A polyhedron representation for computer vision

by BRUCE G. BAUMGART
Stanford University
Stanford, California

USE OF POLYHEDRA IN COMPUTER VISION

My approach to computer vision is best characterized as inverse computer graphics. 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 analyzed to compute detailed geometric models.

For example, the polyhedron in Figure 1 was computed from views of a plastic horse on a turntable by intersecting silhouette cones. As such, silhouette cone intersection is a purely descriptive vision technique; it is a form of wide angle stereo reconstruction. Like in the joke about carving a statue by cutting away everything that does not look like the subject, the approximate shape of the horse is hewed out of 3-D space by cutting away everything that falls outside of the silhouettes. In the example, the model was made from three silhouettes of the horse facing to the left which may be compared with two views of the horse facing to the right. One of the views is a real video image and the other is a display of the result showing how the process automatically constructed a backside for the horse consistent with the given silhouettes.

The present implementation requires a favorably arranged viewing environment (white objects on dark backgrounds or vice versa); application to more natural situations will be possible when a bulk correlation processor (an SPS-41) becomes available for extracting silhouettes by stereo depth discontinuities. Furthermore, the restriction to turntable rotation is for the sake of easy camera solving; this restriction will be lifted by providing stronger feature tracking for camera calibration. The silhouette cone intersection method can 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. 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.

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. In Figure 2 for example, the approximate video appearance of the machine parts schematically depicted (top) can be computed and analyzed for edges (middle) and compared with an edge analysis of an actual video image of the parts (bottom). 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. Visually acquired 3-D geometric models can be of use to other robotic processes such as manipulation, navigation or recognition.

Unfortunately, these two approaches to computer vision (descriptive vision and verification vision) are only as strong as the state of the art in 3-D computer graphics. Consequently, my recent vision work has been largely concerned with the representation and manipulation of 3-D objects; objects which are solid, opaque and rigid. Although there are several significantly different geometric modeling ideas: arrays, 3-D density functions, 2-D parametric functions, volume elements, cross sectional elements, skeletons, manifolds and polyhedra; I have concentrated on polyhedra because they are simple enough to readily handle in a computer and complex enough to represent an arbitrary opaque surface. The rest of this paper is devoted to presenting a particular polyhedron representation for which convenient sets of manipulation routines have been developed.

INTRODUCTION TO THE WINGED EDGE

The Winged Edge polyhedron 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.[1] 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

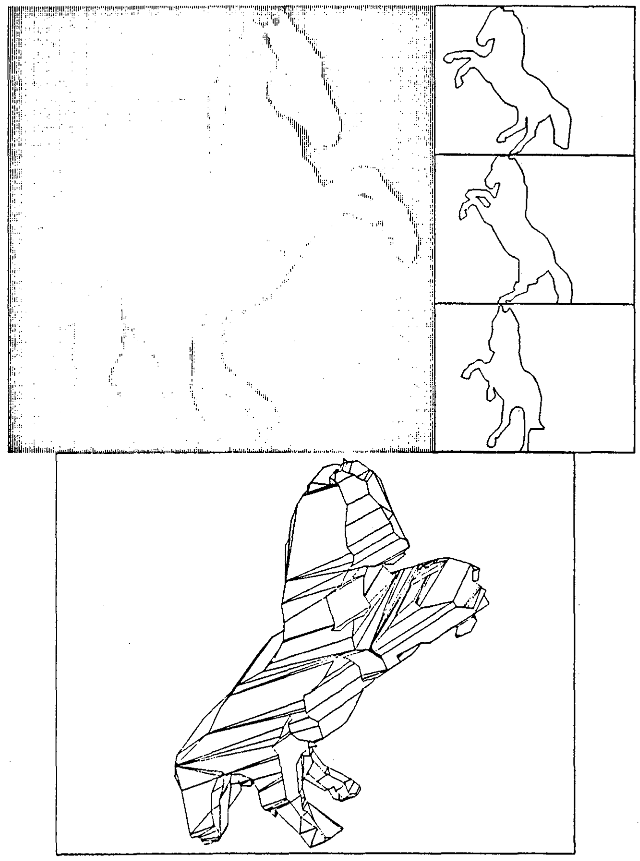


Figure 1—An example of silhouette cone intersection

NFACE(edge) and PFACE(edge). Although my latest language of implementation is PDP-10 machine code, examples will be given in a fictional programming language which combines ALGOL with Knuth's node/link notation.

A polyhedron is 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 to 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 counterclockwise 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.

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 Figures 3 and 4. 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 3 diagrams the arrangement of the links comprising the topology of an edge of a polyhedron

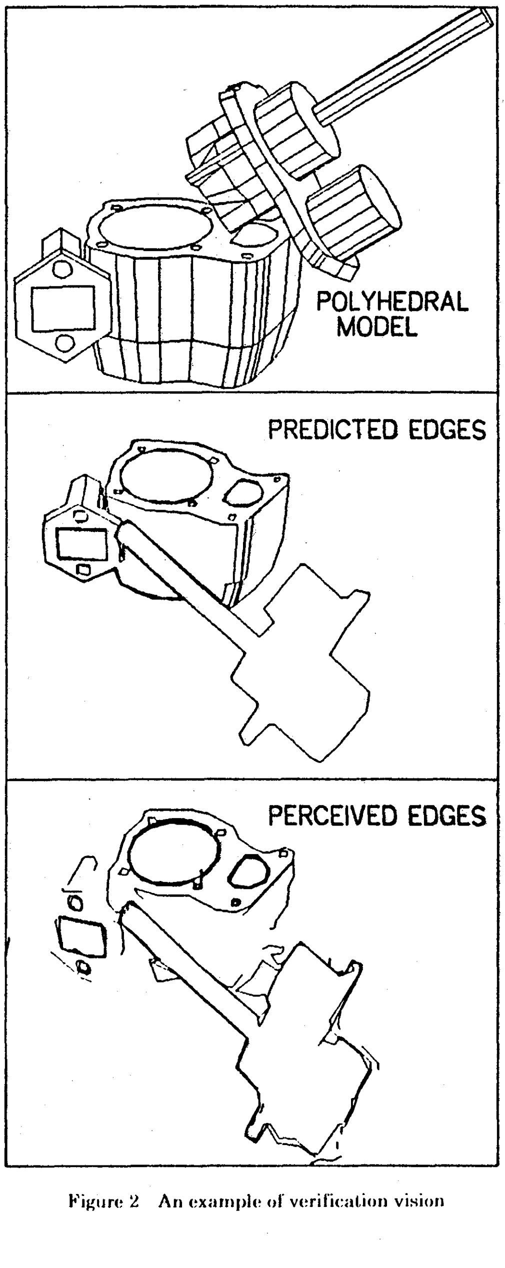
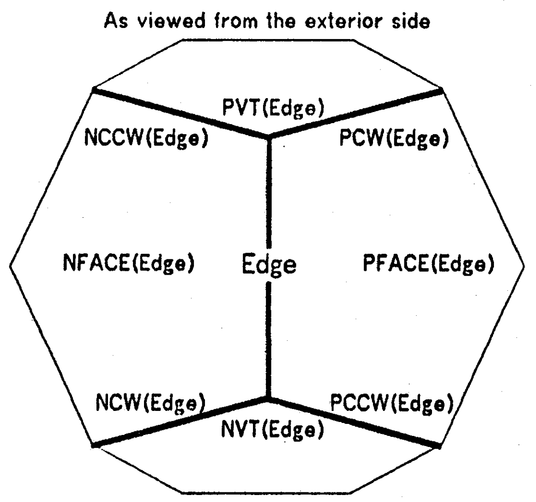


Figure 2 An example of verification vision

as viewed from the exterior side of its surface. Although the vertices in the figure 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 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 Figure 4 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 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.



1. Face ring of a body:
   NFACE(body or face) & PFACE(body or face).
2. Edge ring of a body:
   NED(body or edge) & PED(body or edge).
3. Vertex ring of a body:
   NVT(body or vertex) & PVT(body or vertex).
4. First edge of a face or vertex:
   PED(vertex) or PED(face).
6. The two faces of an edge:
   NFACE(edge) and PFACE(edge).
7. The two vertices of an edge:
   NVT(edge) and PVT(edge).
8. The four wing edges of an Edge:
   NCW(Edge) edge of NFACE Clockwise from Edge.
   PCW(Edge) edge of PFACE Clockwise from Edge.
   NCCW(Edge) edge of NFACE CCW from Edge.
   PCCW(Edge) edge of PFACE CCW from Edge.

Figure 3 Winged edge topology

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.

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

BODY NODE FORMAT

The body node is the head of the face, edge and vertex rings which use words 1, 2, and 3. The body node carries a parts tree structure in words 4 and 5. There is a print name of up to ten characters carried in words -2 an -1. The links of the 8th word are always left free for linkage to user data structures.

-3	TMP		
-2	PNAME1		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.
5	BRO	SIS	Parts Tree.
6	alt	TRAM	Body TRAM.
7	CW	CCW	Body ring.
8	nlnk	plnk	User links.

EDGE NODE FORMAT

The main fields of the edge are explained in the text. The negative three words are used for edge coefficients and for clipped display coordinates. The alt, alt2 and cw fields are used as temporaries. The CCW field points at the body of edge and expedites BGET. The nlnk and plnk fields are kept empty for users.

-3	x1dc AA y1dc		Display Coord. or
-2	x2dc BB y2dc		2-D Edge Coef. or
-1	CC		3-D line Cosines.
0	STATUS BITS		
1	NFACE	PFACE	Two Faces.
2	NED	PED	Edge ring.
3	NVT	PVT	Two vertices.
4	NCW	PCW	Clockwise Wings.
5	NCCW	PCCW	CCW Wing Edges.
6	alt	alt2	Temporaries.
7	cw	ccw	Temporaries.
8	nlnk	plnk	User links.

FACE NODE 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
-2	BB		normal
-1	CC		vector.
0	STATUS BITS		
1	NFACE	PFACE	Face ring.
2	Ncnt	PED	First edge.
3	KK		Distance to origin
4	red grn	blue wht	Reflectivities.
5	Lr Lg Lb	Lw Sm Sn	Lumns.& Spec.Coef.
6	alt	alt2	Temporaries
7	QQ		Video Intensity.
8	nlnk	plnk	User Links.

VERTEX NODE FORMAT

The vertex node contains locus in three forms: 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 Locus.
2	Tjoint	PED	First Edge.
3	NVT	PVT	Vertex ring.
4	XPP		Perspective
5	YPP		Projected Locus.
6	alt ZPP alt2		
7	cw	ccw	Temporaries.
8	nlnk	plnk	User links.

Figure 4—Example of winged edge node formats

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 5. One virtue of the winged edge representation is that both vertex and face perimeters can be traversed in either direction (clockwise or counterclockwise) while being dynamically maintained in "*one ring*".

Given one edge of a face (or vertex) perimeter, the next edge clockwise (or counterclockwise) 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 EØ=(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 counterclockwise from a gvien edge with respect to a given vertex; VCW and VCCW which return the vertex clockwise or counterclockwise 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 or a polyhedron. With node type checking the primitives can be made stronger, for example ECCW(vertex,face) is implemented to return the edge counterclockwise 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.

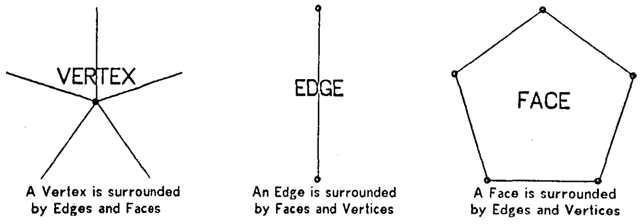

VERTEX
EDGE
FACE
A Vertex is surrounded by Edges and Faces
An Edge is surrounded by Faces and Vertices
A Face is surrounded by Edges and Vertices

Figure 5—Three kinds of perimeters

Two remaining 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) or F=NFACE(E); (iii) Edge-Edge, returns a common vertex or false; (v) Edge-Vertex, returns Boolean value V=PVT(E) or V=NVT(E); (vi) Vertex-Vertex, returns common edge or FALSE. (As in LISP, zero is false and nonzero is true).

BASIC POLYHEDRON SYNTHESIS

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 above are the bedrock for the Euler primitives, which are an elegant set of subroutines for altering polyhedra while always maintaining the Euler relation: $F-E+V=2*B-2*H$ between the numbers of bodies, faces, edges, vertices and handles. Examples of Euler primitives are given in another paper written for this conference[2] as well as Section 3 of Reference 3 and so will not be elaborated here.

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. As a Cartesian frame of reference the tram node is interpreted as a 3-D locus in world coordinates with a right handed triad of orthogonal unit vectors; as a Euclidean transformation the tram node is interpreted as a translation vector followed by a rotation matrix. Tram nodes are further explained in Reference 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 that 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).

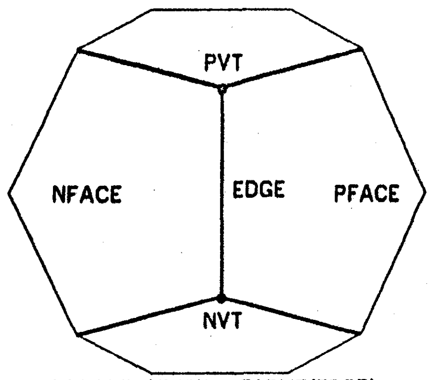
```
PROCEDURE WING(INTEGER E1,E2);
BEGIN
  IF PVT(E1)=PVT(E2)∧PFACE(E1)=
    NFACE(E2) THEN BEGIN PCW(E1)←E2;
    NCCW(E2)←E1;END;
  IF PVT(E1)=PVT(E2)∧NFACE(E1)=
    PFACE(E2) THEN BEGIN NCCW(E1)←E2;
    PCW(E2)←E1;END;
  IF PVT(E1)=NVT(E2)∧PFACE(E1)=
    PFACE(E2) THEN BEGIN PCW(E1)←E2;
    PCCW(E2)←E1;END;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

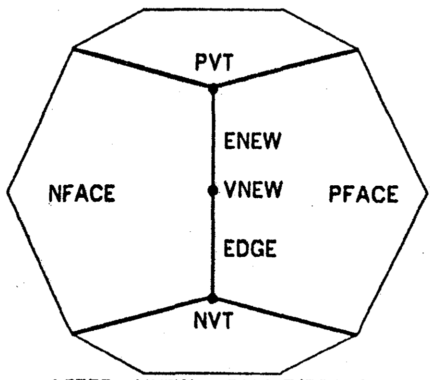
Body nodes can be grouped into a tree structure of parts. The parts tree consumes four link positions (DAD, SON, BRO, SIS) and is maintained in body nodes by the following primitives: BDET(body) detaches a body node from the parts tree, BATT(body1,body2) attaches body1 to the ring of children belonging 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.

EDGE AND FACE SPLITTING

The most important property 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. The edge



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";
```

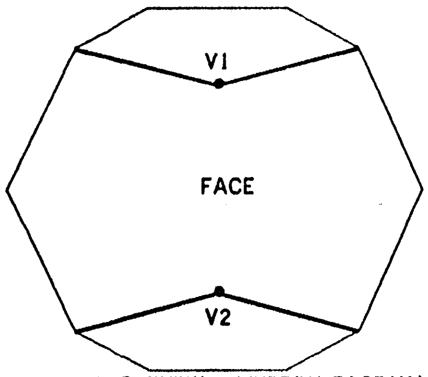
```
INTEGER PROCEDURE KLEV (INTEGER VNEW);
BEGIN "KLEV"
        INTEGER EDGE,ENEW,V,F,B;
        ENEW ← PED(VNEW);
        EDGE ← ECCW(ENEW,VNEW);
COMMENT ORIENT EDGES AS IN DIAGRAM;
        IF NVT(ENEW) ≠ VNEW THEN INVERT(ENEW);
        IF PVT(EDGE) ≠ VNEW THEN INVERT(EDGE);
COMMENT TIE E TO ITS NEW 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.

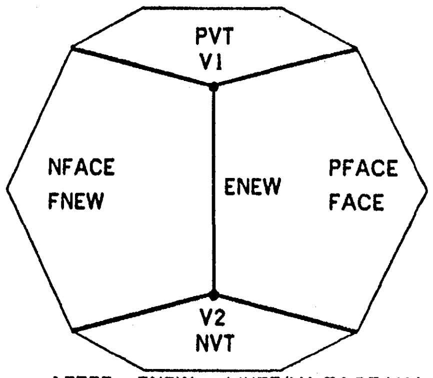
Figure 6—Make and kill edge-vertex

split routine, ESPLIT, makes a new edge and a new vertex and places them into the surface topology as shown in Figure 6; 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 7; the kill face-edge routine, KLFE, undoes a MKFE.

The rest of this section concerns implementation, the use of the split and kill routines illustrate a pattern which applies to the coding of any operations on 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, NTV, 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.



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.

Figure 7—Make and kill face-edge

CONCLUSION

The technical point of this paper is that a polyhedral representation with a coherent and locally alterable topology can be constructed. The larger philosophical point is that computer vision perhaps can be realized by using computer graphics techniques to keep an internal mental simulation in sync with the changing appearance of the external physical reality.

REFERENCES

1. Knuth, Donald Ervin, *The Art of Computer Programming*, Addison-Wesley, Reading, Massachusetts, 1968.
2. Eastman, Lividini & Stoker, "A Database for Designing Large Physical Systems," *Proceedings of the National Computer Conference*, May 1975.
3. Baumgart, Bruce G., *Geometric Modeling for Computer Vision*, Stanford Artificial Intelligence Laboratory, Memo no. AIM-249, Stanford University, October 1974.