A PL1 COMPUTER PROGRAM FOR HIDDEN LINE ELIMINATION

By

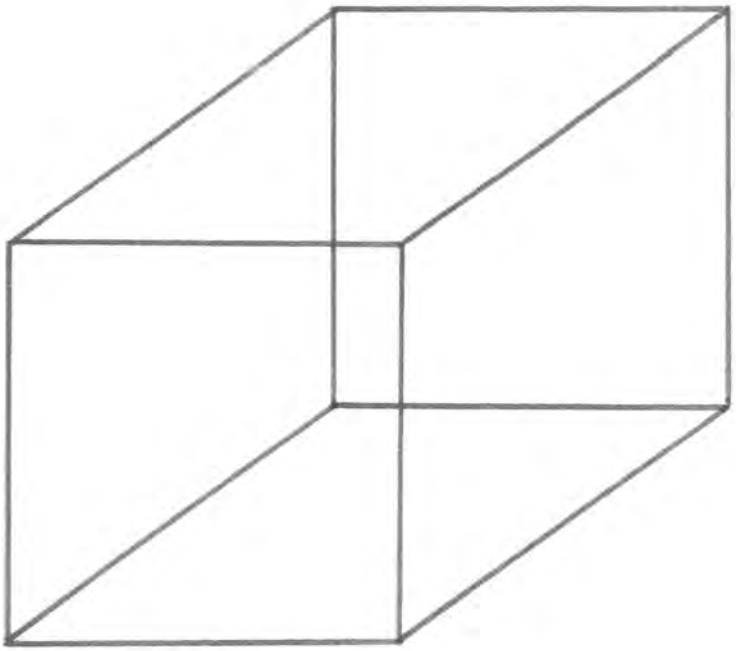
Dennis M. Pfister

TABLE OF CONTENTS

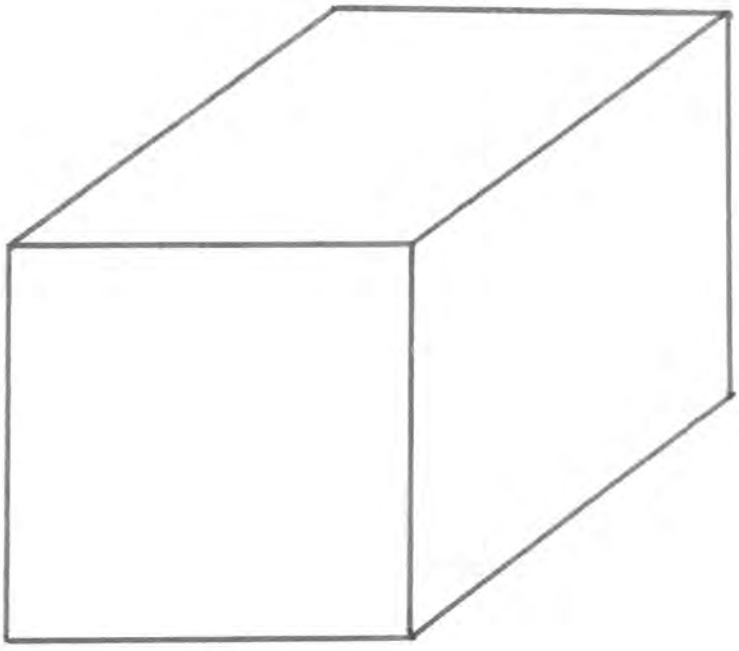
INTRODUCTION

The elimination of lines not normally viewed during a visual perception of a three-dimensional object that is being simulated on a computer graphics display is called hidden line elimination (Fig. 1). My project consists of a PL-1 computer program which implements an algorithm for hidden line elimination written by A. Montanari and R. Galemberti published in the April 1969 issue of the Communications of the ACM.

LIST OF FIGURES



(a)



(b)

Fig. 1 (a) Object without hidden line elimination.

(b) Object with hidden line elimination.

DEFINITIONS

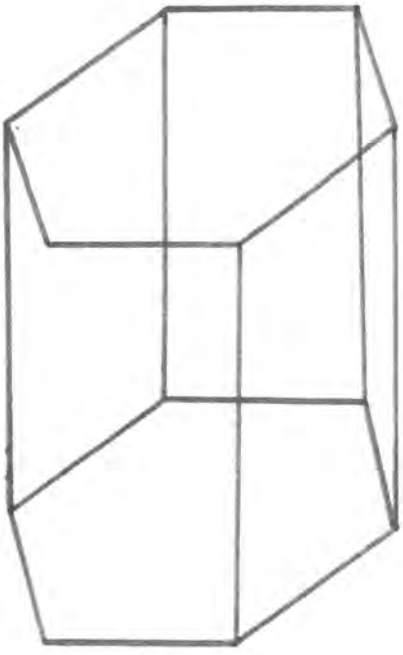
In order to discuss and have an understanding of three-dimensional objects and their projection on a computer graphics view screen some preliminary definitions are in order.

All objects in three-dimensional space are bounded by portions of a plane which are called faces. All faces are always considered closed. The volume of an object is the area enclosed by all the faces of an object.

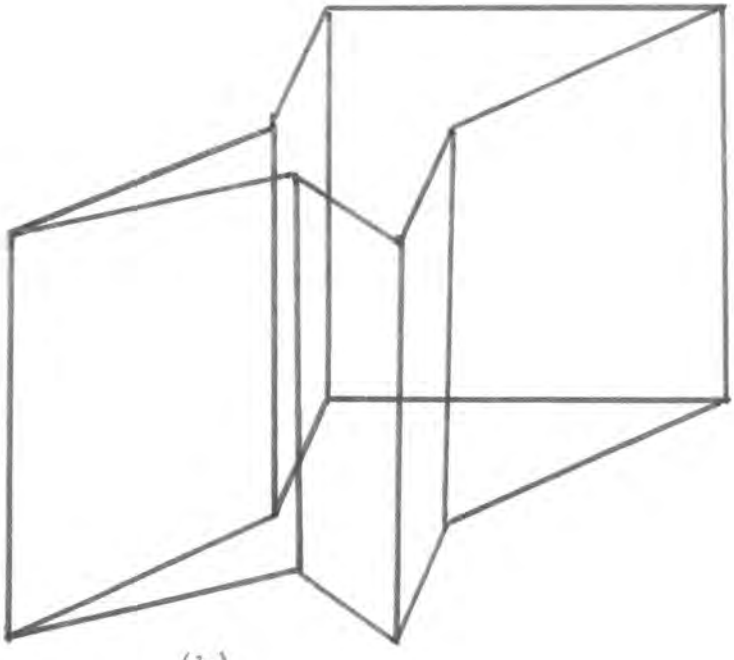
A three-dimensional object is either convex or concave. A convex object is an object that when a plane without bound is laid on any face of that object the plane does not intersect the volume of that object (Fig. 2a). A concave object is an object that when a plane without bound is laid on any face of that object that plane intersects the volume of that object (Fig. 2b). Objects which are both convex and concave will be considered concave in this paper.

Faces are considered on-view if they are not hidden by their own volume; otherwise, a face is considered off-view. Each face consists of edges and each edge has two end points called nodes. Every edge belongs to two faces.

The view-plane is an imaginary plane placed between the viewer and the object, and which also corresponds to the picture tube of the graphics terminal (Fig. 3).



(a)



(b)

Fig. 2 (a) An example of a convex object. (b) An example of a concave object. The necessity of hidden line elimination as an object becomes increasingly complex can readily be seen here.

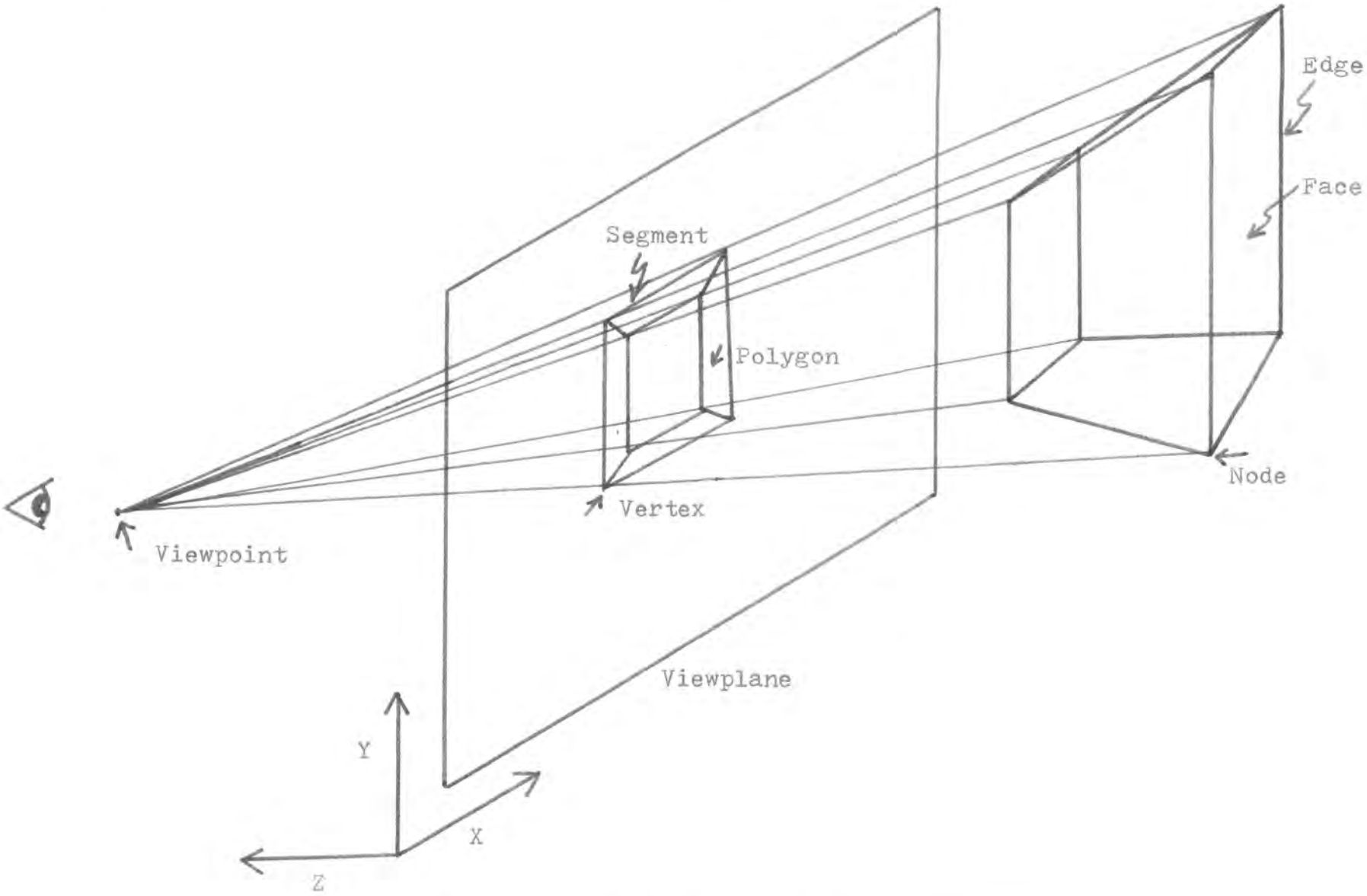


Fig. 3 Projection of an object to the viewplane.

The view-point of the observer is called Z_o. On the view-plane all faces are projected into polygons and all edges into segments, and nodes into vertices (Fig. 3).

A convex dihedral is formed by convex faces sharing a common edge. A concave dihedral is formed by two concave faces sharing a common edge. An edge is convex if it corresponds to a convex dihedral; otherwise, it is concave. A node is concave if it belongs to at least one concave edge; otherwise, it is convex.

DESCRIPTION OF THE ALGORITHM

The algorithm consists of two main steps. Step one is to determine if an edge belongs to a convex or concave dihedral. This step is necessary to determine if an object is convex or concave. In the second step all edges hidden by their own volume are eliminated. This is the actual process of eliminating hidden lines.

In order to determine whether an edge is concave or convex it is first necessary to determine the inward normal vectors of the two faces belonging to the edge in question. This is done for every face of the object.

Let j be the list of faces of an object, where j runs from 1 to the number of faces of the object, and let A_j be the area of any face j. Let n_j be the normal vector to a face j, and h_j be the number of nodes of face j. Let $P_{j,k}$ stand for the list of nodes of any face j, where $k = 1 \ldots h_j$, and where the nodes of every face are numbered in a clockwise direction around the face keeping the perimeter of the face on your right. Now using the formulas

$$2A_j n_j = \sum_{2}^{h_j - 1} (P_{j,k} - P_{j,1}) \times (P_{j,k+1} - P_{j,1}) \qquad (1)$$

$$n_j = 2A_j n_j / 2A_j n_j$$

we can compute the inward normal vector to any face j.

Now let $(P_s - P_r)$ be an edge vector we wish to investigate

for convexity or concavity, where $s > r$, and let j_1 and j_2 be the faces that share (P_s-P_r) as a common edge. J_1 is the face belonging to (P_s-P_r) and j_2 belongs to the edge (P_r-P_s) (Fig. 4a). Now let n_{j1} and n_{j2} be the inward normal vectors to j_1 and j_2 respectively. Now by taking the edge vector (P_s-P_r) and the vector cross product of n_{j1} and n_{j2} and find the vector dot product of these two quantities we have

$$c = (P_s - P_r) \cdot (n_{j1} \times n_{j2})$$

and if $c > 0$ the edge is concave; otherwise, it is convex. This can readily be seen by noting that the normal vectors of each face belonging to the edge (P_s-P_r) are each of unit length and therefore their vector cross product (noting that n_{j1} is always crossed into n_{j2}) is either a positive or negative unit vector lying on the same axis as (P_s-P_r) (Fig. 4b). Now applying the previous formula it can be seen that c can only be positive or negative and denotes whether an edge corresponds to a convex or concave dihedral. It is necessary to perform this procedure once for each object.

The elimination of edges hidden by their own volume is the second and final step of the algorithm. Let j now be a polygon on the picture plane corresponding to a face j in three-dimensional space. Let nodes $P_{j,k}$ be projected into vertices $V_{j,k}$ to the view plane (remembering that nodes become vertices on the view plane). Now by replacing

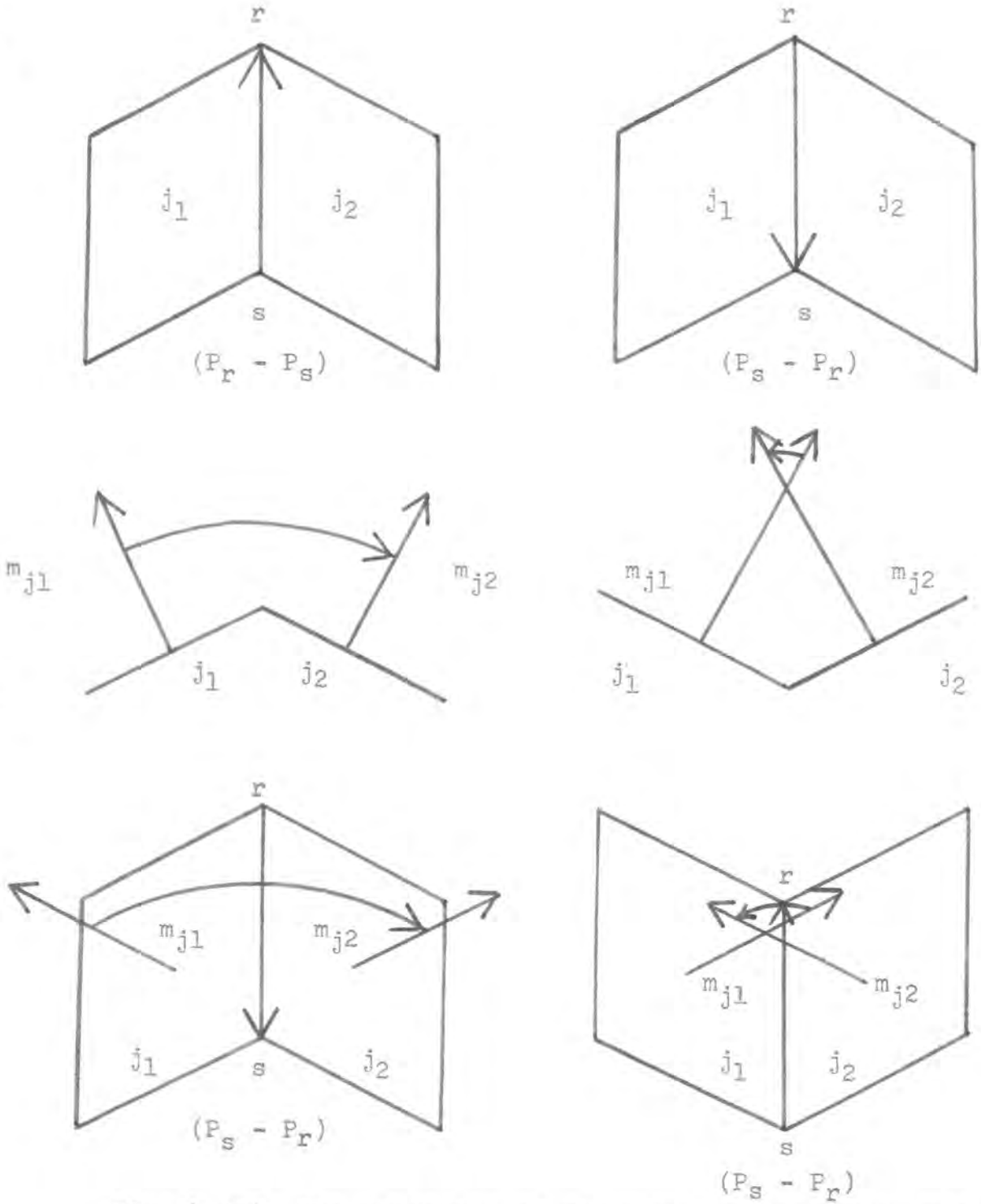


Fig. 4 (a) The direction of the edge vectors of j_1 and j_2 respectively (b) Pictorial representation of determination of convex and concave edges. Remembering that all normal vectors are inward and j_1 is always crossed into j_2.

$P_{j,k}$ with its corresponding $V_{j,k}$ in equation one we have

$$2A_j m_j = \sum_{2}^{h_j-1} (V_{j,k} - V_{j,1}) \times (V_{j,k+1} - V_{j,1})$$

$$m_j = 2A_j m_j / 2A_j m_j$$

which results in a normal unit vector m; which is perpendicular to the picture plane, thus m_j has only one component and this component is on the z axis only. Now if $Z_o \cdot m_j < 0$ (recalling that Z_o is the view point) the face j is on view; otherwise, it is off view (Fig. 5). Now, any segment belonging to two off view faces is erased. As we are only dealing with convex objects the object can now be displayed and it will contain no hidden lines. The previous procedure must be performed for each different projection of the object.

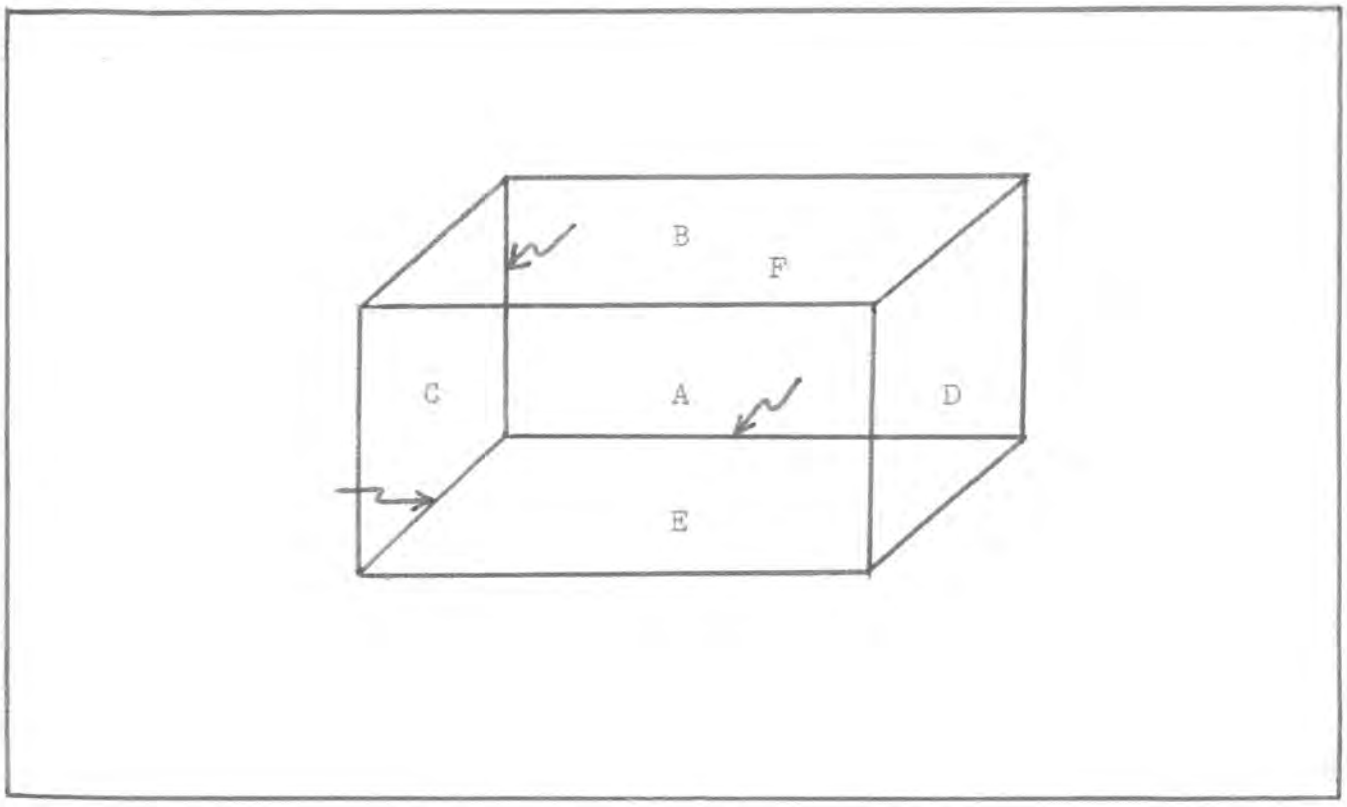


Fig. 5 Z_o is considered in front of the page. Faces C, F, and E are off view because their corresponding polygons have normal vectors which also point out of the page. When these normals are dotted with Z_o the quantity is positive; therefore, these faces are off view. Any segment that shares two off view faces is erased, these segments are indicated by the arrows.

IMPLEMENTATION OF THE ALGORITHM

Throughout the program all data, other than working storage items, are arranged in the form of structures with each structure forming an element of a linked list. A brief description of this type of data structure will be given for those who may not be familiar with it. First a linked list has what is termed a head and a tail, these correspond to a beginning and an end of a list. The head of a list contains what is called a pointer, which is nothing more than the address of the next data element in the list. Every item in the list contains one of these pointers which points to the next item in the list (Fig. 6). In this way it is a simple matter to add or delete any element of the linked list by altering the appropriate pointer.

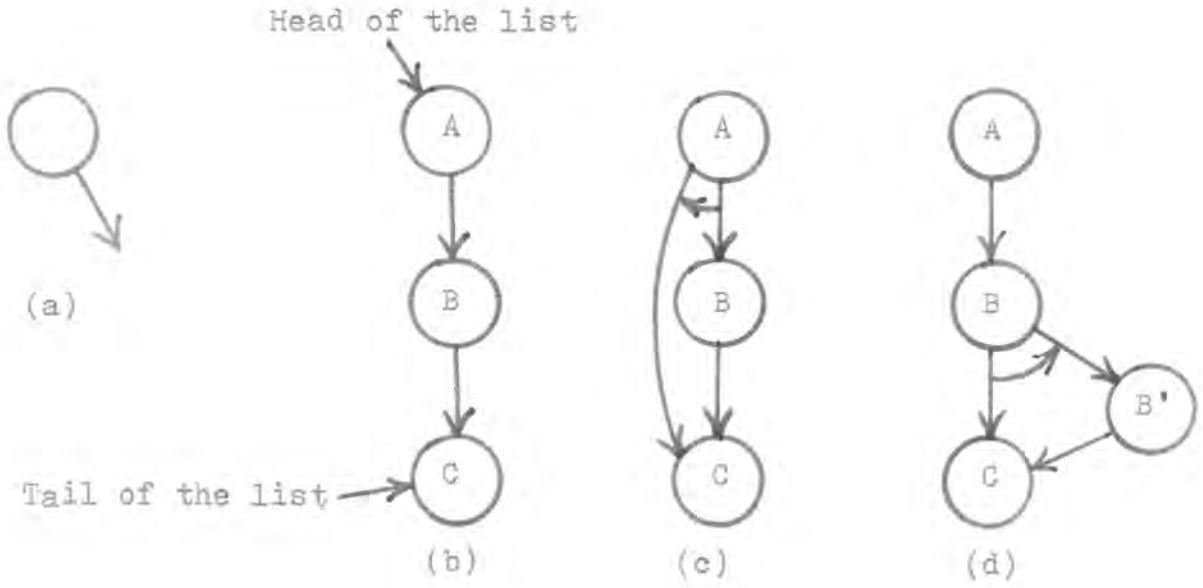


Fig. 6 (a) The circle stands for a single data item in a linked list which could be for example a structure, a single data item, or an entire array. (b) A typical linked list data arrangement. (c) To delete an item from a linked list, for example item B in our list, we merely change the pointer of item A so that it contains the address of item C instead of that of item B. Now as we proceed from the head of the list to the tail of the list item C will be the next item. (d) To add an item to the list, let us say B', we set the pointer of B' to point at C (contain the address of C) and set the pointer of B to contain the address of B'.

DATA ELEMENTS OF THE PROGRAM

The hidden line program is designed as an external procedure so that hidden line elimination may be implemented as needed by the user. As it is designed to supplement other computer graphics programs it is necessary to know the data items required by it and what type of arrangement they should have. It is a requirement of PL-1 that all data elements which are used by both a main procedure and an external procedure be identically declared and structured therefore care must be taken when duplicating the data section of this procedure. Examples of creating the linked lists necessary for the program can be seen in the graphics program written by Mr. Neil Webre that is used as the calling procedure for the hidden line procedure described here, it is included in the computer programs listed in the back of this report.

Also it must be noted that it is the responsibility of the user to transpose the nodes of the object to the view-plane, create the face, vertex and vertex pointer lists. Also Z_o must be supplied by the user. The POINT list and EDGE list is created by the hidden line program. The edge list is only created once and the POINT list is recreated for each new projection of the object to the view-plane.

A detailed listing of all data elements used in the

program is now given.

Each node is listed in a structure labeled VERTEX under the name of POINT. POINT is broken down further to contain the nodes three-dimensional coordinates. This breakdown is XN, YN and ZN. A nodes transpose to the view plane is also contained in the structure under the label of TRAN. TRAN is broken down further to XP, YP and ZP which are the coordinates of the transpose. NEXT contains the pointer which is the address of the next node structure in the linked list. CON provides storage for an indicator which if it contains a 1 the node is concave and if it zero the node is convex. The internal name of the node, and integer, is stored under VERTEX_ID.

Edges are also stored in a linked list data arrangement labeled EDGES. END(2) is a one-dimensional two element pointer which contains the addresses of the two nodes which form the edge. OFF_VIEW_FLAG is used as a marker also, if it contains a 1 the edge is off view, if it contains a zero the edge is on view. EDGE_VECTOR is the vector difference of the two nodes, which are also vectors, and the resultant components are stored in EDGE_VECTOR's breakdown XE, YE, and ZE, NEXT_EDGE is a pointer that contains the address of the next element in the edge list. FACE_LIST(2) is a one-dimensional two element array with FACE_LIST(1) containing the address of the face name corresponding to j_1 of the edge and FACE_LIST(2) containing

the face name corresponding to j_2 of the edge. CONCAVITY_TEST is used also as a marker, if it contains a 1 the edge is concave, if it contains a zero it is convex.

Face data is also in the form of a linked list with each face having its necessary data arranged in structure form. The structure name is FACE. FACE_NAME contains an integer which represents the name of a face. NORMAL_VECTOR contains the vector components of the inward normal vector of the face, these are stored in EYE, JAY and KAY. ON_OFF_VIEW I is an indicator quantity which contains a zero if the face is on view and a 1 if it is off view. NEXT_FACE is a pointer that contains the address of the next face in the linked list. NUM_VERX contains the number of nodes the face consists of. VERTEX_LABEL_LIST is a one-dimensional sixteen element array which contains the name (VERTEX_ID) of each node belonging to the face.

POINT is the last of the linked list data items of the program. POINT is the list from which the object is displayed. The head and tail (HEADF and TAILF) of POINT is declared external, this is done to allow manipulation of POINT without passing parameters between procedures. Contained in POINT is POINTS which is broken down into XI,YI,XF and YF which are the coordinates of the line to be displayed. NEXT_POINT is a pointer containing the address of the next point in the POINT list.

The remaining quantities are used as working storage

items which provide the necessary temporary storage locations for various insundry calculations and operations.

THE PROGRAM

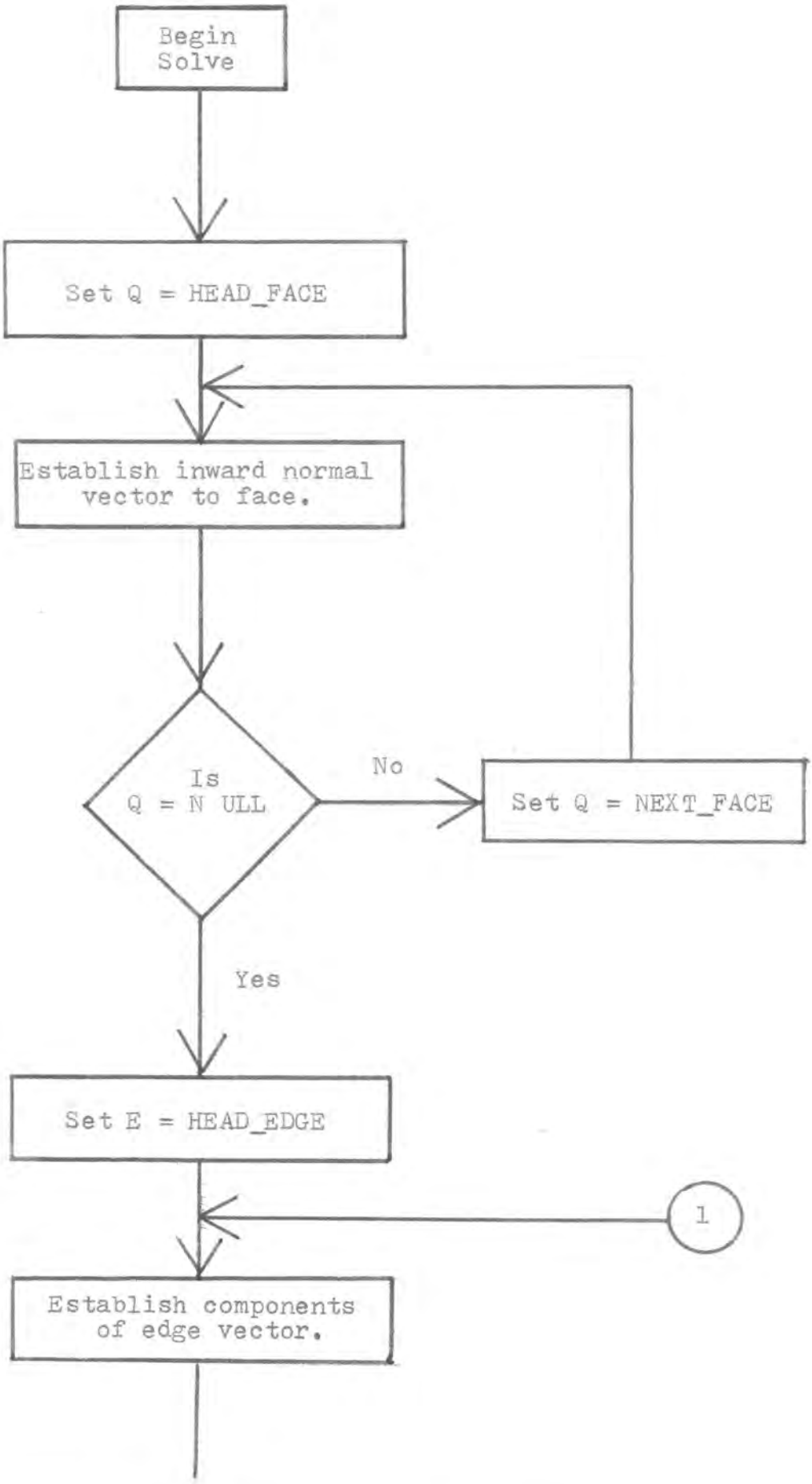
The first step of the program is identical to that of the algorithm, that is, the determination of convex and concave edges. This step can be completely eliminated if the object is known to be completely convex. However as this step is only executed once for any object the computer time that would be lost by its inclusion would be very small. The logic necessary to implement this section of the algorithm is contained from statement number 17 thru statement number 67.

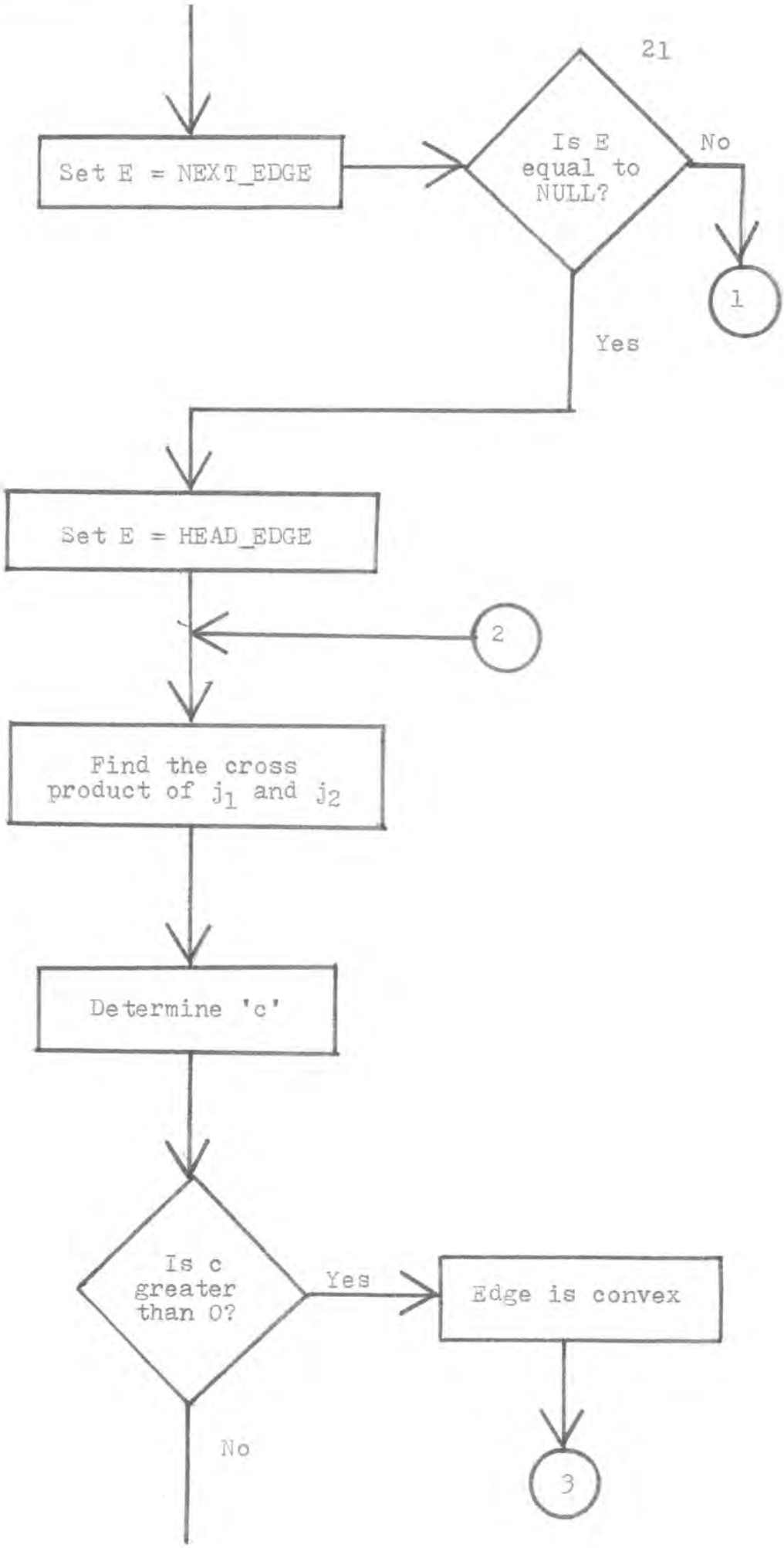
The second step of the algorithm is implemented at the entry point HIDE and continues to the end of the program. This section handles the actual elimination of hidden lines. After the data has been correctly set up it is only necessary to call HIDE once for each different projection of the same object. HIDE will then return a POINT list which will be a list of all the line segments to be displayed.

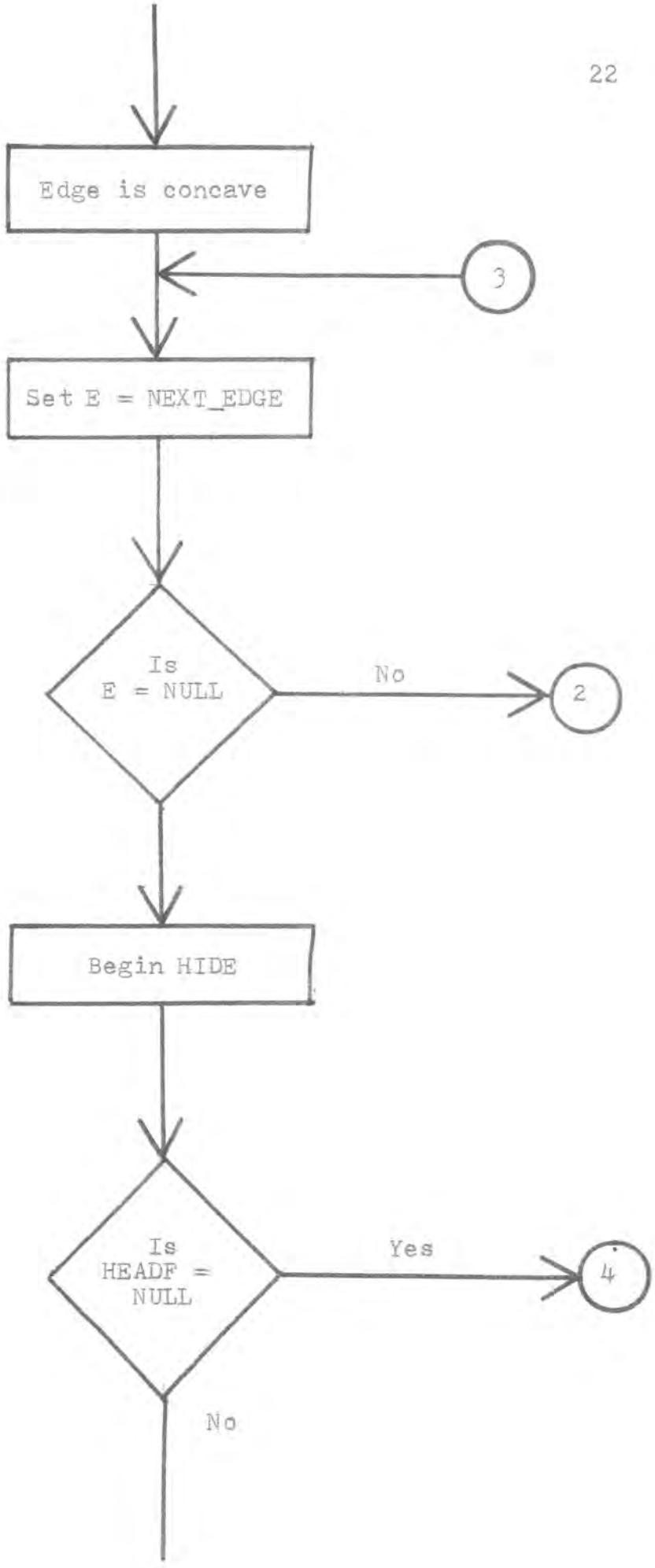
A general account of what each section of the program performs will now be given. A more detailed account is given through the use of a flow chart of the program (Fig. 7). If even a more detailed account is necessary it is easy to follow the logic of each section of the program as it has been written in a straightforward manner. This is especially easy to follow once the purpose of each section is known. From statement number 17 to statement

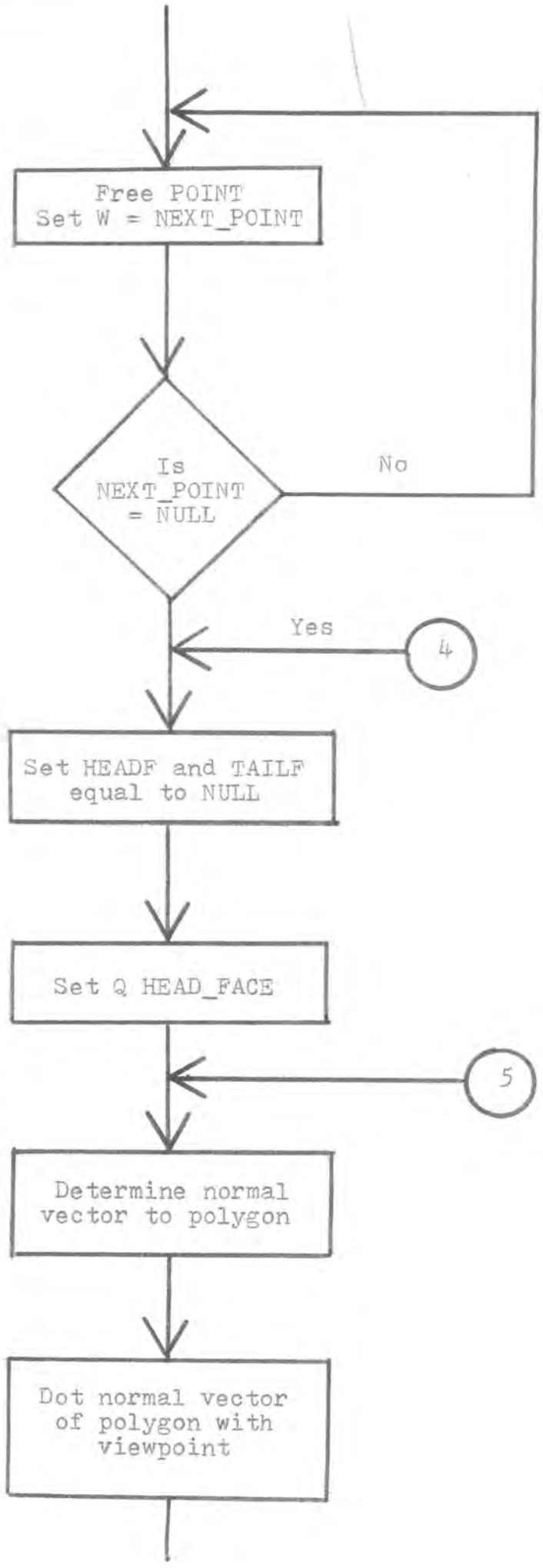
number 42 we calculate the components of the normal vector of each face of the object. From statement number 43 thru 52 we establish the components of the edge vectors. Whether an edge is convex or concave is determined from statements 53 thru 67.

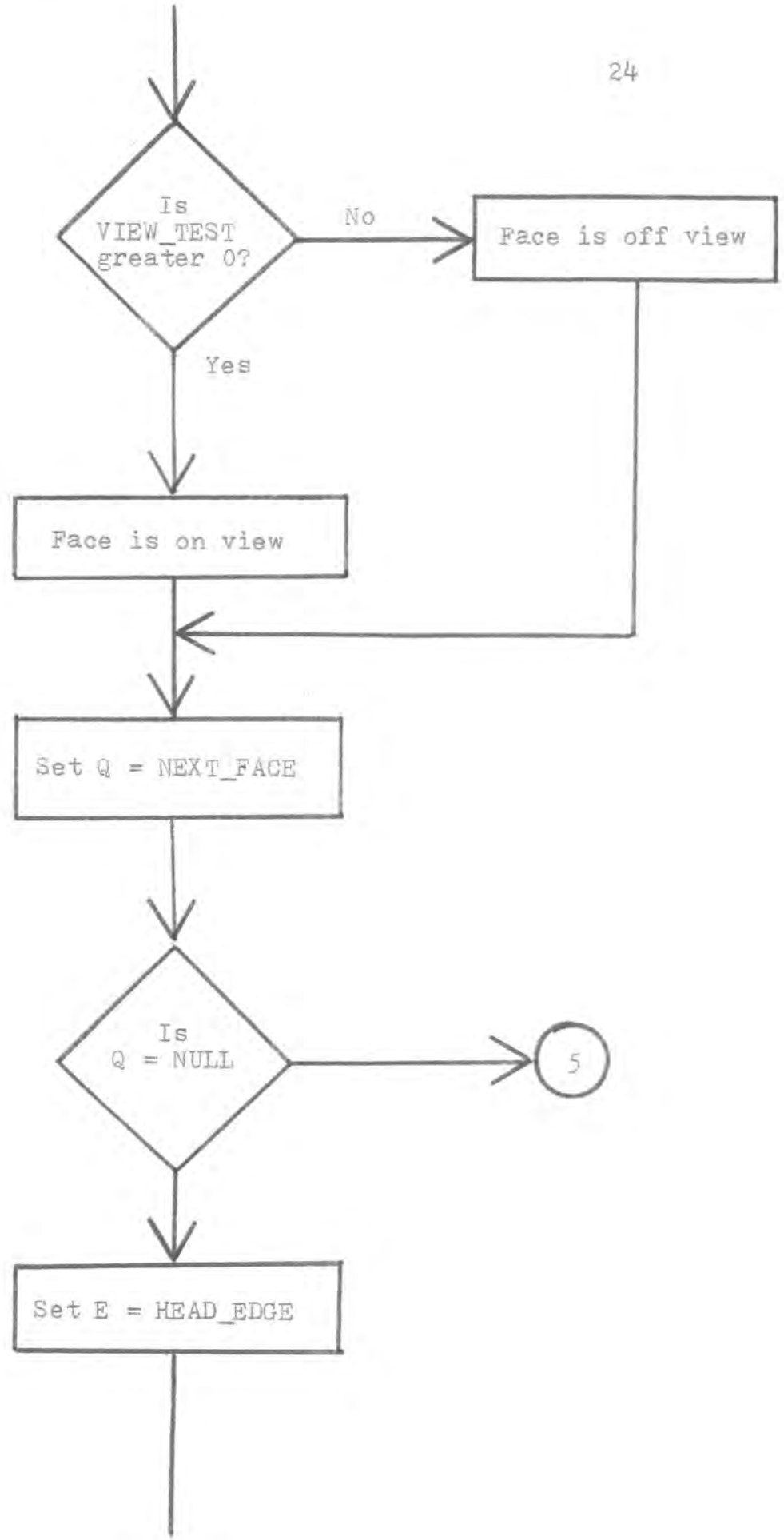
HIDE, where the actual elimination takes place begins at statement number 68. On view and off view faces are determined from statement numbers 68 thru 98. Statement numbers 99 thru 114 establishes whether an edge is on view or off view. Statement numbers 115 thru 135 create the list of edges which are to be displayed.


Begin Solve
Set Q = HEAD_FACE
Establish inward normal vector to face.
Is Q = N ULL
No
Set Q = NEXT_FACE
Yes
Set E = HEAD_EDGE
1
Establish components of edge vector.


Set E = NEXT_EDGE
Is E equal to NULL?
No
1
Yes
Set E = HEAD_EDGE
2
Find the cross product of j_1 and j_2
Determine 'c'
Is c greater than 0?
Yes
Edge is convex
No
3


Edge is concave
3
Set E = NEXT_EDGE
Is
E = NULL
No
2
Begin HIDE
Is
HEADF =
NULL
Yes
4
No


Free POINT
Set W = NEXT_POINT
Is
NEXT_POINT
= NULL
No
Yes
4
Set HEADF and TAILF
equal to NULL
Set Q HEAD_FACE
5
Determine normal
vector to polygon
Dot normal vector
of polygon with
viewpoint


Is
VIEW_TEST
greater 0?
No
Face is off view
Yes
Face is on view
Set Q = NEXT_FACE
Is
Q = NULL
5
Set E = HEAD_EDGE

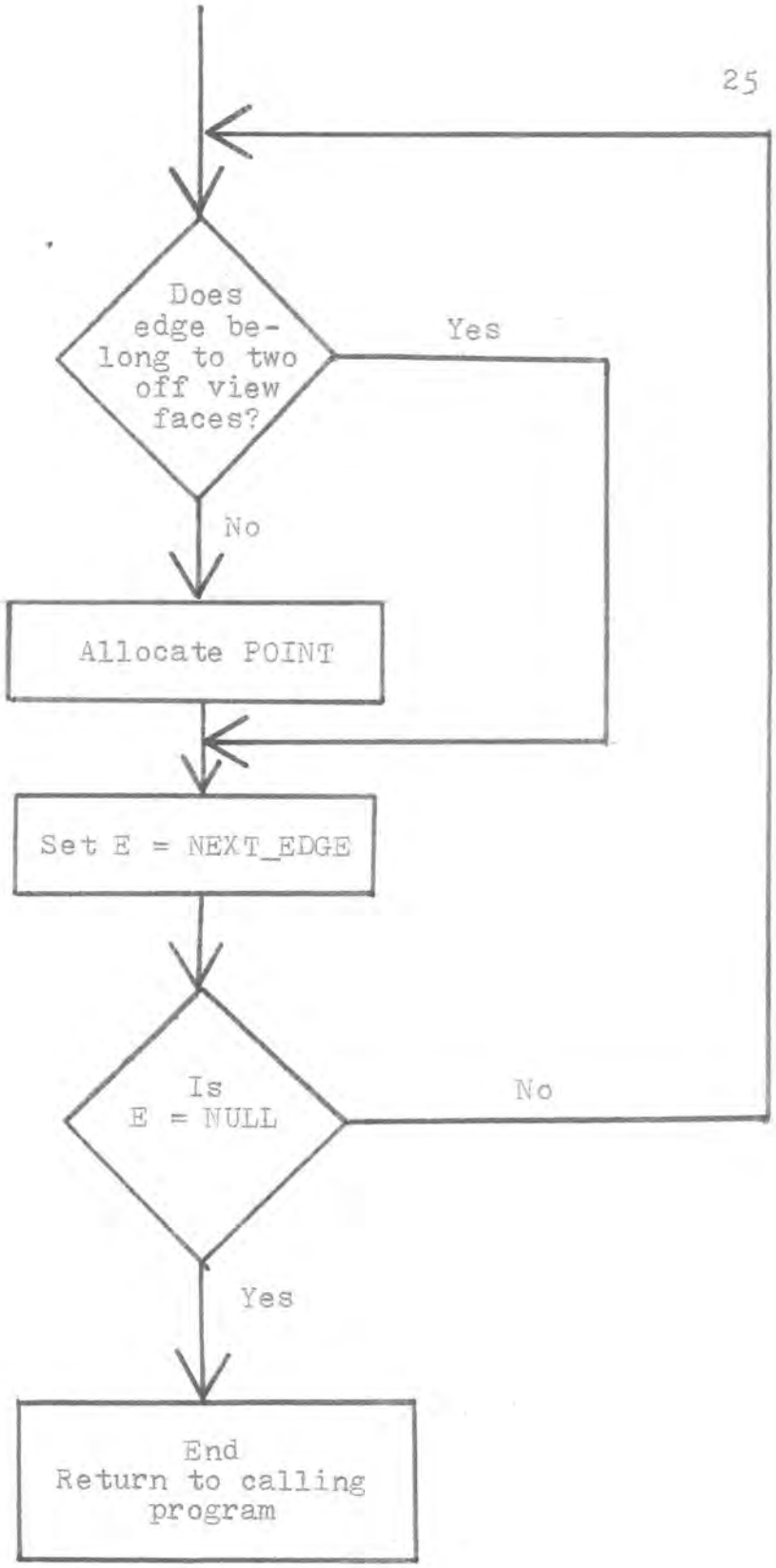


Fig. 7 Flow chart of program.

APPENDIX

Hidden Line Program

F EFFECT */

*/

D SPACE */

```
        SOLVE: PROCEDURE(HEAD_EDGE,HEAD_FACE);


STMT LEVEL NEST
  1                 SOLVE: PROCEDURE(HEAD_EDGE,HEAD_FACE);
  2     1           DCL (HEADF,TAILF) EXTERNAL POINTER;
                    /* THE BEGINNINGS OF EDGE AND FACE LISTS */
  3     1           DCL 1 POINT BASED(F),
                    /* LIST OF POINTS OF THE OBJECT WITH HIDDEN LINE ELIMINATIO
                        2 POINTS,
                           ( 3 XI,3 YI, 3 XF, 3 YF) FLOAT BIN,
                    /* THE INITIAL AND FINAL POINTS OF LINE SEGMENTS TO BE PRIN
                        2 NEXT_POINT POINTER;
                    /* ADDRESS OF NEXT POINT IN POINTS LIST */
  4     1           DCL VPL(100) POINTER EXTERNAL;
                    /* ARRAY OF POINTERS POINTING TO NODES IN 3-D SPACE */
  5     1           DCL 1 EDGE BASED(E),
                    /* EDGE LIST STRUCTURE */
                        2 END(2) POINTER,
                    /* POINTER THAT POINTS TO THE VERTICES THAT MAKE UP EDGE IN
                        2 OFF_VIEW_FLAG FIXED BIN(15),  /* 1 IF OFF,0 IF ON*/
                    /* FLAG WHICH SIGNIFIES IF EDGE IS ON VIEW OF OFF VIEW */
                      2 EDGE_VECTOR,
                           ( 3 XE, 3 YE, 3 ZE) FLOAT BIN,
                    /* VECTOR  COMPONENTS OF EDGE */
                        2 NEXT_EDGE POINTER,
                    /* ADDRESS OF NEXT EDGE IN EDGE LIST */
                        2 FACE_LIST(2) POINTER,
                    /* NAMES OF FACES BELONGING TO EDGE */
                           2 CONCAVITY_TEST FIXED BIN(15); /* 1 IF CONC,0IF CON
                    /* FLAG THAT SIGNALS WHETHER AN EDGE IS CONCAVE OR CONVEX *
  6     1           DCL 1 VERTEX BASED(P),
                    /* VERTEX LIST OF THE OBJECT IN 3-D SPACE */
                        2 POINT,
                           ( 3 XN, 3 YN, 3 ZN) FLOAT BIN,
                    /* VERTICES IN  3-D SPACE */
                        2 TRAN,
                           ( 3 XP, 3 YP, 3 ZP) FLOAT BIN,
                    /* TRANSPOSITION TO THE VIEW PLANE */
                        2 NEXT POINTER,
                    /* ADDRESS OF NEXT VERTEX IN LIST */
                        2 CON FIXED BIN(15),
                    /* FLAG THAT SHOWS IF AN OBJECT IS CONCAVE OR CONVEX */
                    /* 1 IF CONCAVE, 0 IF CONVEX*/
                        2 VERTEX_ID,
                    /* NAME OF THE VERTEX */
                             3 NUMBER FIXED BIN(15);
  7     1           DCL 1 FACE BASED(Q),
                    /* FACE LIST STRUCTURE */
                        2 FACE_NAME FIXED BIN(15),
                    /* NAME OF THE FACE */
                        2 NORMAL_VECTOR,
                    /* NORMAL VECTOR TO FACE */
                           (3 EYE, 3 JAY, 3 KAY) FLOAT BIN,
                    /* COMPONENTS OF NORMAL VECTOR */
                        2 ON_OFF_VIEW FIXED BIN(15),
                    /* ON VIEW IF 0, OFF VIEW IF 1*/
                    /* FLAG WHICH SHOWS IF FACE IS ON VIEW OR OFF-VIEW */
                        2 NEXT_FACE POINTER,
                    /* ADDRESS OF NEXT FACE IN LIST */
```

ACE LIST */

ES*/

```
      SOLVE: PROCEDURE(HEAD_EDGE,HEAD_FACE);

STMT LEVEL NEST
                          2 NUM_VERT FIXED BIN(15),
                        /* CONTAINS THE NUMBER OF VERTICES THE FACE CONTAINS */
                           2 VERTEX_LABEL_LIST(16) FIXED BIN(15);
                        /* CONTAINS THE NAMES OF THE VERTICES */
   8     1              DCL (KV,VIEW_TEST) FLOAT BIN;
   9     1              DCL (WAY,OTHER_POINT,WI,PAY,W,V,VI) POINTER;
  10     1              DCL C FLOAT BIN;
  11     1              DCL (AA1,AA2,BB1,BB2)  FLOAT BIN;
                        /* HEAD_EDGE IS BEGINNING OF EDGE LIST, HEAD_FACE IS HEAD
                        DCL (HEAD_EDGE,HEAD_FACE)POINTER;
                        DCL (XT,YT,ZT) FLOAT BIN;
                        DCL (A1,A2,A3,B1,B2,B3) FLOAT BIN;
                        DCL (I,J,K) FLOAT BIN;
                        DCL TEMP FLOAT BIN;


                        /* DETERMINATION OF CONCAVE AND CONVEX EDGES */


  12     1              Q = HEAD_FACE;
  13     1              REBOUND: P =                    VPL(VERTEX_LABEL_LIST(1));
  14     1              I = 0; J = 0; K = 0;
  17     1              DO II = 2 TO NUM_VERT -1;
  18     1     1        W =                  VPL(VERTEX_LABEL_LIST(II));
                        /* FINDING THE VECTOR DIFFERENCE BETWEEN NODES OF A FACE *
  19     1     1        A1 = (W -> XN) - XN;
  20     1     1        A2 = (W -> YN) - YN;
  21     1     1        A3 = (W -> ZN) - ZN;
  22     1     1        V =                   VPL(VERTEX_LABEL_LIST(II+1));
  23     1     1        B1 = (V -> XN) - XN;
  24     1     1        B2 = (V -> YN) - YN;
  25     1     1        B3 = (V -> ZN) - ZN;
                        /* FINDING THE CROSS PRODUCT OF NODE VECTORS */
  26     1     1        I = I + (A2*B3-B2*A3);
  27     1     1        J = J + (+B1*A3 - A1*B3);
  28     1     1        K = K + (A1*B2-B1*A2);
  29     1     1        END;
                        /* ESTABLISHING THE INWARD NORMAL VECTOR OF A FACE */
  30     1              TEMP = SQRT(I*I+J*J+K*K);
  31     1              EYE = I/TEMP;
  32     1              JAY = J/TEMP;
  33     1              KAY = K/TEMP;
  34     1              Q = NEXT_FACE;
  35     1              IF Q = NULL THEN GO TO OUT;
  37     1              GO TO REBOUND;


                        /* WE NOW HAVE ALL OF THE INWARD NORMAL VECTORS OF ALL THE


                        /* ESTABLISHING WHAT NODES MAKE UP WHAT EDGES */

  38     1              OUT: E = HEAD_EDGE;
```

THE TWO FACES

```
        SOLVE: PROCEDURE(HEAD_EDGE,HEAD_FACE);

STMT LEVEL NEST

 39     1          BACK: P = END(2);
 40     1          OTHER_POINT = END(1);
 41     1          XE = (XN - OTHER_POINT -> XN);
 42     1          YE = (YN - OTHER_POINT -> YN);
 43     1          ZE = (ZN - OTHER_POINT -> ZN);
 44     1          E = NEXT_EDGE;
 45     1          IF E = NULL THEN GO TO POLYGONS;
 47     1          GO TO BACK;


                   /* DETERMINATION OF CONVEX AND CONCAVE EDGES */


 48     1          POLYGONS: E = HEAD_EDGE;
 49     1          FRONT: Q = FACE_LIST(1);
 50     1          PAY = FACE_LIST(2);
 51     1          CONCAVITY_TEST = 0;
                   /* FINDING THE CROSS PRODUCT OF THE INWARD NORMAL VECTORS
                   BELONGING TO THE EDGE IN QUESTION */
 52     1          XT = ((JAY*(PAY -> KAY))-(KAY*(PAY -> JAY)));
 53     1          YT = ((-EYE*(PAY -> KAY)) + (KAY*(PAY -> EYE)));
 54     1          ZT = ((EYE*(PAY -> JAY)) - (JAY*(PAY -> EYE)));
 55     1           C = (XT*XE) + (YT*YE) + (ZT*ZE);
 56     1          IF C > 0. THEN CONCAVITY_TEST = 1;
                   /* IF C IS > 0 THE EDGE IS CONVEX ELSE IT IS CONCAVE */
 58     1          HERE: E = NEXT_EDGE;
 59     1          IF E = NULL THEN GO TO ON_VIEW_PROCESS;
 61     1          GO TO FRONT;
 62     1          ON_VIEW_PROCESS: GO TO FINIS;


                   /* FINDING ON VIEW AND OFF VIEW FACES */

 63     1          HIDE: ENTRY(HEAD_EDGE,HEAD_FACE);
 64     1          IF HEADF = NULL THEN GO TO QUIT;
 66     1          F = HEADF;
 67     1          L: W = NEXT_POINT;
 68     1          FREE POINT;
 69     1          IF  W = NULL THEN GO TO QUIT;
 71     1          F= W;
 72     1          GO TO L;
 73     1          QUIT: HEADF,TAILF = NULL;
 74     1          Q = HEAD_FACE;
 75     1          UP: P =                    VPL(VERTEX_LABEL_LIST(1));
 76     1          KV = 0.0;
 77     1          ON_OFF_VIEW = 1;
```

= 1;

1;

E COORDINATES OF

Calling Program

Y,VZ,

```
        MAIN:PROCEDURE OPTIONS (MAIN);


STMT LEVEL NEST

  1                 MAIN:PROCEDURE OPTIONS (MAIN);

                     /* DECLARE THE VARIABLE ATTRIBUTES */

  2     1           DCL (IUNIT,NIL)BIN FIXED(31);
  3     1           DCL (RR,SS) FIXED BIN(15);
  4     1           DCL (TT,PP) POINTER;
  5     1                 DCL (IGSP,IGRAFD,IGDS1) FIXED BIN (31,0);

  6     1           DCL STPOS ENTRY (,FLOAT BIN,FLOAT BIN);

  7     1           DCL SDATL ENTRY (,FLOAT BIN,FLOAT BIN,FLOAT BIN,FLOAT BIN);
  8     1           DCL RQATN ENTRY(,,FIXED BIN(31),,FIXED BIN(31),FIXED BIN(31
  9     1           DCL MLITS ENTRY(,FIXED BIN(31));
 10     1           DCL UU POINTER;
 11     1           DCL ENATN ENTRY(,FIXED BIN(31),FIXED BIN(31));
 12     1           DCL PSGMT ENTRY(FIXED BIN(31),FLOAT BIN,FLOAT BIN,
                    FLOAT BIN,FLOAT BIN);

 13     1           DCL(RMAG,COS_PHI,SIN_PHI,COS_THETA,SIN_THETA,URX,URY,URZ,VX
                        UVX,UVY,UVZ,VMAG,QMAG,DMAG,TX,TY,TZ )FLOAT BIN;

 14     1           DCL (R,D,PHI,THETA  )FLOAT BIN;
 15     1           DCL ARAY(10) FIXED BIN(31,0);
 16     1           DCL TMAG FLOAT BIN;
 17     1           DCL WASTE FIXED BIN(15);

 18     1           DCL  (DR,DD,DTHETA,DPHI)FLOAT BIN;
 19     1           DCL (TPA,TPB) FIXED BIN (31);

 20     1                DCL (X,Y,Z)FLOAT BIN,(NEDGES,TAG)FIXED BIN(15);

 21     1                DCL (XT,YT)FLOAT BIN;
 22     1           DCL (ALEVEL,KEY) FIXED BIN (31,0), CYCLE BIT(1);
 23     1           DCL IGDS2 FIXED BIN (31,0);
 24     1           DCL FLIP BIT(1) INITIAL ('0'B);
 25     1           DCL NULL BUILTIN;
 26     1           DCL (HEADF,TAILF) EXTERNAL POINTER;
 27     1           DCL (HEADQ,TAILQ) POINTER;

 28     1           DCL ID CHAR(30),(HEAD,TAIL)POINTER , W POINTER,
                        DELAY FIXED BIN;
 29     1           DCL (LABEL,NVERT) BIN FIXED(15);
 30     1           DCL VPL(100) POINTER EXTERNAL;
 31     1           DCL 1 POINT BASED(F),
                        2 POINTS,
                           ( 3 XI,3 YI, 3 XF, 3 YF) FLOAT BIN,
                        2 NEXT_POINT POINTER;
 32     1           DCL 1 EDGE BASED(E),
                        2 END(2) POINTER,
                        2 OFF_VIEW_FLAG FIXED BIN(15),
                      2 EDGE_VECTOR,
                           ( 3 XE, 3 YE, 3 ZE) FLOAT BIN,
                        2 NEXT_EDGE POINTER,
                        2 FACE_LIST(2) POINTER,
```

```
        MAIN:PROCEDURE OPTIONS (MAIN);

STMT LEVEL NEST

                        2 CONCAVITY_TEST(2) FIXED BIN(15);
 33     1           DCL 1 VERTEX BASED(P),
                        2 POINT,
                           ( 3 XN, 3 YN, 3 ZN) FLOAT BIN,
                        2 TRAN,
                           ( 3 XP, 3 YP, 3 ZP) FLOAT BIN,
                        2 NEXT POINTER,
                        2 CON FIXED BIN(15),
                        2 VERTEX_ID,
                           3 NUMBER FIXED BIN(15);
 34     1           DCL 1 FACE BASED(Q),
                        2 FACE_NAME FIXED BIN(15),
                        2 NORMAL_VECTOR,
                           (3 EYE, 3 JAY, 3 KAY) FLOAT BIN,
                        2 ON_OFF_VIEW FIXED BIN(15),
                        2 NEXT_FACE POINTER,
                        2 NUM_VERT FIXED BIN(15),
                         2 VERTEX_LABEL_LIST(16) FIXED BIN(15);
 35     1           DCL (HEADE,TAILE) POINTER;
 36     1           DCL C FLOAT BIN;
 37     1               DCL (SOLVE,HIDE) ENTRY (POINTER,POINTER) EXTERNAL;
 38     1           DCL (III,JJJ) FIXED BIN(15);
 39     1           HEADQ,TAILQ = NULL;
 40     1           HEADF,TAILF = NULL;
 41     1           HEADE,TAILE = NULL;
 42     1           HEAD,TAIL = NULL;
 43     1           VPL = NULL;


                      /*  INITIALIZE THE GPS ROUTINES  */

 44     1               IUNIT=40;

 45     1               NIL=-5;

 46     1               CALL INGSP(IGSP,NIL);
 47     1               CALL INDEV(IGSP,IUNIT,IGRAFD);
 48     1              CALL INGDS(IGRAFD,IGDS1);
 49     1                CALL INGDS(IGRAFD,IGDS2);
 50     1              CALL SDATL(IGDS1,-1.0,-1.0,1.0,1.0);
 51     1                CALL SDATL(IGDS2,-1.0,-1.0,1.0,1.0);
 52     1           CALL SALRM(IGRAFD);
 53     1                CALL CRATL(IGRAFD,ALEVEL);
 54     1                 CALL ENATN(ALEVEL,0,-14);
 55     1           CALL MLITS(ALEVEL,3);
 56     1           TPA = 1; TPB = 40;
 58     1           CALL SPEC(IGSP,TPA,TPB);

                     /*   READ THE ID PORTION OF A DATA CARD  */
 59     1           RID: CALL RQATN(ALEVEL,KEY,1,ARAY,0,-14);
 60     1                IF KEY ¬= 0 THEN GO TO PROCESS;
 62     1           RID1: CYCLE = '0'B;
 63     1                 GET LIST(ID);
```

ND GO TO

T CARD')

;

K;END;

```
        MAIN:PROCEDURE OPTIONS (MAIN);

STMT LEVEL NEST
                        /*  BRANCH ACCORDING TO THE INSTRUCTION ON THE CARD  */


  64     1                  I=INDEX(ID,'VIEW DATA');

  65     1                  IF I¬=0 THEN GO TO VIEWD;

  67     1                  I=INDEX(ID,'VERTEX');

  68     1                  IF I¬=0 THEN GO TO VERT;
  70     1               I = INDEX(ID,'FACE');
  71     1               IF I ¬= 0 THEN GO TO FAC;

  73     1                  I=INDEX(ID,'PLOT');

  74     1                  IF I¬=0 THEN GO TO PLT;

  76     1                  I=INDEX(ID,'BEGIN STRUCTURE DATA');

  77     1                  IF I¬=0 THEN GO TO BS;

  79     1                  I=INDEX(ID,'END STRUCTURE DATA');
  80     1               IF I ¬= 0 THEN GO TO CSOLVE;

  82     1                  I=INDEX(ID,'STOP');

  83     1                  IF I¬=0 THEN GO TO STP;

                        /*  COULD NOT FIND A LEGIT INSTRUCTION ON THIS CARD. PRIN
                              THE NEXT CARD */

  85     1                  PUT EDIT(ID,'NOT A LEGAL INSTRUCTION, SKIP AND READ
                             (SKIP(2),COLUMN(15),A,A);

  86     1                  GO TO RID;
                           /* CREATE A FACE NODE; LINK INTO FACE LIST  */
  87     1               FAC: GET DATA(LABEL,NVERT);
  88     1               ALLOCATE FACE;
  89     1               FACE_NAME = LABEL; NUM_VERT = NVERT;
  91     1               GET LIST((VERTEX_LABEL_LIST(L)DO L = 1 TO NVERT));
  92     1               IF HEADQ = NULL THEN HEADQ,TAILQ = Q;
  94     1               TAILQ -> NEXT_FACE = Q;
  95     1               NEXT_FACE = NULL;
  96     1               TAILQ = Q;
  97     1               GO TO RID;
                        /* PROCESS ATTENTION KEY */
  98     1               PROCESS: IF KEY = 1 THEN CALL RQATN(ALEVEL,KEY,2,ARAY,0,-
 100     1                     IF KEY = -1 THEN GO TO STP;
 102     1               IF KEY=2 THEN DO;DR,DD,DTHETA,DPHI = 0.0;CYCLE='1'B;GOTO
 108     1                  CYCLE = '1'B;
 109     1                  IF KEY = 3 THEN DO;DTHETA= 2.0;GO TO FPROC;END;
 114     1                  IF KEY = 4 THEN DO;DTHETA=-2.0;GO TO FPROC;END;
 119     1                  IF KEY = 5 THEN DO;DTHETA= 0.0;GO TO FPROC;END;
```

TERS */

```
      MAIN:PROCEDURE OPTIONS (MAIN);

STMT LEVEL NEST

124     1                IF KEY = 6 THEN DO;DPHI= 2.0  ;GO TO FPROC;END;
129     1                IF KEY = 7 THEN DO;DPHI=-2.0  ;GO TO FPROC;END;
134     1                IF KEY = 8 THEN DO;DPHI= 0.0  ;GO TO FPROC;END;
139     1                IF KEY = 9 THEN DO;DR= 0.25   ;GO TO FPROC;END;
144     1                IF KEY =10 THEN DO;DR=-0.25   ;GO TO FPROC;END;
149     1                IF KEY =11 THEN DO;DR= 0.0    ;GO TO FPROC;END;
154     1                IF KEY =12 THEN DO;DD= 0.2    ;GO TO FPROC;END;
159     1                IF KEY =13 THEN DO;DD=-0.2    ;GO TO FPROC;END;
164     1                IF KEY =14 THEN DO;DD= 0.0    ;GO TO FPROC;END;
169     1           IF CYCLE THEN GO TO PLT1; ELSE GO TO RID;
172     1           FPROC: KEY = 0;
173     1           GO TO TICK;

                     /* INPUT VIEW DATA, AND RECOMPUTE THE TRANSFORMATION PAR

174     1           VIEWD:GET DATA(R,D,THETA,PHI,DR,DD,DTHETA,DPHI);
175     1           VEWCYC: RMAG = R + DR;
176     1                DMAG=D+DD;
177     1               THETA=THETA+DTHETA;
178     1               PHI=PHI+DPHI;
179     1                C=3.141593/180.;

180     1                COS_PHI=COS(PHI*C);
181     1                SIN_PHI=SIN(PHI*C);
182     1                COS_THETA=COS(THETA*C);
183     1                SIN_THETA=SIN(THETA*C);

184     1                URX=COS_PHI*SIN_THETA;
185     1               URY=-SIN_PHI;
186     1                URZ=-COS_PHI*COS_THETA;

                     /*  VIEW DATA UPDATE COMPLETE, GET NEXT INSTRUCTION */
187     1                  IF CYCLE THEN GOTO PLT1;

189     1                GO TO RID;


                     /* VERTEX DATA - READ AND CREATE DATA NODE */

190     1           VERT:GET DATA (X,Y,Z,NEDGES,TAG);

191     1           ALLO:ALLOCATE VERTEX SET(P);
192     1           IF NEDGES ¬= 0 THEN GET LIST((WASTE DO L=1 TO NEDGES));

194     1                XN=X;  YN=Y;  ZN=Z;  NUMBER=TAG;  CON=0;
199     1           VPL(NUMBER) = P;

200     1                IF HEAD=NULL THEN DO;
202     1   1           HEAD,TAIL=P;
203     1   1             GO TO LAB1;
204     1   1                 END;

205     1                TAIL -> NEXT=P;
```

```
        SOLVE: PROCEDURE(HEAD_EDGE,HEAD_FACE);


  STMT LEVEL NEST

                        /* DETERMINING NORMAL VECTOR OF POLYGONS */
    78     1            DO JAK = 2 TO NUM_VERT - 1;
    79     1     1      WI =                    VPL(VERTEX_LABEL_LIST(JAK));
    80     1     1      AA1 = (WI -> XP) - XP;
    81     1     1      AA2 = (WI -> YP) - YP;
    82     1     1      VI =                    VPL(VERTEX_LABEL_LIST(JAK+1));
    83     1     1      BB1 = (VI -> XP) - XP;
    84     1     1      BB2 = (VI -> YP) - YP;
                        /* CROSS PRODUCT */
    85     1     1      KV = KV +(AA1*BB2 - BB1*AA2); /* NORMAL TO POLYGON */
    86     1     1      END;
    87     1            VIEW_TEST = KV*3.;
    88     1            IF VIEW_TEST < 0. THEN ON_OFF_VIEW = 0;
                        /* FACE IS ON VIEW IF VIEW_TEST < 0 ELSE IT IS OFF VIEW
    90     1            Q = NEXT_FACE;
    91     1            IF Q = NULL THEN GO TO RIGHTON;
    93     1            GO TO UP;


                        /* ELIMINATION OF EDGES HIDDEN BY THEIR OWN VOLUME */


    94     1            RIGHTON: E = HEAD_EDGE;
    95     1            AROUND: OFF_VIEW_FLAG = 0;
    96     1            Q = FACE_LIST(1);
    97     1            WAY = FACE_LIST(2);
    98     1            IF ON_OFF_VIEW + WAY -> ON_OFF_VIEW = 2 THEN  OFF_VIEW_FL
   100     1            IF CONCAVITY_TEST = 0 THEN GO TO FINISH;
   102     1            IF ON_OFF_VIEW + WAY -> ON_OFF_VIEW = 1 THEN OFF_VIEW_FL
                        /* IF ON_OFF_VIEW EQUALS 1 EDGE IS OFF VIEW */
   104     1            FINISH: E = NEXT_EDGE;
   105     1            IF E = NULL THEN GO TO THIRD_PART;
   107     1            GO TO AROUND;
   108     1            THIRD_PART: E = HEAD_EDGE;
   109     1            LOOP: IF OFF_VIEW_FLAG = 1 THEN GO TO LONG;
   111     1            P = END(2);
   112     1            WAY = END(1);
                        /* ALLOCATING AN ELEMENT OF THE POINT LIST WHICH CONTAIN
                        AN ON VIEW EDGE */
   113     1            ALLOCATE POINT;
   114     1            XI = XP; YI = YP; XF = WAY -> XP; YF = WAY -> YP;
   118     1            NEXT_POINT = NULL;
   119     1            IF HEADF = NULL THEN TAILF,HEADF = F;
   121     1            ELSE DO; TAILF -> NEXT_POINT = F;
   123     1     1      TAILF = F;
   124     1     1      END;
   125     1            LONG: E = NEXT_EDGE;
   126     1            IF E = NULL THEN GO TO FINIS;
   128     1            GO TO LOOP;
   129     1            FINIS: RETURN;
                        /* RETURN TO CALLING PROCEDURE */
   130     1            END;
```

A;

GO TO THE

O FILL THE

```
        MAIN:PROCEDURE OPTIONS (MAIN);

STMT LEVEL NEST

206     1               TAIL=P;
207     1           LAB1:NEXT=NULL;

                    /*  VERTEX DATA IN AND LINKED, GO TO NEXT INSTRUCTION */

208     1               GO TO RID;

                    /*  DO TRANSFORMS, FILL PLOT BUFFER, AND PLOT  */

209     1           PLT: GET DATA(DELAY);
210     1           PLT1: P=HEAD;
211     1           LOP1:IF(XN=0.&YN=0.&ZN=0.) THEN DO;
213     1   1                       XP,YP=0.0;
214     1   1                         GO TO LAB2;
215     1   1                            END;

216     1               VX=XN+RMAG*URX;
217     1               VY=YN+RMAG*URY;
218     1               VZ=ZN+RMAG*URZ;
219     1               VMAG=SQRT(VX*VX+VY*VY+VZ*VZ);

220     1               UVX=VX/VMAG;
221     1               UVY=VY/VMAG;
222     1               UVZ=VZ/VMAG;

223     1               QMAG=DMAG/(UVX*URX+UVY*URY+UVZ*URZ);

224     1               TX=QMAG*UVX-DMAG*URX;
225     1               TY=QMAG*UVY-DMAG*URY;
226     1               TZ=QMAG*UVZ-DMAG*URZ;
227     1               TMAG=SQRT(TX*TX+TY*TY+TZ*TZ);

228     1               YP= TX*SIN_PHI*SIN_THETA+TY*COS_PHI-TZ*SIN_PHI*COS_T
229     1               XP=TX*COS_THETA+TZ*SIN_THETA;

                    /* TRANSFORM COMPLETE, IF THIS IS NOT THE LAST VERTEX, THE
                                NEXT ONE */

230     1           LAB2:IF NEXT=NULL THEN GO TO GEN;

232     1               P=NEXT;

233     1               GO TO LOP1;

                    /* REINITIALIZE THE DISPLAY BUFFER, GENERATE THE VECTORS,
                            DISPLAY BUFFERS */
234     1           GEN: CALL HIDE(HEADE,HEADQ);
235     1           F = HEADF;
236     1           LAB4: IF FLIP THEN CALL PSGMT(IGDS1,XI,YI,XF,YF);
238     1           ELSE CALL PSGMT(IGDS2,XI,YI,XF,YF);
239     1           IF NEXT_POINT ¬= NULL THEN DO;
241     1   1       F = NEXT_POINT;
242     1   1       GO TO LAB4;
243     1   1                   END;
```

2) & VPL(

```
      MAIN:PROCEDURE OPTIONS (MAIN);


STMT LEVEL NEST

                        /* TERMINATE THE PRESENT DISPLAY, AND PLOT THE NEW BUFFER
 244     1                    IF FLIP THEN DO;
 246     1     1                     CALL EXEC(IGDS1);
 247     1     1                     CALL RESET(IGDS2);
 248     1     1                     END;
 249     1                 ELSE DO;
 250     1     1                     CALL EXEC (IGDS2);
 251     1     1                     CALL RESET(IGDS1);
 252     1     1                     END;
 253     1               FLIP = ¬ FLIP;
 254     1                IF ¬ CYCLE THEN DO;DELAY(CELAY); GOTO RID; END;
 259     1               TICK: CALL RQATN(ALEVEL,KEY,1,ARAY,0,-14);
 260     1                IF KEY ¬= 0 THEN GOTO PROCESS;
 262     1                ELSE GOTO VEWCYC;


 263     1               CSOLVE: Q = HEADQ;
 264     1               PLOP: DO L=1 TO NUM_VERT;
 265     1     1         III = VERTEX_LABEL_LIST(L);
 266     1     1         IF L = NUM_VERT THEN JJJ = VERTEX_LABEL_LIST(1);
 268     1     1         ELSE JJJ = VERTEX_LABEL_LIST(L+1);
 269     1     1         E = HEADE;
 270     1     1         LIT: IF E = NULL THEN DO;
 272     1     2         ALLOCATE EDGE;
 273     1     2         IF HEADE = NULL THEN HEADE,TAILE = E;
 275     1     2         TAILE -> NEXT_EDGE = E;
 276     1     2         TAILE = E;
 277     1     2         NEXT_EDGE = NULL;
 278     1     2         FACE_LIST(1) = Q;
 279     1     2         END(1) = VPL(III);
 280     1     2         END(2) = VPL(JJJ);
 281     1     2         GO TO PLOPE;
 282     1     2         END;
 283     1     1         IF (VPL(III) = END(1) & VPL(JJJ) = END(2)) | (VPL(III) = E
 284     1     1         JJJ) = END(1))  THEN DO;
 285     1     2           FACE_LIST(2) = Q; GO TO PLOPE; END;
 288     1     1         ELSE DO;
 289     1     2         E = NEXT_EDGE; GO TO LIT; END;
 292     1     1         PLOPE: END PLOP;
 293     1               Q = NEXT_FACE;
 294     1               IF Q ¬= NULL THEN GO TO PLOP;
 296     1               CCSLV: CALL SOLVE(HEADE,HEADQ);
 297     1               GO TO RID;

                        /*   NEW STRUCTURE - FREE OLD STORAGE */

 298     1               BS:  IF HEAD=NULL THEN GO TO RID;

 300     1                    P=HEAD;

 301     1               LAB5:W=P;
 302     1                    P=NEXT;
 303     1                    FREE W -> VERTEX;
 304     1                    IF P=NULL THEN GO TO LAB6;
```

```
      MAIN:PROCEDURE OPTIONS (MAIN);

STMT LEVEL NEST

306     1                 GO TO LAB5;

307     1               LAB6:HEAD,TAIL=NULL;
308     1               Q = HEADQ;
309     1               LAP5: W = Q;
310     1               Q = NEXT_FACE;
311     1               FREE W -> FACE;
312     1               IF Q ¬= NULL THEN GO TO LAP5;
314     1               HEADQ,TAILQ = NULL;
315     1               HEADF,TAILF = NULL;
316     1               HEADE,TAILE = NULL;

317     1                    GO TO RID;


                        /*  END RUN   */
318     1               STP: CALL SALRM(IGRAFD);
319     1               PUT EDIT ('DISPLAY TERMINATED FROM CARDS')(X(10),A);
320     1                   RETURN;
321     1                    END MAIN;
```

BIBLIOGRAPHY

Galimberti, R. and Montanari, U., "An Algorithm for Hidden Line Elimination," Communications of the ACM, April 1969, Vol. 12, Number 4, pp. 206-211.