space in which everything is embedded. The trouble with CFEV is that the important matter of a polyhedron surface has to be salvaged; it can not be abandoned, because models without good surface representations can not predict appearance, which is one of my requirements.

```
SUMMARY OF POLYHEDRON PRIMITIVES.

A. EULER PRIMITIVES.

     1. BNEW ← MKBFV;             make a body, face & vertex.
     2. KLBFEV(B);                kill a body & all its pieces.
     3. VNEW ← MKEV(F,V);         make edge & vertex.
     4. ENEW ← MKFE(V1,F,V2);     make face & edge.
     5. VNEW ← ESPLIT(E);         split an edge.
     6. F    ← KLFE(ENEW);        kill face & edge leaving a face.
     7. E    ← KLEV(VNEW);        kill edge & vertex leaving an edge.
     8. V    ← KLVE(ENEW);        kill vertex & edge leaving a vertex.
     9. B    ← GLUE(F1,F2);       glue two faces together.
*   10. BNEW ← UNGLUE(E);         unglue along a seam containing E.

B. SOLID PRIMITIVES.

     1. VPEAK ← PYRAMID(F);       form a pyramid on a face.
     2. F ← PRISM(F);             form a rectangular prism.
     3. F ← CWPRISMOID(F)         form a clockwise prismoid.
     4. F ← CCWPRISMOID(F);       form a counter clockwise prismoid.
     5. ROTCOM(F);                complete a solid of rotation.
     6. FVDUAL(B);                form face vertex dual of a body.
     7. BNEW ← MKCOPY(B);         make a copy of a body.
     8. EVERT(B);                 turn a body surface inside out.
     9. B1 ← BUN(B1,B2);          form union of body interiors.
    10. B1 ← BIN(B1,B2);          form intersection of body interiors.

C. GEOMETRIC PRIMITIVES.

     1. TRANSLATE(Q,R);
     2. ROTATE(Q,R);
     3. DILATE(Q,R);
     4. REFLECT(Q,R);

D. IMAGE PRIMITIVES.

     1. PROJECTOR(CAMERA,WORLD);
     2. ELIST←CLIPER(WINDOW,WORLD);
     3. OCCULT(WORLD);
*    4. SHADOW(SUN,WORLD);
*    5. TV ← MKVID(WINDOW,WORLD);
*    6. B2D ← MKB2D(WINDOW,WORLD);
*    7. B2D ← CAREYE(TV);

* under construction, Oct 1972.
```

III, PRIMITIVES ON POLYHEDRA,

In this section a number of primitives for doing things to polyhedra are explained, Although these primitives are currently implemented using the winged edge data structure, they do not require a particular polyhedron representation, Indeed, many of these primitives were originally implemented in a LEAP polyhedron representation very similar to that of Falk, Feldman and Paul [reference 5], Thus, the primitives of this section are on a level logically independent from the operations of the previous section,

Another aspect of these primitives is that they can be used as the basis of a "graphics language" or more accurately as a package of subroutines for geometric modeling. In this vein, the primitives are currently collected as a package called GEOMES for Geometric Modeling Embedded in SAIL; and as GEOMEL, Geometric Modeling Embedded in LISP, A third language, called GEOMED, arises out of the command language of a geometric model editor based on the primitives,

The primitives are shown in four groups in the summary. The first group, the Euler Primitives, were Inspired by Coxeter's proof of Euler's formula, section 10.3 of [reference 2]. Although the proof only required three primitives, additional ones of the same ilk were developed for convenience. The second group is composed of some polyhedron primitives that were coded using the Euler primitives. The third group is for primitives that move bodies, faces, edges and vertices; or compute geometric values such as length and volume, This group is underdeveloped for two reasons: one, because I have done these computations ad hoc to date; and two, because they imply the subject of animation which is large and difficult and not of central importance to vision. With the exception of the camera, my worlds are nearly (but not absolutely) static, A less impoverished geometric group will be presented in the future, The final group, has three well developed primitives for making 2D images; and several primitives that when finished will realize part of the vision system that I am trying to build.

III. A. Euler Primitives.

As mention above, the Euler primitives are based on the Euler Equation F-E+V = 2*B-2*H; where F, E, V, B and H stand for the number of faces, edges, vertices, bodies and handles that exist. The term "handle" comes from topology, and is the number of well formed holes in a surface; a sphere has no handles, a torus has one handle, and an IBM flowcharting template has 26 handles. The Euler equation restricts the possible topologies of FEV graphs that can be polyhedra; although such Eulerian Polyhedra do not necessarily correspond to what we normally call a solid classical polyhedron. Strict adherence to constructing a polyhedron that satisfies Euler equation F-E+V = 2*B - 2*H Would require only four primitives:

		+F -E +V = 2*B - 2*H
1.	Make Body, Face and Vertex	+1 . . . . +1. . . . +1. . . . . .
2.	Make Edge and Vertex.	. . . -1 +1. . . . .
3.	Make Face and Edge.	+1 -1. . . . .
4.	Glue two faces of one body.	-2 +N -N. . . . . +1
4.'	Glue two faces of two bodies.	-2 +N -N. . . . -1. . . . . .

However, the four corresponding destructive primitives are also possible and desirable:

		+F -E +V = 2*B -2*H
1.	Kill Body, Face and Vertex	-1. . . . -1. . . . -1. . . . . .
2.	Kill Edge and Vertex.	. . . +1 -1. . . . .
3.	Kill Face and Edge.	-1 +1. . . . .
4.	Unglue along a seam.	+2 -N +N. . . . . -1
4.'	Unglue along a seam.	-2 +N -N. . . . +1. . . . . .

And finally the operation of splitting an edge at a midpoint into two edges became so important in forming T-Joints during hidden line elimination that the ESPLIT primitive was introduced in place of the equivalent KLFE, MKEV, MKFE sequence.

In using the Euler primitives, some non-classical polyhedra are tolerated as transitional states of the construction; these transitional states are called:

Seminal Polyhedron.
Wire Polyhedron.
Lamina Polyhedron.
Shell Polyhedron.
Face with Wire Spurs on its perimeter.

A seminal polyhedron is like a point; a wire polyhedron is linear with two ends like a single piece of wire; lamina and shell polyhedra are surfaces, and the picturesque phrase about spurs is a restriction on how faces are dissected into more faces. These terms will be explained in more detail when they are needed.

III,A. Euler Primitives.

1, BNEW←MKBFV; Make Seminal Body.

The MKBFV primitive returns a body with one face and one vertex and no edges, Other bodies are formed by applying primitives to the seminal MKBFV body, The seminal body is initially attached as a Part of the world.

2, KLBFEV(BNEW); Kill Body and all its pieces.

The KLBFEV primitive will detach and delete from memory the body given as an argument as well as all its faces, edges, vertices and sub-parts.

3, VNEW ← MKEV(F,V); Make an edge and a vertex.

The MKEV primitive takes a face, F, and a vertex, V, of F's perimeter and it creates a new edge, ENEW, and a new vertex, VNEW. ENEW and VNEW are called a "wire spur" at V on F, MKEV returns the newly made vertex, VNEW; ENEW can be reached since PED(VNEW) is always ENEW, Only one wire spur Is allowed at V on F at a time.

When applied to the face of a seminal body, MKEV forms the special polyhedron called a "wire" and returns the new vertex as the "negative" end of the wire. A wire polyhedron is illustrated in figure 3.1. When applied to the negative end of a wire, MKEV extends the wire; however if applied to any other vertex of the wire, MKEV refuses to change anything and merely returns its vertex argument.

Figure 3.1 - A Wire Polyhedron, Figure 3.2 - VNEW←MKEV(F,V);

```
seminal vertex  • V1
Positive en3   +|  of wire.
                | E1
               -|
                • V2
               +|
                | E2
negative end   -| of wire,
latest vertex   • V3
```

```
             +V
             /|\
            / |←←←←ENEW spur.
           /  |  \
          / -VNEW \
         /         \
        /     F     \
       /             \
      •---------------•
```

FIGURE 3.4 - TWO EXAMPLES USING EULER PRIMITIVES. (see page 30).

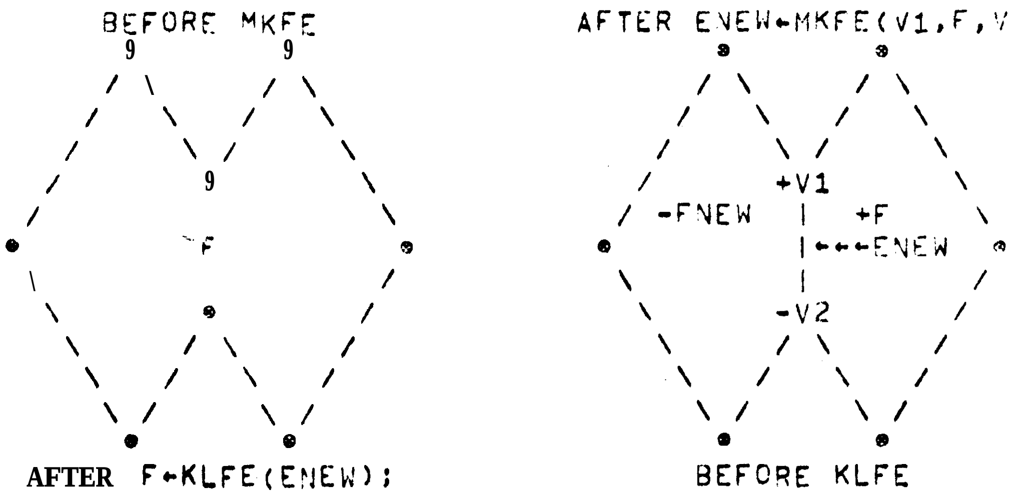
```
¤ MAKE A CUBE;
INTEGER PROCEDURE MKCUBE;
BEGIN "MKCUBE"
        INTEGER B,F,E,V1,V2,V3,V4;
¤ CREATE SEMINAL POLYHEDRON;
        B ← MKBFV;      F ← PFACE(B);  V1 ← PVT(B);
        XWC(V1)←+1;     YWC(V1)←+1;      ZWC(V1)←-1;
¤ MAKE SEMINAL POLYHEDRON INTO A LAMINA POLYHEDRON;
        V2 ← MKEV(F,V1);         XWC(V2)←-1;
        V3 ← MKEV(F,V2);         YWC(V3)←-1;
        V4 ← MKEV(F,V3);         XWC(V4)←+1;
        F  ← MKFE(V1,F,V4);
¤ MAKE FOUR SPURS ON THE LAMINA;
        V1 ← MKEV(F,V1);         ZWC(V1)←+1;
        V2 ← MKEV(F,V2);
        V3 ← MKEV(F,V3);
        V4 ← MKEV(F,V4);
¤ JOIN SPURS TO FORM FINAL FACES;
        E  ← MKFE(V1,F,V2);
        E  ← MKFE(V2,F,V3);
        E  ← MKFE(V3,F,V4);
        E  ← MKFE(V4,F,V1);
        RETURN(B);
END "MKCUBE";
```

```
¤ FORM A PYRAMID ON A FACE;
INTEGER PROCEDURE PYRAMID (INTEGER F);
BEGIN   "PYRAMID"
        INTEGER V,VØ,E,EØ,PEAK,EX;
        REAL X,Y,Z; INTEGER I;
        X←Y←Z←I←Ø;
¤ GET A VERTEX OF THE FACE AND MAKE A SPUR TO A PEAK;
        E←EØ←PED(F);
        VØ ← VCW(EØ,F);
        PEAK ← MKEV(F,VØ);
¤ CONNECT THE OTHER VERTICES OF THE FACE TO THE PEAK;
        WHILE TRUE DO
        BEGIN
                V ← VCCW(E,F);
                X←X+XWC(V);      Y←Y+YWC(V);      Z←Z+ZWC(V);
                INCREM(I);
                IF V=VØ THEN DONE;
                E ← ECCW(E,F);
                EX ← MKFE(PEAK,F,V);
        END;
¤ POSITION THE PEAK VERTEX AT THE CENTER OF THE FACE;
        XWC(PEAK)←X/I;  YWC(PEAK)←Y/I;  ZWC(PEAK)←Z/I;
        RETURN(PEAK);
END     "PYRAMID";
```

4. ENEW ← MKFE(V1,F,V2);

The MKFE primitive can be thought of as a face split. Given a face and two of its vertices, MKFE forms a new face on the clockwise side of the line V1 to V2 leaving the old face on the counter clockwise side, V1 becomes the PVT of ENEW, V2 becomes the NVT of ENEW, F becomes the PFACE of ENEW and FNEW becomes the NFACE of ENEW; also ENEW becomes the PED of F and FNEW.

Figure 3.3 - MKFE and KLFE.



MKFE is also used to join the two ends of a wire polyhedron to form a "lamina"; or the two ends Of wire spurs to split a face; Or an end of a wire Sour and a regular perimeter vertex to split a face; A "lamina polyhedron" has only two faces and thus no volume.

EULER EXAMPLES.

The use of the primitives discussed so far is illustrated by the example subroutines in figure 3.4 on page 29. The make cube subroutine starts by placing a seminal vertex at (1,1,1); Then a wire of-three edges is made using the MKEV primitive. As the code implies, MKEV places its new vertex at the locus of the old one, The ends of the wire are joined with a MKFE to form a lamina polyhedron, then a spur is placed on each of the vertices of the lamina, and finally the spurs are joined.

The pyramid example is more realistic, since polyhedra are not generated 8X nihil, but rather arise out of the vision routines and the geometric editor. PYRAMID takes a face as an argument (which is assumed to have no spurs) and runs a spur from one vertex to the middle of the faces, then all the remaining vertices of the face are Joined to that spur to form a pyramid.

III. A. Euler Primitives. (Continued).

5. VNEW ← ESPLIT(E); Edge Split.

This primitive splits an edge by making a new vertex and a new edge. Its implementation is very similar to the midpoint example on page 19. ESPLIT is heavily used in the hidden line eliminator.

6. F ← KLFE(ENEW); Kill Face Edge.

This primitive Kills a face and an edge leaving one face. Since this primitive is intended to be an inverse of MKFE, the NFACE of ENEW is killed. However the NFACE and PFACE of an edge may be swapped by using the INVERT(E) primitive. See Figure 3.3 for KLFE.

7. E ← KLEV(VNEW); Kill Edge Vertex.

This primitive kills an edge and a vertex leaving one edge. This primitive will eliminate spurs made with MKEV and midpoints made with ESPLIT; in a pure form it would have to leave vertices with a valence greater than two untouched, however it In fact "un-pyramids" them with a series of KLFE's and then kills the remaining spur.

8. V ← KLVE(ENEW); Kill Vertex Edge.

This primitive kills a vertex and an edge leaving one vertex. This primitive is the face-vertex dual of KLFE, namely instead of killing NFACE of E and fixing up PFACE's perimeter, KLEV kills the NVT of E and fixes up PVT of E's perimeter.

9. B ← GLUE(F1,F2); Glue two faces.

This primitive glues two faces together forming one new body out of two old ones (the body of F1 survives) or forming a handle on the given body. The number of edges In the two faces must be the same and their orientation should be opposite (exterior to exterior).

*14. BNEW ← UNGLUE(E); Unglue along seam. *not implemented.

This primitive unglues along the seam containing E. The UNGLUE primitive requires that a loop of edges be marked as a "seam" along which unglue will form two opposite faces. The marks are made in the temporary type bit in the edge node. Of the given body. If the cut forms two disjoint bodies then a new body is made on the NFACE side of the original E argument.

III, B, SOLID PRIMITIVES.

1, VPEAK ← PYRAMID(F);
2, F ← PRISM(F);
3, F ← CWPRISMIOD(F);
4, F ← CCWPRISMIOD(F);

These four primitives are called the "sweep primitives", because they form a simple polyhedron from a face in a fashion that appears like sweeping the face along, The sweep primitives (with the exception of PYRAMID) do not change the location of the given face but merely copy its perimeter, forming new faces and edges between the old perimeter and the new perimeter, The pyramid primitive has already appeared as an example on page 29.

Starting with a nine sided face lamina, the rocket in figure 3,6 was formed from the bottom by sweeping two prism stages, then two counter clockwise prismoid stages, and then two clockwise prismoid stages8 and finally one pyramid to form the nose cone, the fins were made by prism sweeping every third face of the first stage,

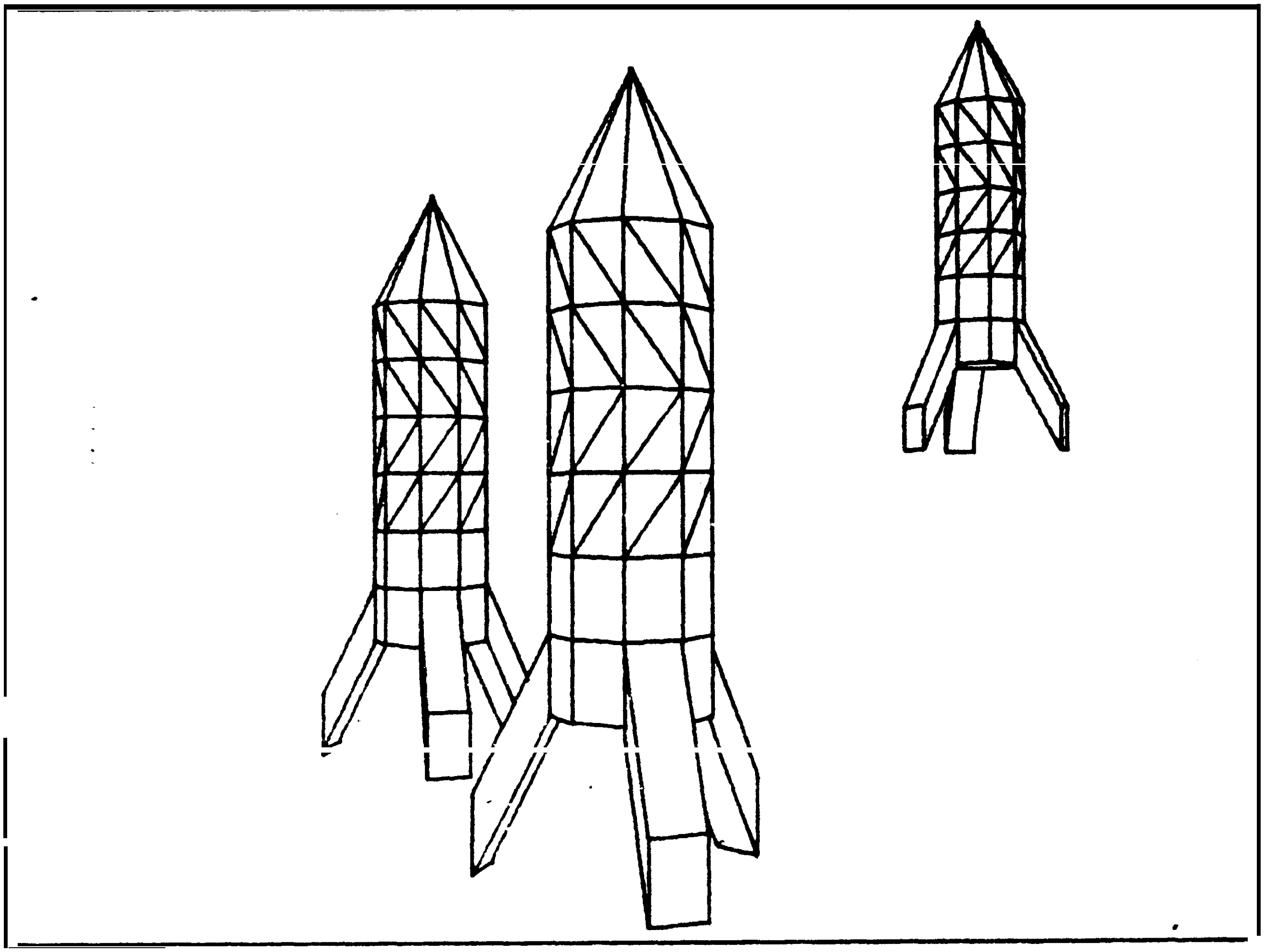


FIGURE 3.6 - Rockets made with sweep primitives.

III. B. SOLID PRIMITIVES, (continued).

5, ROTCOM(F); **Rotation Completion.**

As illustrated in the first three frames of figure 3.7 below, wire faces can be swept to form a--shell, When a wire face is swept by a sweep primitive (other than pyramid) it is marked as a shell face of rotation and its original perimeter count Is kept for later sweeps to refer to, In the third frame the shell has been positioned so that its slot can be seen, The shellface now includes all the edges of both pole caps as well as the two meridians of the slot, ROTCOM takes Such a shell face and breaks it into two polar faces and as many other faces as necessary, by means of tne count that was saved.

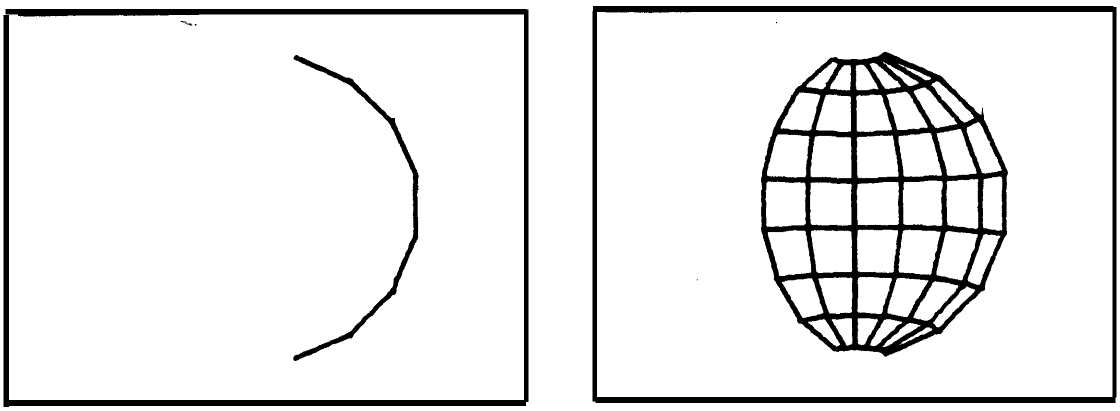
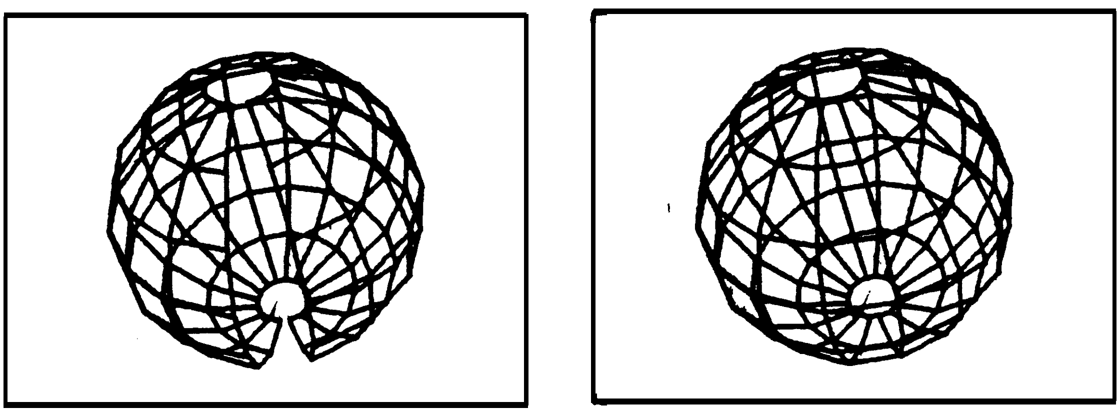
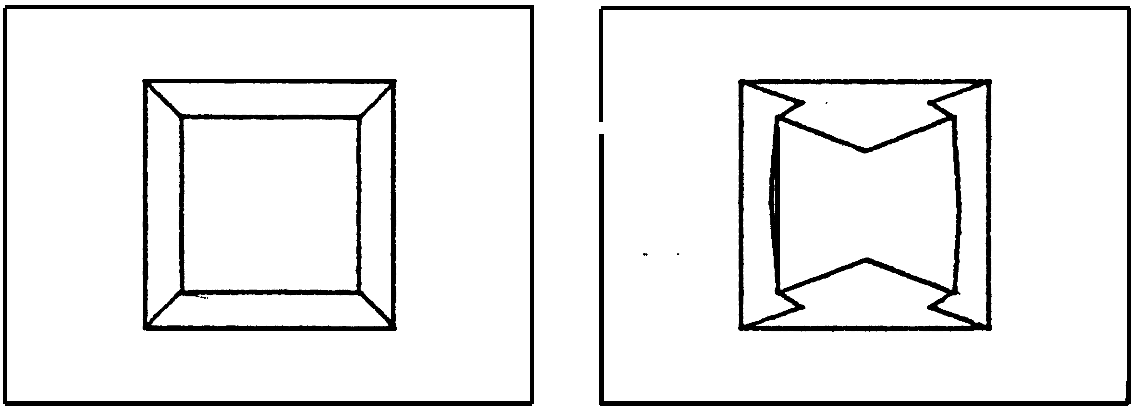


FIGURE 3.7 - Solid formed by rotation.





Euclid's construction of a dodecahedron from a cube.

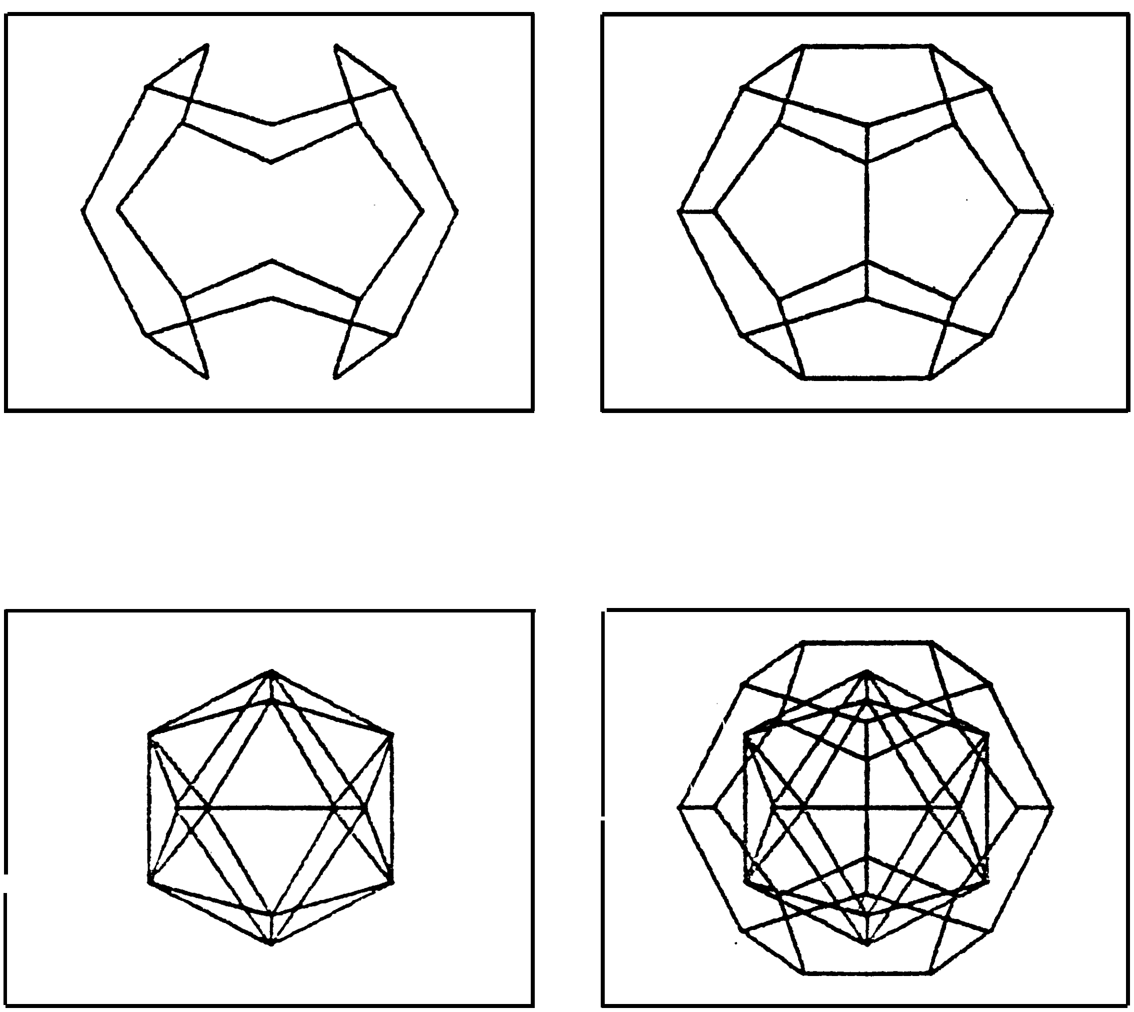


FIGURE 3.8 - Dual of a Dodecahedron.

III. B. Solid Primitives. (continued).

6. FVDUAL(B);
7. BNEW←MKCOPY(B);

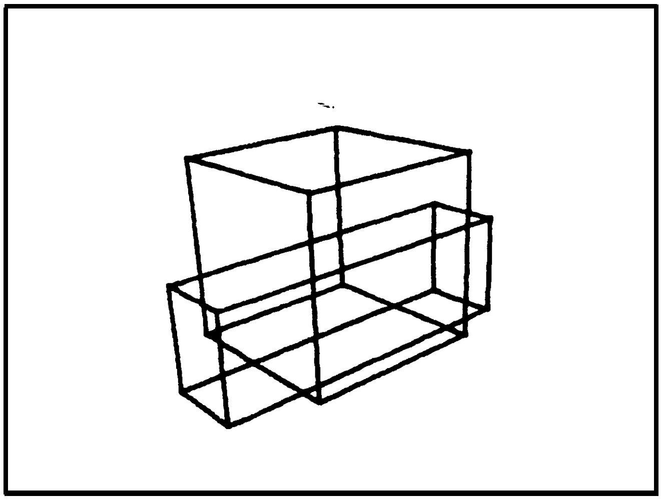
These two primitives illustrate the extremes from a class of miscellaneous primitives, FVDUAL is a worthless curosity and MKCOPY is quite useful but uninteresting. FVDUAL(B) of a body changes all the faces of a body into vertices and all the vertices into faces, in the winged edge data structure this merely requires computing a locus for each face (its center), re-"typing" faces and vertices, and then swapping the face and vertex link positions in each face, edge and vertex of the body,

Figure 3.8 illustrates Euclid's construction of a dodecahedron from a cube. The unit cube is formed, then all its edges are midpointed and translated 0.2 units into the three pairs of Parallel faces; then the midpoints are lifted 0.3 units off the plane of each face of the cube; then MKFE is applied six times to split the eight sided faces into five sided faces; giving a dodecahedron (nearly regular), Applying the FVDUAL to the dodecahedron yields the icosahedron,

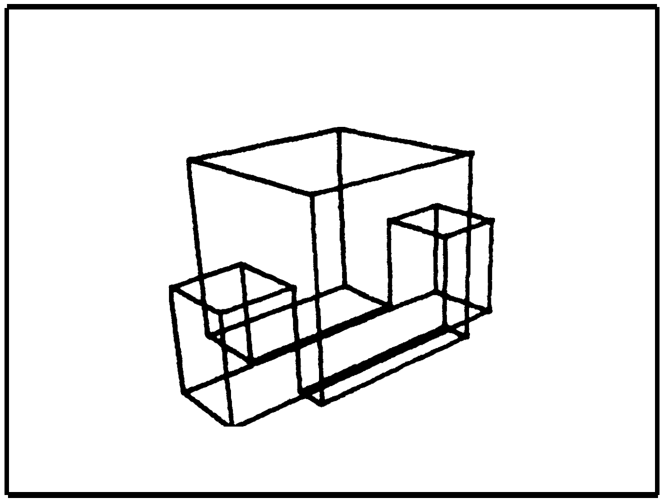
III. B. Solid Primitives. (continued).

8. EVERT(B);
9. B1←BUN(B1,B2);
10. B1←BIN(B1,B2);

These three primitives perform the Boolean operations on polyhedron interior volumes. EVERT(B) turns a body inside out, thus changing a cube into a room, as a solid into a bubble. Objects with infinite "interiors" are permissible; such polyhedra are impossible in many classical developements of solid Geometry which make the interior of a polyhedron to be the region of space with finite volume, by definition. The body union is BUN, which allows B1 to survive if the interiors of the bodies are not disjoint. A body with two disjoint polyhedrons is shunned. The body intersection is BIN, which allows B1 to survive if the interiors of the bodies are not disjoint.



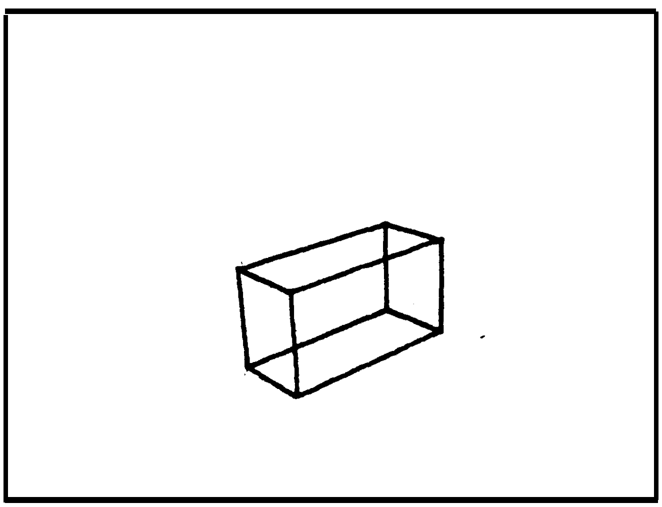
TWO BODIES



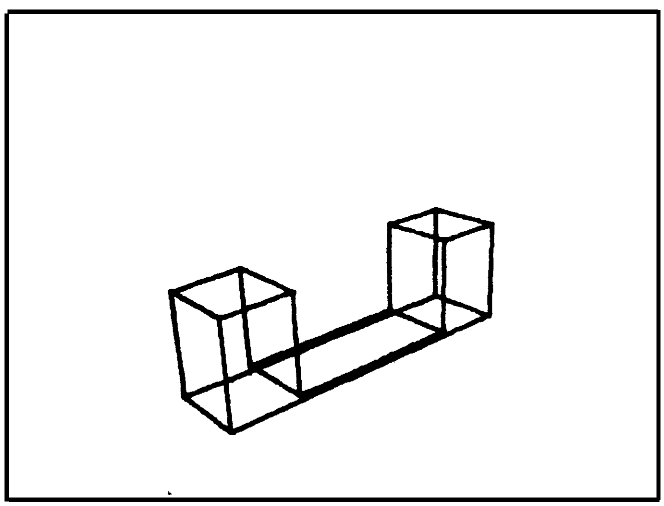
BODY UNION

FIGURE 3.9

BODY INTERSECTION



BODY SUBTRACTION



C. GEOMETRIC PRIMITIVES.

```
1. TRANSLATE(Q,R);     Q argument is a body, face, edge or vertex.
2. ROTATE(Q,R);        R argument is a transformation array with
3. DILATE(Q,R);              respect to world coordinates.
4. REFLECT(Q,R);
```

The four Euclidean transformations are translation, rotation, reflection and dilation; and as first mentioned in Klein's Erlangen Program, 1872, these four primitives form a group. The primitives may be applied to bodies, faces, edges or vertices in order to change vertex world locii. Thus a body is the set of vertices in its vertex ring, a face is the set of vertices on its perimeter, an edge is the two vertices which are its ends, and a single vertex is itself; but there are special cases having to do with faces. (In GEOMED a special counter, negative Fcnt, is maintained in wire sweep faces in order to make solids of rotation). The second argument R is a pointer to a transformation array in world coordinates of four rows and three columns:

```
XWC, YWC, ZWC
IX,  IY,  IZ
JX,  JY,  JZ
KX,  KY,  KZ
```

For translation, only the XWC, YWC and ZWC are involved and all the vertices are translated in the obvious fashion:

```
X ← X + XWC;     Y ← Y + YWC;     Z ← Z + ZWC;
```

whereas for rotation (dilation and reflection) the innermost computation applied to each vertex is:

```
X ← X + XWC;     Y ← Y + YWC;     Z ← Z + ZWC;
XX ← IX*X + IY*Y + IZ*Z;
YY ← JX*X + JY*Y + JZ*Z;
ZZ ← KX*X + KY*Y + KZ*Z;
X ← XX - XWC;    Y ← YY - YWC;    Z ← ZZ - ZWC;
```

At this point, I should now present a few general primitives for setting up such transformation arrays, but I don't have them yet. The problem involves selecting frames of references, strength of transformation, axes of transformations, origins of frames and modes such as absolute, relative or interpolated. At present in my applications these matters are handled ad hoc (the most general solution being the ROTDEL and EUCLID subroutines of GEOMED). The heart of deriving a transformation array is to get a frame of reference REF and an amount of rotation DEL and to compute the matrix product:

```
R ← (transpose(REF)cross(DEL cross REF));
```

For dilation (larger or smaller) cross DEL with a non-unity diagonal matrix; for reflections flip the row signs on desired axes.

D. IMAGE PRIMITIVES.

1. PROJECTOR(CAMERA,WORLD);
2. ELIST←CLIPER(WINDOW,WORLD);
3. OCCULT(WORLD);
* 4. SHADOW(SUN,WORLD);
* 5. TV ← MKVID(WINDOW,WORLD);
* 6. B2D ← MKB2D(WINDOW,WORLD);
* 7. B2D ← CAREYE(TV);

* under construction, Oct 1972.

PROJECTOR computes the perspective projected locus of all the vertices in a Given world from a given camera. CLIPER computes the portions o f 3D lines that are visible within a given display window. OCCULT compares all the edges, faces and vertices in a given world; using their current projected coordinates; faces, edges and vertices that are not visible from the implied camera's viewpoint are marked as hidden; faces, edges and vertices that are visible are marked as visible; and faces, edges and vertices that were initial ly partially visible are broken up into visible and hidden portions. The new faces, edges and vertices introduced by OCCULT are marked so that they can be removed,

Ths following four primitives are sti I I be ing developed. SHADOW will literally build a world with shadows in it; shadow calls OCCULT twi ce, once for the SUN and once for the camera, There Is no conceptual difficulty in doing many point sources, but I shall get one source working at a time. The MKVID primitive generates TV intensity fasters from the world model after OCCULT or SHADOW has been applied. The MKB2D primitive generates a 2D data structure of regions and edges (which is almost a copy of the 3D structure that has been presented, but with special attention paid to T-joints); this B2D data structure is an image model. Final ly, the CAREYE primitive converts TV Intensity rasters into B2D image structure. A detailed discription o f these image primitives can not be given at this time (OCT 1972), because I haven't finished making them,

IV. APPLICATIONS.

The single application around which the geometric modeling of this paper is being built is for a computer television vision (TVV ?) system for looking at real world scenes. I believe that a computer must have a means of representing what it is intended to see and further that the representation must have (in principle) an inverse relation to a television image. My first premise is rarely questioned, the second premise is frequently questioned. One alternative position is that so called "features" can be extracted from an image and then used by a heuristic problem solver to find an association between the perceived features and previous general knowledge; it is then stated that there is no need to go from the general knowledge or even from the so called image "features" back down to a television image, even just in principle. I wish to state the opposite, there is a need to go from the general representation to a television image in order to develop computer vision without having to solve several other problems of Artificial Intelligence. Applications of geometric modeling other than television vision might include: architectural drawing, computer animation, and modeling for laser, radar, and sonar image systems.

IV. A. Modeling: GEOMED - a drawing program.

GEOMED, Geometric Model Editor, is for making and editing polyhedra. The command language of GEOMED provides the Euler primitives in the context of a push down stack (the PADPDL) of bodies, faces, edges and vertices. The main difference between an interactive program and a programming language being that the former carries along a working context so that most arguments and data do not have to be explicitly named.

V	-	make seminal vertex body,
E	-	make edge and vertex,
J	-	make edge and face,
G	-	glue two faces.

In addition to the stack, GEOMED provides frames of reference for the Euclidean transformations; there is a world frame, body frames, camera frames, relative frame and face frames. Also the strength of a Euclidean transformation can be halved or double, set directly or entered numerically in several kinds of units. And finally the transformation can be done once or repeatedly by keying chords of Stanford's extra shift keys named "control" and "meta" with a.; : 0 - or * character. These characters are not mnemonics but were chosen because of thier positions on the keyboard.

: ;	-	transform about X-axis.
)	-	transform about Y-axis.
- *	-	transform about Z-axis.

		no shifts	- TRANSLATION,
a	-	control	- ROTATION,
β	-	meta	- DILATION.
ε	-	meta-control	- REFLECTION,

Finally, GEOMED provides access to all the solid primitives and hidden line elimination, along with commands for the stack, input, output, display, and switch toggling. The commands are detailed in the operating note, SAILON-68, along with a list of GEOMES and GEOMEL subroutines. Two examples should suffice to illustrate how concise and illegible GEOMED command strings are:

```
1.   V:)\E;E(E:J↑/*S-↑                 forms a cube,
2.   V:@E*E*E*E*E*E*E*J↑!
     \\:@S)S)S)S)S)S)S)S)G↑            forms a torus.
```

Thus a polyhedron can be represented in a few characters (which must be compiled); one might hope that such a "representation by generation" could provide a link between semantic and geometric models. The hard direction is to get from a polyhedron model to the command string.

IV. B. Graphics: OCCULT - a hidden line eliminator.

OCCULT is a hidden line eliminator; it is neither a Watkins nor a Warnock algorithm but is rather a throw-back to the naive idea of comparing each edge with all the other edges and having ways to dampen the potentially large number of comparisons that might occur.

There are three kinds of dampening in OCCULT. The first (used in other hidden eliminators) is to get rid of the faces that have their backs to the camera and to consider for comparision only the edges with one potentially visible face. These edges are called "folds". The second kind of dampening, is to hide everything connected to the hidden portion of an edge when a fold crossing is discovered, this is made possible by the winged edge primitives which allow Polyhedron surfaces to be easily traversed topologically; and by the Euler primitives which allows the edges to be quickly broken into visible and hidden portions without losing their topology. The third kind of dampening Involves having a raster of edge buckets to localize the comparisons.

The reason for doing hidden line elimination in this fashion is to get the topology of the image regions and edges in a modeled scene including the shadows. OCCULT was used to make some of the figures that appeared earlier in this paper; for example the arm model in figure 1.2, (which required twelve seconds of PDP-10 compute time). A paper on OCCULT should be available before the end of the year, 1972.

IV. C. Vision: CAREYE - a video region-edge finder.

CAREYE, Cart Eye, is the oldest, most rewritten, yet least finished part of the application. At present its best trick is to take a Television image and convert it into video intensity contour lines similar to those discussed by Krakaur and Horn (of M.I.T.). From VIC, Video Intensity Contours8 the image goes through two processes: first, the camera locus-orientation for the image is solved by finding feature points In the image that coorespond with known land mark point in the world; and second, after the camera is solved, the locus cf previously unknown regions of the image must be added to the world model; the third dimension of such unknown regions being assumed to be very large, until evidence is found in succeeding images that make the region "pop out" of the background. These two processes are called Camera Locus Solving and Body Locus Solving; CAMLS and BODLS; and are the missing links in making polyhedron models merely by looking at objects and scenes of objects.

References:

1. AGIN
Representation and Description of Curved Objects
Stanford Artificial Intelligence AIM-173, 1972.

2. COXETER
Introduction to Geometry
John Wiley & Sons, Inc. New York, 1961.

3. EVES
A Survey of Geometry.
Allyn and Bacon, Inc, Boston, 1965.

4. FALK
Computer Interpretation of Imperfect Line Data
as a Three Dimensional Scene,
Stanford Artificial Intelligence AIM-132, 1970.

5. FELDMAN, FALK & PAUL
Computer Representation of Simply Described Scenes.
Stanford Artificial Intelligence SAILON-52, 1969.

6. GUZMAN
Computer Recognition of Three Dimensional Objects,
Project MAC Technical Report, 1968,

7. KNUTH
The Art of Computer Programming,
Volume 1 - Fundamental Algorithms.
Chapter 2 - Information Structures.
Addison-Wesley, Reading, Mass. 1968,

8. ROBERTS
Machine Perception of Three Dimensional Solids
Lincoln Laboratory Technical Report #315, 1963.

9. SOBEL
Camera Models and Machine Perception.
Stanford Artificial Intelligence AIM-121, 1970.