

AD-762 023

HIDDEN SURFACE LINE DRAWING ALGORITHM

Michael Archuleta

Utah University

Prepared for:

Advanced Research Projects Agency
Rome Air Development Center

June 1972

DISTRIBUTED BY:

NTIS

**National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151**



The views and conclusions in this document are those of the author(s) and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency of the U.S. Government.

This document has been approved for public release and sale; its distribution is unlimited.

ia

AD 762 023

HIDDEN SURFACE
LINE DRAWING ALGORITHM

by

Michael Archuleta

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U S Department of Commerce
Springfield VA 22151

June 1972

UTEC-CSc-72-121

This research was supported in part by the University of Utah Computer Science Division and by the Advanced Research Projects Agency of the Department of Defense, monitored by Rome Air Development Center, Griffiss Air Force Base, New York 13440, under contract F30602-70-C-0300.

ACKNOWLEDGMENTS

The author is grateful to Dr. Gary Watkins for his guidance, inspiration, and continuous encouragement. Needless to say, the world of computer graphics is more exciting than ever since the Watkins' Algorithm surfaced in a sea of occluded rasters.

TABLE OF CONTENTS

	Page
Acknowledgments	ii
List of Figures	iv
Abstract	v
INTRODUCTION	1
WATKINS' ALGORITHM	1
LINE DRAWING ALGORITHM	3
INTCLP	11
POLMAK	13
EDGMAK	19
PACK	31
CROSS	33
DRAW	37
DRAWM - DRAWV - SETLIN	39
OUTSTR - LDLPT - LDRPT - STLPT - STRPT - LOD2HF - STR2HF - MYMAXO - MYMINO	41
UNPACK	43
XINTER	45
ZVALUE	47
INFREE - GETVAR - LSTSET - GETBLK - RETBLK	49
HIDDEN	51
Illustrations	65
References	69

LIST OF FIGURES

Figure		Page
1:	Two spheres at 256 scan line resolution	65
2:	Building processed at 256 scan line resolution	66
3:	Building processed at 1024 scan line resolution	67
4:	1600 edge object with and without hidden surface removal at 256 scan line resolution	68

ABSTRACT

This paper describes a fast procedure in processing hidden surface pictures with the output in vector form. The program has been written expressly for a Decsystem 10 and has performed successfully on three different installations.

The algorithm which is being used is a modification to the Watkins' Algorithm. Consequently, the program has inherited many key features which provide for speed, compactness, and flexibility.

INTRODUCTION

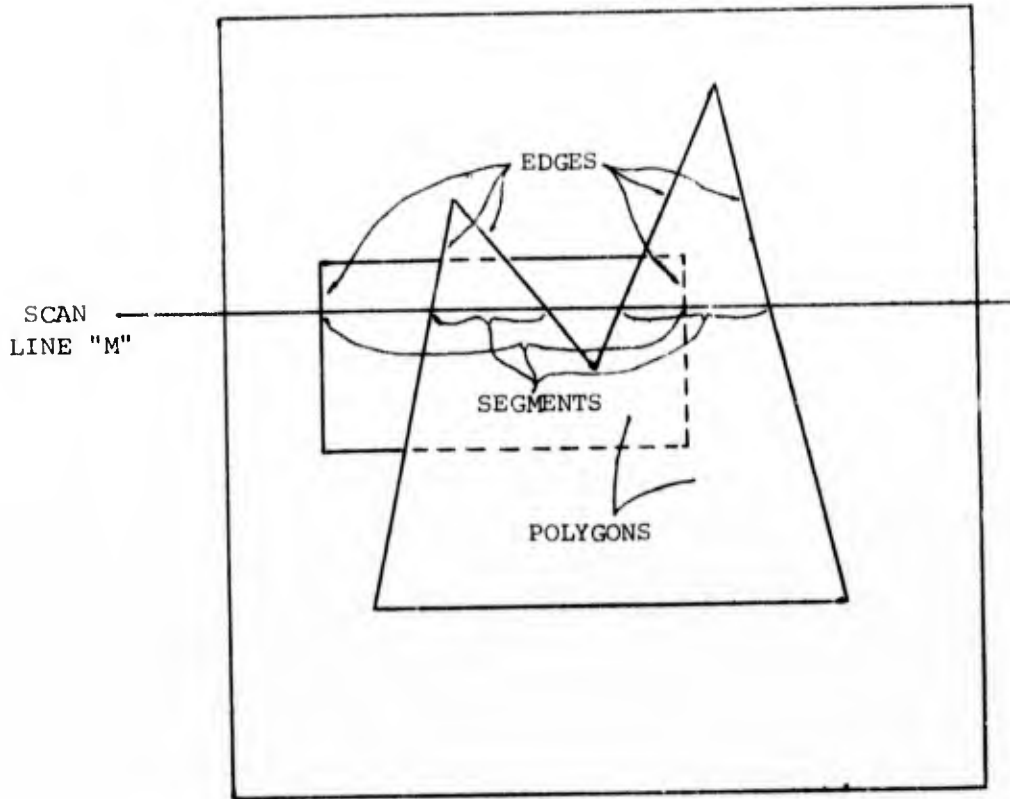
There has long been a need for a very fast hidden surface line drawing algorithm. At the University of Utah, Dr. Gary Watkins developed a very efficient continuous tone hidden surface algorithm which could be implemented in hardware to produce pictures at rates of 30 frames per second. It turns out that with a few minor modifications, the same algorithm can be adapted to produce line drawings.

There were two requirements that had to be fulfilled: (1) A procedure had to be developed that would handle horizontal edges, since these were discarded during preframe processing and (2) Information would have to be extracted from the existing algorithm which could generate lines. It will be necessary to briefly explain Watkins' Algorithm in order to understand the line drawing procedure more clearly.

WATKINS' ALGORITHM

Hidden surface pictures are processed by using a raster approach originally developed by the Romney Hidden Surface Algorithm. The user is required to describe his objects in terms of polygons and associated edges. Preframe processing clips all of these edges so that they are within the view area, and then sorts them with respect to Y.

On any given scan line, pairs of edges that belong to the same polygon are linked together to form segments. For the current scan line, these segments are compared to determine which set of segments is visible.



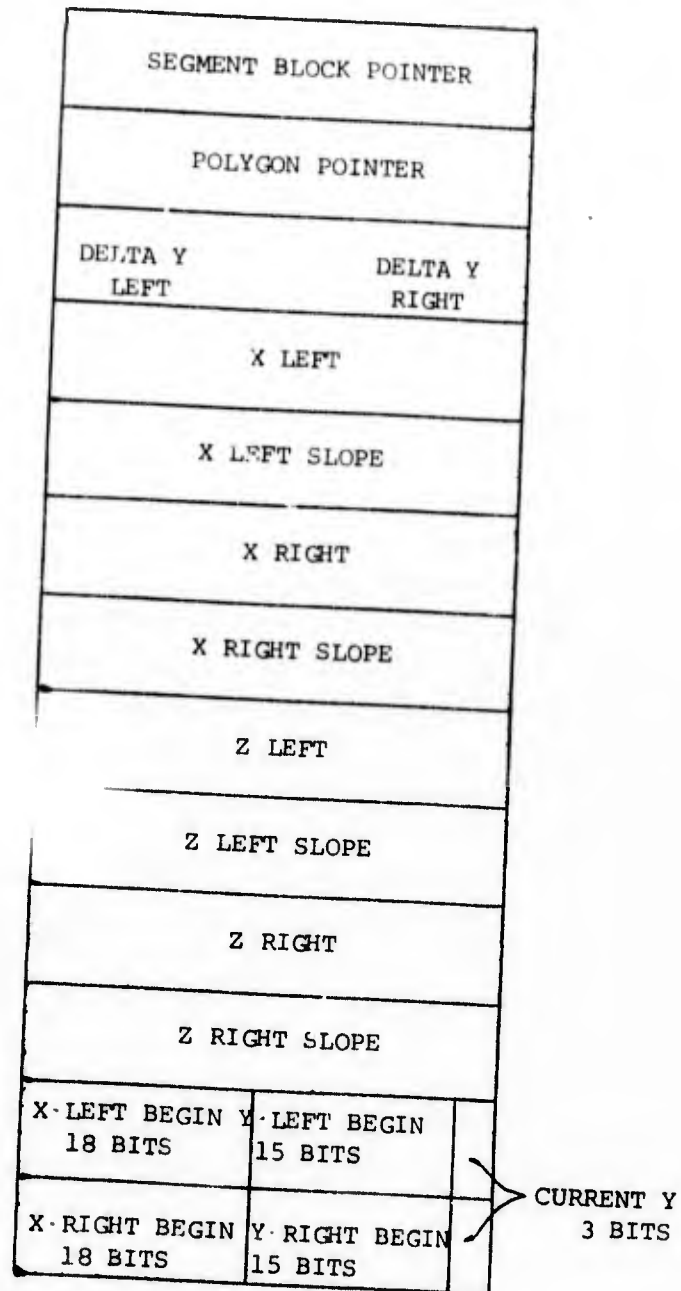
RELATIONSHIP BETWEEN EDGES AND SEGMENTS

From this point, the information is passed to a shader which produces the continuous tone for a visible segment. After a scan line has been completed, all segments are updated to the next scan line. Some minor sorting is performed at this time, due to exiting edges and new entering edges.

LINE DRAWING ALGORITHM

During preframe processing, all horizontal edges are saved in a free storage list with a pointer to the polygon to which they belong. These horizontal edges are also sorted in Y. When the hidden surface algorithm comes up with a visible segment, this segment is checked to see if the polygon to which it belongs has a horizontal edge existing within the region of the visible span. One minor problem arises with this type of an approach. Preframe processing stores all edges so that the vertices lie between scan lines. Two polygons which shared the same horizontal edge were processed in a way that the hidden surface algorithm physically separated the two polygons which led to the two displayed horizontal edges. In order to keep the sorting of these edges as simple as possible, a simple rule was formed: All horizontal edges are visible on the current scan line to which they really belong, and possibly on the previous scan line. Thus, when two polygons share an edge, the edge will be displayed twice, but the two lines will be coincident.

Since each segment is composed of a left and a right edge, it is very easy to add flags to the segment block indicating if the edge is



SEGMENT BLOCK

really visible on this scan line. Every time the hidden surface algorithm passes a visible segment over to the "shader," a counter in the segment block is incremented for the left or right edge of the segment, stating that the edge has been visible for "N" scan lines. Two conditions now arise stipulating when a line should be physically drawn. If the edge is exiting on this scan line, then draw. If the edge was visible on the previous scan line and not this scan line, then draw. Whenever an edge is drawn, the counter inside the segment block is cleared. The first time that an edge becomes visible, the starting point of the edge is stored so that the end point of the line is available. This holds true for edges which are intermittently visible. Following is a picture of the segment block.

Unfortunately, there are three minor flaws which exist as a result of letting the Watkins' Algorithm do the hidden surface processing.

It was stated earlier that preframe processing stores edges so that vertices lie between scan lines. Consequently, a line is drawn so that it extends from a point that lies between scan lines. The difficulty arises when two adjacent edges are visible on a scan line, but one of the edges crosses in front of the other edge on the next scan line. Drawing to the midpoint between scan lines results in lines extending a little over other edges. Fortunately, there is enough information available at this time to capture this problem and draw the correct line without overlap for edges that are visible on the previous scan line. Edges which enter for the first time run the risk of creating overlap. This problem could be handled by storing the previous scan line of information.

The second problem deals with intersection of surfaces. Since an imaginary edge exists at points of intersection, there does not exist a segment block where a counter could be incremented to specify that the edge of intersection has been visible. There are two solutions to this dilemma. Whenever an intersection is encountered by the "shader," it will display a dot. The second solution is to create a segment block which contains similar edge information, and then process the line. This would require some overhead in programming, but the aesthetics of the lines would be worth the effort.

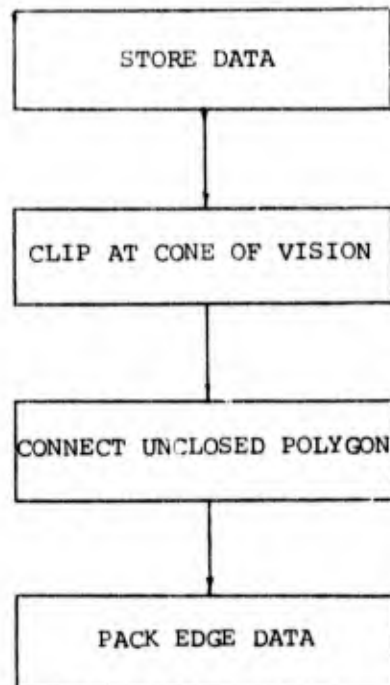
The third problem deals with the resolution of the picture. The package which is distributed works with a resolution of 256 scan lines in Y. Thus, objects may look a little distorted when displayed, due to the perspective transformation and roundoff to the scan line. The user can compensate for this by changing the source code so that the resolution is greater. Computation time will increase linearly. In file FSTLIN.F4, change BUCKY(0/255), HBUCKY(256) to BUCKY(0/1023), HBUCKY(1024) and in file EDGLIN.MAC, change YRESOLUTION=256 to YRESOLUTION=1024, if the user wishes to increase the Y resolution to 1024. Increasing the resolution will also decrease the amount of overlap when edges cross. There is no point in making the resolution greater than the resolution of the display device.

There are a number of very useful features concerning the hidden surface line drawing algorithm. Since the program works on a low resolution, very little time is devoured in processing a picture. Pictures containing 1000 edges take about 15 seconds of CPU time on a Decsystem 10. Time-sharing users don't have to wait very long for a picture to be processed.

Code and data for the algorithm require about 2000 words of memory. Consequently, the program will do little page faulting during execution. Each edge requires four words of memory from the free storage list, thus permitting complex pictures to reside in a very limited area when core is at a premium.

Perhaps the modification of the Watkins' Algorithm was not the most ideal approach to hidden surface processing for line drawings. However, it is believed that speed, compactness, and flexibility of the procedure will keep many users happy for the present.

Following is a brief explanation of how an edge is processed. The data describing the end point is stored into working registers. This data is then clipped to the four planes of the cone of vision ($Z = Y$, $Z = -Y$, $Z = -X$, $Z = X$). The processed edge is checked to see if it lies within the cone of vision. If the edge is visible and not a horizontal, then the data for the edge is packed into a four word block for hidden surface processing.



This page defines the registers which are used inside the routine EDGMAK. It should be noted that accumulators are not saved, with the exception of AC7. All arithmetic is performed in AC0, AC1, and AC2, while indexing is performed in AC3, AC4, AC5, and AC6. The only sub-routines external to EDGMAK are IFIX, VECTO, MOVETO, and GETVAR, which returns a block from free storage. There are three entry points to this routine: EDGMAK, POLMAK, and INTCLP. Data is passed over in global areas FREE, SCOPE, CLIP2, CLIPH, CLIP3, EYES, and COLOUR. Data is sent out to the world via global areas INTENS, and FREE.

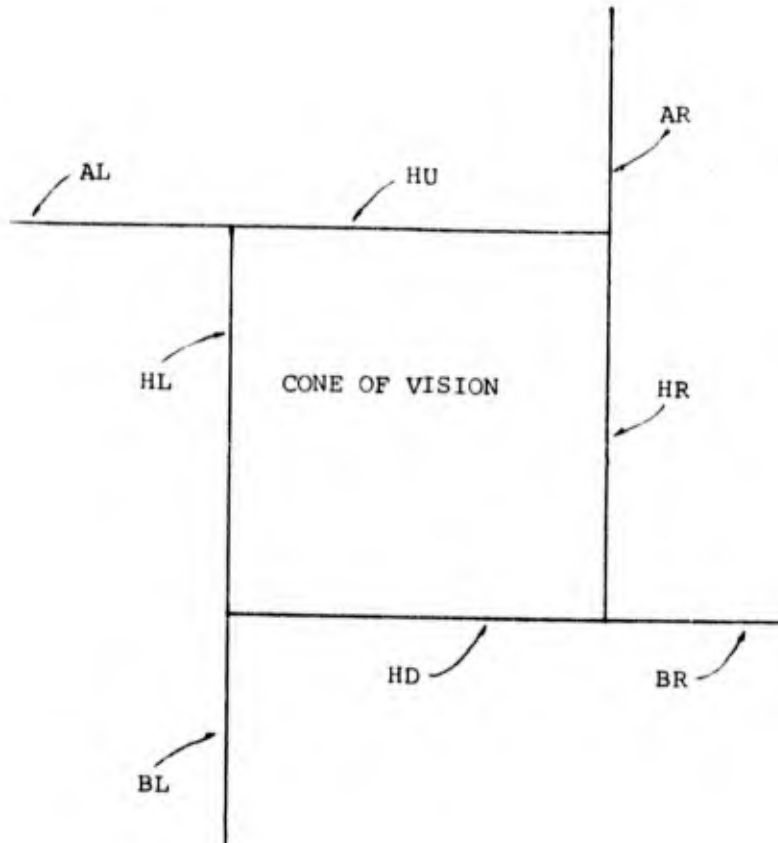
```

TITLE EDGMK
ENTRY EDGMK,POLMAK,INTCLP
EXTRN IFIX,FREE,GETVAR,SCOPE
EXTRN CLIP2,CLIP1,CLIP3
YRESOLUTION=^D256 ;CHANGE THIS NUMBER IF MORE
;PRECISION DESIRED IN Y DIRECTION. KEEP TO A POWER OF TWO. 512,1024,?
**048
POLYPT: Z ;POLYGON TO WHICH THIS EDGE BELONGS
FLAGS=7 ;THE FOLLOWING THIRTEEN FLAGS WILL BE STORED IN AC7
AL=400000 ;COUNTER FOR TOP LEFT CORNER
AR=200000 ;COUNTER FOR TOP RIGHT CORNER
BL=100000 ;COUNTER FOR BOTTOM LEFT CORNER
BR=400000 ;COUNTER FOR BOTTOM RIGHT CORNER
BU=200000 ;COUNTER FOR TOP WINDOW
BD=100000 ;COUNTER FOR BOTTOM WINDOW
BL=4000 ;COUNTER FOR LEFT WINDOW
BR=2000 ;COUNTER FOR RIGHT WINDOW
XT=1000 ;POLYGON PASSES ON THE RIGHT OF ORIGIN
YV=400 ;POLYGON PASSES ON TOP OF ORIGIN
ZN=200 ;POLYGON PASSES IN FRONT OF ORIGIN
HOLDU=100 ;HOLDU.OR.HOLDD
;XF=40 ;CX=0 IS TRUE FLAG
CX=CLIP2 ;X COEFFICIENT OF PLANAR EQUATION
CY=CLIP2+1 ;Y COEFFICIENT OF PLANAR EQUATION
CZ=CLIP2+2 ;Z COEFFICIENT OF PLANAR EQUATION
CD=CLIP2+3 ;CONSTANT IN PLANAR EQUATION
COLOR=CLIP2+4 ;COLOR OF POLYGON
IPRIORITY=CLIP2+5 ;PRIORITY OF POLYGON (0-15)
HID=CLIP3+3 ;INTENSITY COMPONENT
XPT1=CLIP3 ;X BEGIN
YPT1=CLIP3+1 ;Y BEGIN
ZPT1=CLIP3+2 ;Z BEGIN
NPT1=CLIP3+3 ;INTENSITY BEGIN
XPT2=CLIP3+4 ;X END
YPT2=CLIP3+5 ;Y END
ZPT2=CLIP3+6 ;Z END
;PT2=CLIP3+7 ;INTENSITY END
LASEDG=CLIP3+10 ;LAST EDGE FLAG
XR: Z ;X RESOLUTION
YR: Z ;Y RESOLUTION
BUCKY=SCOPE ;Y BUCKET SORT ARRAY
HBUCKY=SCOPE+YRESOLUTION ;HORIZONTAL EDGE BUCKET SORT ARRAY
VX: BUCK 10 ;X VERTICE ARRAY
VY: BUCK 10 ;Y VERTICE ARRAY
VZ: BUCK 10 ;Z VERTICE ARRAY
YX=3 ;INDEX POINTER FOR BEGIN POINT
YB=5 ;DITTO
YZ=4 ;INDEX POINTER FOR END POINT
YE=6 ;DITTO
TEMPX: Z ;REGISTER FOR CLIPPED X
TEMPY: Z ;REGISTER FOR CLIPPED Y
TEMPZ: Z ;REGISTER FOR CLIPPED Z
; ALL ARITHMETIC IS PERFORMED IN AC0,AC1,AC2
; AC7-17 ARE NOT TO BE TOUCHED SINCE THEY ARE USED IN THE CALLER
^L

```

INTCLP

INTCLP is responsible for initializing the appropriate flags for hidden surface processing. XR and YR are used in perspective scaling transformations. BUCKY and HBUCKY have to be filled with ones prior to each new frame.



```

EXTERN EYES,INTENS
LISTMD=EYES+1
IENHI=INTENS ;INTENSITY PASSED TO ALGORITHM
IENLO=INTENS+1 ;INTENSITY PASSED TO ALGORITHM
BACKG=INTENS+2 ;BACKGROUND PERCENTAGE
IFX1: 1000 ;RESOLUTION IN X
IFY1: YRESOLUTION ;RESOLUTION IN Y
IBACK=EYES+3 ;BACKGROUND INTEN USED BY PROGRAM
IENHIH=EYES+4 ;HI INTEN USED BY PROGRAM
IENLOH=EYES+5 ;LO INTEN USED BY PROGRAM
IX=EYES+6 ;X RESOLUTION USED BY HIDDEN
IY=EYES+7 ;Y RESOLUTION USED BY HIDDEN
LISTST=FREE+3 ;ADDRESS OF FIRST EDGE BLOCK
INTCLP: Z
    MOVE @0(16)
    MOVEM PICTURE#
    SETDM POLYPT ;CLEAR POLYGON POINTER
    SETDM LISTST ;CLEAR LIST START ADDRESS
    SETZM LISTMD ;CLEAR LIST MADE UP FLAG
    MOVE 1,IFY1
CLRBUK: SETDM BUCKY-1(1)
    SETDM HBUCKY-1(1)
    SOJG 1,CLRBUK
FIXRES: MOVE IFX1
    SOJ
    MOVEM IX ;IX_IFX1-1
    ASH 10
    MOVEM XR ;XR_IX*256
    MOVE IFY1
    ASH -1
    MOVEM YR ;YR_IFY1/2
    MOVE IFY1
    SOJ
    MOVEM IY ;IY_IFY1-1
    MOVEM YRESOLUTION ;GET IT
    ASH -11
    AJJ 0,0
    MOVNS 0,0
    HRRM DRAW4 ;-1 FOR 256, -2 FOR 512, -3 FOR 1024
    MJVE INTENS+3 ;GET X SCREEN SIZE
    ASH -7
    HRRM DRAW2
    MOVE INTENS+4 ;GET Y SCREEN SIZE
    ASH -7
    HRRM DRAW3
    JRA 16,(16)

```

^L

POLMAK

POLMAK clears the clipping flags and increments the polygon count. These macros perform some simple algebraic manipulations. The Y intersection is basically: $TEMPY=ALPHA*DY+VY$ in the case of clipping to the plane $Z = v$. The X intersection is: $TEMPX=ALPHA*DX+VX$ in the case of clipping to the plane $Z = Y$.

```

POLYMAX: Z
AUS      POLYPT
SETZM    FLGLOC# ;CLEAR CLIPPING FLAGS
JRA      16,(16)
DEFINE   DIFFER (A,B,C) ;A_3-C
<MOVE    B
SUB      C
MOVEM    A>
DEFINE   FIXJRT2 (A) ;PERFORMS T1.XDR.T2
<MOVE    T1
XDR      T2
JUMPGE   A> ;JUMP IF EDGE DID NOT CROSS
DEFINE   CLIPPER (DEL1,IMP1,BEG1,DEL2,IMP2,BEG2,C)
<JSR     ALPHA ;GET THE ALPHA
INTSCT   DEL1,IMP1,BEG1
MOV'C    1,TEMPZ
INTSCT   DEL2,IMP2,BEG2>
DEFINE   INTSCT (DELTA,LOC,BEGIN)
<MOVE    1,DELTA
MUL      1,0
ASHC     1,1
ADD      1,BEGIN
MOVEM    1,LOC> ;LOC=DELTA*ALPHA+BEGIN
ALPHA:   Z ;THIS SUBROUTINE CALCULATES ALPHA
MOVE     2,T1
SUB      2,T2
MOVE     T1
SETZ     1,
ASHC     0,-1
DIV      0,2
JRST     @ALPHA ;ALPHA=T1/(T1-T2)
DEFINE   SUM (A,B,C) A_3+C
<MOVE    B
ADD      C
MOVEM    A>

```

^L

SLOPE evaluates the expression: $SLO = (BEGVALUE - ENDVALUE) / DELTAY$.

BEGPNT evaluates the expression: $BEG = STARTVALUE + SLOPE / 2$.

CORNER is responsible to evaluate the X, Y, and Z value at a corner when a polygon wraps around said point. The planar equations are used in this case. $VX = CD / (CX - CY - CZ)$ where $-X = Z = Y$. Consequently, $VY = -VX$ and $VZ = -VX$.

```

DEFINE SLOBEG (START,BEGVAL,1SLOPE,END)
<SLOPE 1SLOPE,END,START
BEGPNT  BEGVAL,START>
DEFINE SLOPE (SLO,END,BEG)      ;EVALUATES SLOPE
<MOVE  BEG
SUB  END
IDIV  2      ;AC2 HAS DELTAY
MOVEM SLO>
DEFINE BEGPNT (BEG,START)      ;DETERMINES BEGINNING VALUE
<JMPGE 0,.,+2  ;SKIP IF SLOPE POSITIVE
ADJ  0
ASH  0,-1  ;ADD HALF OF THE SLOPE
ADD  0,START
MOVEM 0,BEG>
DEFINE CORNER (A,OPER1,OPER2,B,C)
<MOVN  2,CD
MOVE  1,CX
OPER1 1,CY
OPER2 1,CZ
FDVR  2,1
JSA  16,IFIX
JUMP  2
MOVEM VX+A
MOV'B VY+A
MOV'C VZ+A>
DEFINE FORXYZ (A,B,C)
<IRP A,<MOVE  V'A-1(C)
MOVEM  V'A+B-1>>
DEFINE TRNSFR (A,B)
<FORXYZ <X,Y,Z>,A,B>
DEFINE LODIMP (A) ;LOADS TEMPORARY VALUES INTO VERTECE ARRAY
<MOVE  TEMPX
MOVEM  VX+A
MOVE  TEMPY
MOVEM  VY+A
MOVE  TEMPZ
MOVEM  VZ+A>
DEFINE STUFF (A,%B) ;DETERMINES WHICH END POINT TO STUFF IN
<MOVE  T1
JUMPL %B
LODIMP 3
JRST  A
LODIMP 2>

```

%B:
^L

PERSP performs the perspective transformation. The equation is:
$$\text{PERS} = (\text{VAL} * \text{RES}) / \text{ZVAL} + \text{RES}$$
 where VAL may be either X or Y. RES was calculated inside INTCLP. Note that if the user specified the X resolution to be 1024, the RES would be $512 * 511$. Consequently, the use of double precision arithmetic was used so that the user could provide X and Y values up to $2^{35} - 1$.

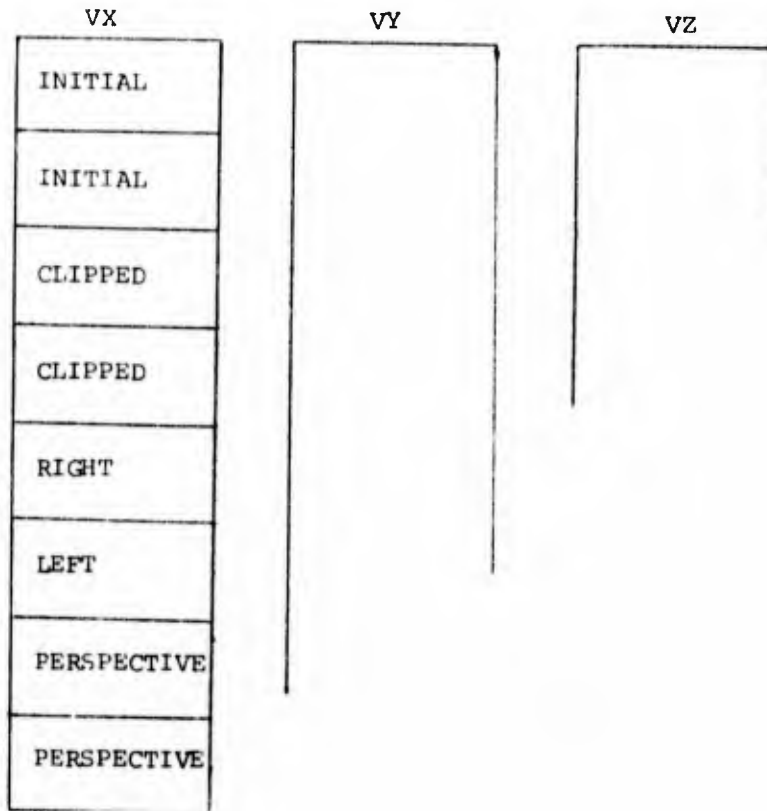
ZINVRS performs a $1/Z$ operation, where 1 is essentially $2^{30} - 1$.

```
DEFINE DUBSTR (A,B)
<MOVEM 1,V'A+B ;STORE ENDPOINT INTO WORKING REGISTER
MOVEM 1,V'A+B+2>
DEFINE STOREZ (A,B)
<MOVE 1,A
DUBSTR Z,B>
DEFINE STORXY (A,B,C)
<MOVE 1,A
DUBSTR B,C>
DEFINE PERSP (PERS,VAL,RES,ZVAL) ;PERSPECTIVE TRANSFORMATION
<MOVE VAL
MUL RES
DIV ZVAL
ADD RES
MOVEM PERS>
DEFINE ZINVR (PERS,ZVAL) ;INVERSE Z VALUE
<HRLJI 7777
SKIPG 1,ZVAL ;SKIP IF Z IS STRICTLY POSITIVE
MOVEI 1,1
IDIV 0,1
MOVEM PERS>
```

^L

EDGMAK

The calling program passes the begin point X, Y, Z and end point X, Y, Z. These numbers are stuffed into working registers called VX, VY, VZ. Locations 1 and 2 of these registers contain the initial values; locations 3 and 4 contain the clipped values; 5 and 6 retain the most recent left or right clipped value; and locations 7 and 8 contain the transformed perspective value.



EDSMAN: Z

```

MOVEM 7,AC7#
MOVE  FLAGS,FLGLOC

```

```

; THE FOLLOWING CODE PLACES THE BEGIN AND END POINTS OF THE
; EDGE INTO WORKING REGISTERS. DX,DY,DZ,DN ARE THE DIFFERENCES
; OF THE BEGIN AND END POINTS.

```

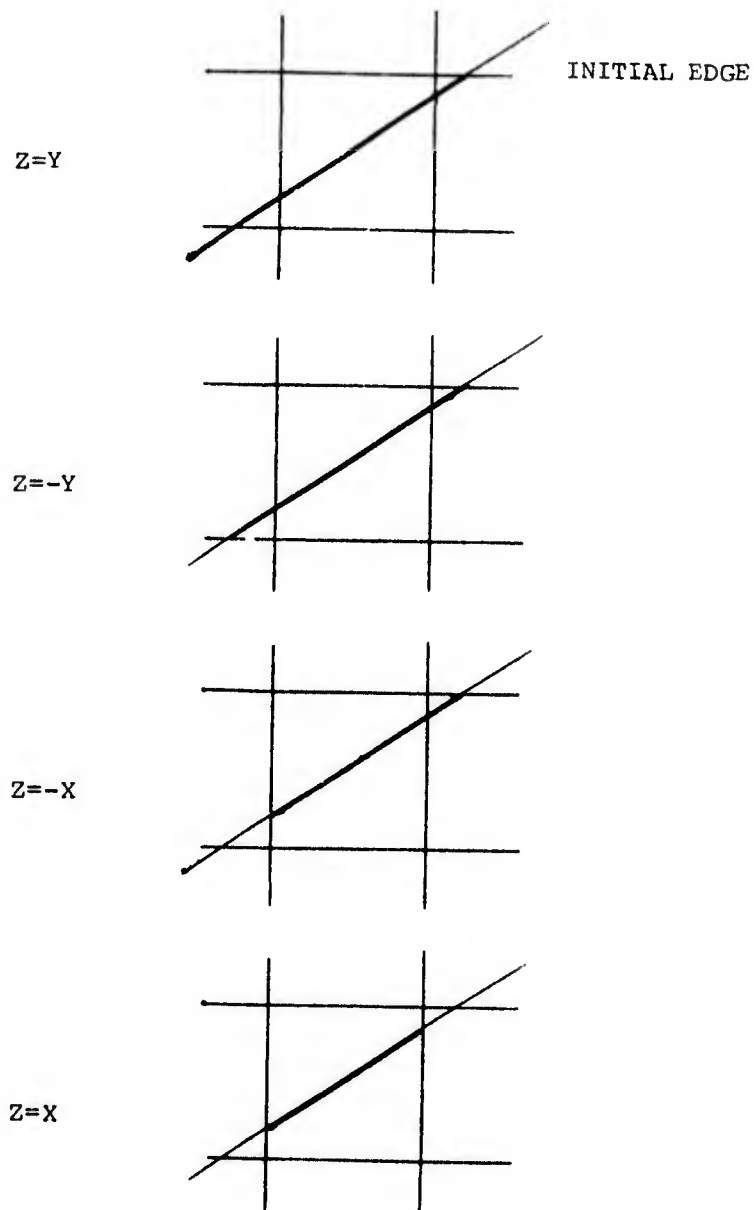
```

STORXY XPT1,X,0
STORXY XPT2,X,1
STORXY YPT1,Y,0
STORXY YPT2,Y,1
STOREZ ZPT1,0
STOREZ ZPT2,1
MOVEI  YB,1
MOVEI  YE,2
DIFFER DX#,VX+1,VX
DIFFER DY#,VY+1,VY
DIFFER DZ#,VZ+1,VZ
SETZM  T3#
SETZM  T4#

```

^L

The TLXORT2 check determines if the edge crosses the plane which is going to be clipped to. CLIPPER evaluates the XYZ coordinate of intersection. TLXORT2 and STUFF now determine if the new clipped point should be stuffed into the begin or end clipped point location (V*+3,4).



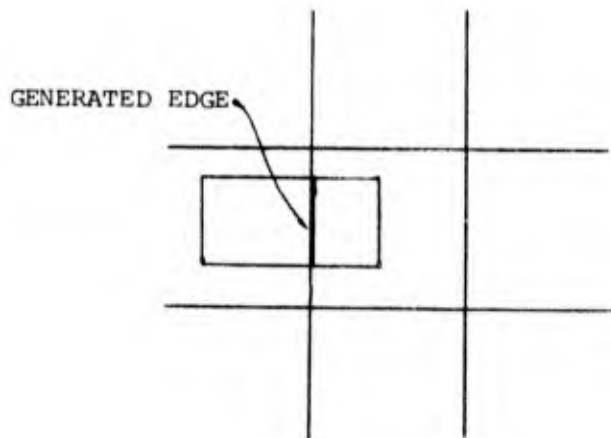
: THIS PAGE OF CODE PERFORMS THE CLIPPING AT THE FOUR PLANES
 : OF THE CONE VISION. THE ORDER OF CLIPPING IS Z=Y, Z=-Y, Z=X, Z=-X.

```

DIFFER T1#,VZ,VY
DIFFER T2#,VZ+1,VY+1
TIXORT2 P23
CLIPPER DY,TEMPY,VY,DX,TEMPX,VX,EM
STUFF P25
P26: MOVN TEMPZ
CAML TEMPX
IRC FLAGS,AL ;COMP ABOVELEFT COUNTER IF EDGE CROSS
**ED
MOVN TEMPX
CAMG TEMPZ
IRC FLAGS,HU ;COMP HOLDUP IF EDGE CROSSED WINDOW
P28: SUM T1,VZ,VY
SUM T2,VZ+1,VY+1
TIXORT2 P38
CLIPPER DY,TEMPY,VY,DX,TEMPX,VX,NM
SUM T1,VZ+2,VY+2
SUM T2,VZ+3,VY+3
TIXORT2 P36
STUFF P36
P36: MOVE TEMPZ
CAMG TEMPX
IRC FLAGS,BR ;COMP BLOWRIGHT COUNTER IF EDGE CROSSE
(*)
MOVN TEMPX
CAMG TEMPZ
IRC FLAGS,HD ;COMP HOLDDOWN IF EDGE CROSSED BOT WIND
**d
P38: DIFFER T1,VZ,VX
DIFFER T2,VZ+1,VX+1
TIXORT2 P43
CLIPPER DX,TEMPX,VX,DY,TEMPY,VY,EM
DIFFER T1,VZ+2,VX+2
DIFFER T2,VZ+3,VX+3
TIXORT2 P46
ADS T4
STUFF P46
P46: MOVE TEMPZ
CAMG TEMPY
IRC FLAGS,AR ;COMP ABOVERIGHT IF EDGE CROSSED
P48: SUM T1,VX,VZ
SUM T2,VX+1,VZ+1
TIXORT2 P58
CLIPPER DX,TEMPX,VX,DY,TEMPY,VY,NM
SUM T1,VX+2,VZ+2
SUM T2,VX+3,VZ+3
TIXORT2 P56
ADS T3
STUFF P56
P56: MOVN TEMPZ
CAML TEMPY
IRC FLAGS,BL ;COMP BELOWLEFT COUNTER IF EDGE CROSS
**ED
^L

```

Once all the clipping has been performed, a check is made to see if the edge lies within the window. If it does, then additional checks must be made to see if the processed edge crossed either the left or right window boundary. If so, then store the clipped value into either location 4(right) or 5(left) of the V^* register. If this is an even time that an edge has crossed a boundary, then generate a new edge at the boundary.



```

P53:  MOVW  VY+2
      CAMLE VZ+2
      JRST P75      ;JUMP IF CLIPPED EDGE IS OUTSIDE WINDOW
      MOVW  VX+2
      CAMLE VZ+2
      JRST P75      ;JUMP IF CLIPPED EDGE IS OUTSIDE WINDOW
      MOVEI YB,3
      MOVW  VZ+2
      CAMLE VX+2
      MOVEI YB,4
      MOVE  T3
      JUMPG P68      ;JUMP IF EDGE CROSSED LEFT WINDOW
P64:  MOVEI  YE,3
      MOVE  VZ+2
      CAMLE VX+2
      MOVEI YE,4
      MOVE  T4
      JUMPG P70      ;JUMP IF EDGE CROSSED RIGHT WINDOW
P66:  MOVEI  YX,3
      MOVEI  YZ,4
      MOVEI  P75
      HRRM  JUMPER
      JRST  P90      ;G) PROCESS THIS EDGE
P68:  TRCE  FLAGS,HL
      JRST  P72
      TRNSFR 5,YB
      JRST  P64
P70:  TRCE  FLAGS,HR
      JRST  P74
      TRNSFR 6,YE
      JRST  P66
P72:  MOVE  YX,YB
      MOVEI YZ,5
      MOVEI P64
      HRRM  JUMPER
      JRST  P90
P74:  MOVE  YX,YE
      MOVEI YZ,6
      MOVEI P66
      HRRM  JUMPER
      JRST  P90
^L

```

If this is the last edge of the polygon, then all the clipping flags must be checked to see if an unclosed polygon currently exists. One of the pieces of information required is the intersection of the polygon with the X, Y, and Z axis.

The flags are interrogated first to see if any processing need be done on the left side of the window, then similarly for the right side of the boundary.

```

P 75:  SKIPL  LASEDG  ;SKIP IF THIS IS THE LAST EDGE OF POLYGON
        JRST  P98    ;NOT LAST EDGE
        TRNN  FLAGS,AL+BL+AR+BR
        JRST  P98    ;JUMP IF ALL EDGES ARE PROCESSED
        TRNE  FLAGS,HU+HD
        TRJ   FLAGS,HDHU    ;SET IF HU.DR.HD IS TRUE
        SKIPN CX
        TRJ   FLAGS,CXF     ;SET IF CX=0 IS TRUE
        DEFINE PLNINT (AXIS,FLG)
        <MOVE  AXIS
        XOR   CD
        JUMPGE .+2
        TRJ   FLAGS,FLG>   ;SET IF FLAG INTERSECTS ON +AXIS
        PLNINT CX,XT
        PLNINT CY,YV
        PLNINT CZ,ZN
        TRC   FLAGS,-1     ;COMPLIMENT THE WORLD
;GROUND RULES FOR TYING UP THE LOOSE ENDS:
;A "1" IMPLIES FALSE
;A "0" IMPLIES TRUE
;TRNN IMPLIES SKIP IF A FALSE
;TRNE IMPLIES SKIP IF A TRUE
        MOVEI  YX,1
        MOVEI  YZ,2
        TRNE  FLAGS,HL
        JRST  P80    ;JUMP IF NOTHING TO BE DONE ON THE LEFT
        MOVEI  YZ,5
        MOVEI  P80
        HRRM  JUMPER
        TRNE  FLAGS,AL
        JRST  .+4
        TRNE  FLAGS,YV
        TRNE  FLAGS,BL
        JRST  P83    ;CONNECT TO TOPLEFT
        MOVEI  YX,2
        JRST  P84    ;CONNECT TO BOTTOMLEFT
P 30:  TRNE  FLAGS,HR
        JRST  P81    ;JUMP IF NOTHING TO BE DONE ON THE RIGHT
        MOVEI  YX,1
        MOVEI  YZ,6
        MOVEI  P81
        HRRM  JUMPER
        TRNE  FLAGS,AR
        JRST  .+4
        TRNE  FLAGS,YV
        TRNE  FLAGS,BR
        JRST  P86    ;CONNECT TO TOPRIGHT
        MOVEI  YX,2
        JRST  P88    ;CONNECT TO BOTTOMRIGHT

```

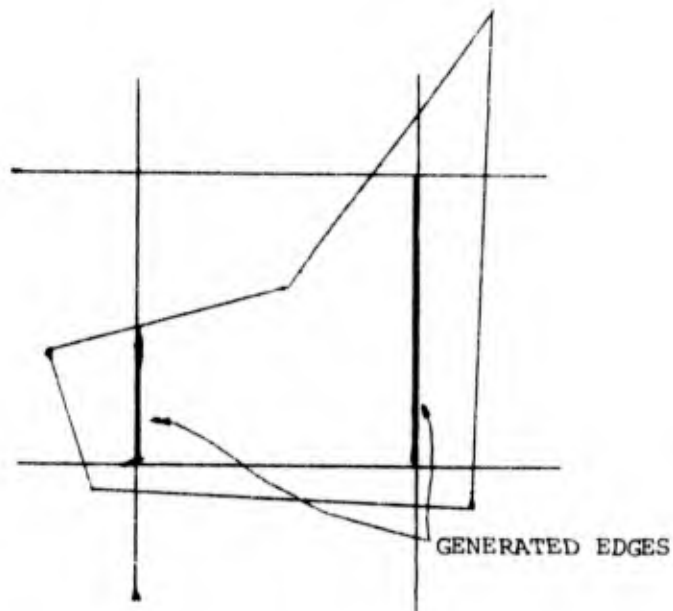
^L

The equation which is used to ascertain connect to top and bottom left is:

$$\begin{aligned}
 & (.NOT.HL.AND.AL.AND.BL.AND. \\
 & ((HU.OR.HD).AND.(.NOT.XT.OR.(CXF)))) .OR. \\
 & (.NOT.(HU.OR.HD).AND.ZN)) .
 \end{aligned}$$

The equation which is used to ascertain connect to top and bottom right is:

$$\begin{aligned}
 & (.NOT.HR.AND.AR.AND.BL.AND.(((HU.OR.HD).AND.(XT.OR.CXF))) \\
 & .OR.(.NOT.(HU.OR.HD).AND.ZN)) .
 \end{aligned}$$



```

P51:  MOVEI  YX,1
      MOVEI  YZ,2
      MOVEI  P82
      HRRM   JUMPER
      TRNE   FLAGS,HL
      TRNE   FLAGS,AL+BL
      JRST   P82      ;NOTHING TO CONNECT ON THE LEFT
      TRNE   FLAGS,ZN
      JRST   F2
      TRNE   FLAGS,HDHU
      JRST   P83      ;CONNECT TO TOPANDBOTTOMLEFT
F2:   TRNE   FLAGS,HDHU
      JRST   P82      ;NOTHING TO CONNECT ON THE LEFT
      TRIN   FLAGS,XT
      TRIN   FLAGS,CXF
      JRST   P83      ;CONNECT TO TOPANDBOTTOMLEFT
P52:  TRIN   FLAGS,HR
      JRST   P98      ;NOTHING TO CONNECT ON THE LEFT
      MOVEI  YX,1
      MOVEI  YZ,2
      MOVEI  P98
      HRRM   JUMPER
      TRNE   FLAGS,AR+BR
      JRST   P98      ;NOTHING TO CONNECT ON THE RIGHT
      TRNE   FLAGS,ZN
      JRST   F3
      TRNE   FLAGS,HDHU
      JRST   P86      ;CONNECT TO TOPANDBOTTOMRIGHT
F3:  TRNE   FLAGS,HDHU
      JRST   P98      ;NOTHING TO CONNECT ON THE RIGHT
      TRNE   FLAGS,XT
      TRIN   FLAGS,CXF
      JRST   P86      ;CONNECT TO TOPANDBOTTOMRIGHT
      JRST   P98      ;NOTHING TO CONNECT ON THE RIGHT
^L

```

This sets up the code for the appropriate corner calculation.

When the program has arrived at P90, the end points of the edge to be processed are held in index locations YX and YZ. Perspective transformation yields $0.LEQ.VX.LEQ.FRAMEX-1$ and $0.LEQ.VY.LEQ.FRAMEY$.

The EXCH is performed to insure that V*+7 is the begin point (has the highest Y value).

HORZED makes up a horizontal edge block for the hidden surface algorithm. This edge is then entered in the HBUCKY list at the appropriate scan line on which it exists. It is also entered on the previous scan line. Following is a picture of the horizontal edge block.

POLYPT	NEXT EDGE
XLEFT	XRIGHT

```

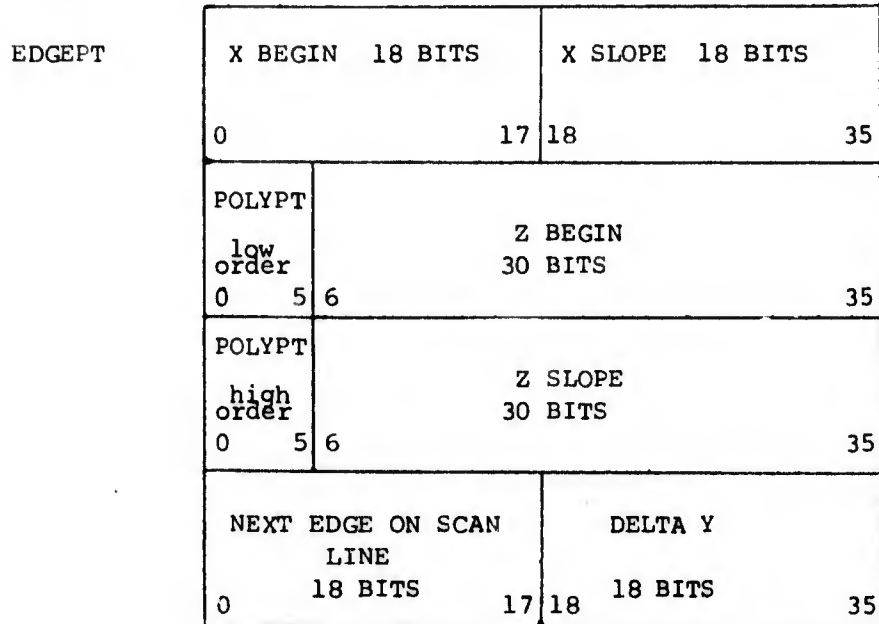
; TOPLEFT CORNER CALCULATION
P83:  CORNER 0,FSBR,FSBR,NM,NM
      CAIN  YZ,5
      JRST  P90
; BOTTOMLEFT CORNER CALCULATION
P84:  CORNER 1,FADR,FSBR,EM,EM
      JRST  P90
; TOPRIGHT CORNER CALCULATION
P85:  CORNER 0,FADR,FADR,EM,EM
      CAIN  YZ,6
      JRST  P90
; BOTTOMRIGHT CORNER CALCULATION
P88:  CORNER 1,FSBR,FADR,NM,EM
P90:  PERSP  VX+6,VX-1(YX),XR,VZ-1(YX)
      PERSP  VX+7,VX-1(YZ),XR,VZ-1(YZ)
      PERSP  VY+6,VY-1(YX),YR,VZ-1(YX)
      PERSP  VY+7,VY-1(YZ),YR,VZ-1(YZ)
      MOVE  0,PICTURE
      JUMPL 0,LINMOD          ;JUMP IF VECTOR MODE IS SET
      ZINVR5 VZ+6,VZ-1(YX)
      ZINVR5 VZ+7,VZ-1(YZ)
      MOVEI  YX,6
      MOVEI  YZ,7
      MOVE  VY+6
      CAML  VY+7
      EXCH  YX,YZ
P94:  MOVE  2,VY(YX)
      SUB  2,VY(YZ)
      JUMPE 2,HORZED          ;JUMP IF HORIZONTAL EDGE
      MOVEM 2,DELTAY#        ;DELTAY_VY(YX)-VY(YZ)
      SLDBEG VX(YZ),IXBEG#,IXSL#,VX(YX)
      SLDBEG VZ(YZ),IZBEG#,IZSL#,VZ(YX)
      MOVE  IPRIDRITY
      ASH  6
      ADDM  IZBEG  ;ADD PRIORITY TO Z BEGI^
      JRST  PACK
HORZED: MOVEI 2,2
      JSA  16,GETVAR
      JUMP 2          ;RETURN POINTER IN AC2
      JUMP 2          ;GET A BLOCK OF SIZE 2
      MOVE 0,POLYPT    ;GET POLYGON POINTER
      HRLM 0,FREE-1(2) ;STORE IT
      MOVE 0,VX(YX)    ;GET X
      MOVE 1,VX(YZ)    ;GET X
      CAML 0,1        ;INSURE XLEFT IS IN AC0 AND
      EXCH 0,1        ;XRIGHT IS IN AC1
      ASH  0,-11      ;DROP FRACTIONAL BITS
      ASH  1,-11      ;DITTO
      HRLM 0,FREE(2)   ;STORE XL
      HRRM 1,FREE(2)   ;STORE XR
      MOVE 1,VY(YX)    ;GET SCAN LINE
      JUMPE 1,JUMPER   ;DON'T DO IT IF SCAN LINE 0
      HRRM 0,HBUCKY-1(1) ;GET NEXT POINTER
      HRRM 2,HBUCKY-1(1) ;CURRENT LINE
      HRLM 2,HBUCKY(1) ;PREVIOUS LINE
      HRRM 0,FREE-1(2) ;STORE NEXT POINTER
      JRST JUMPER

```

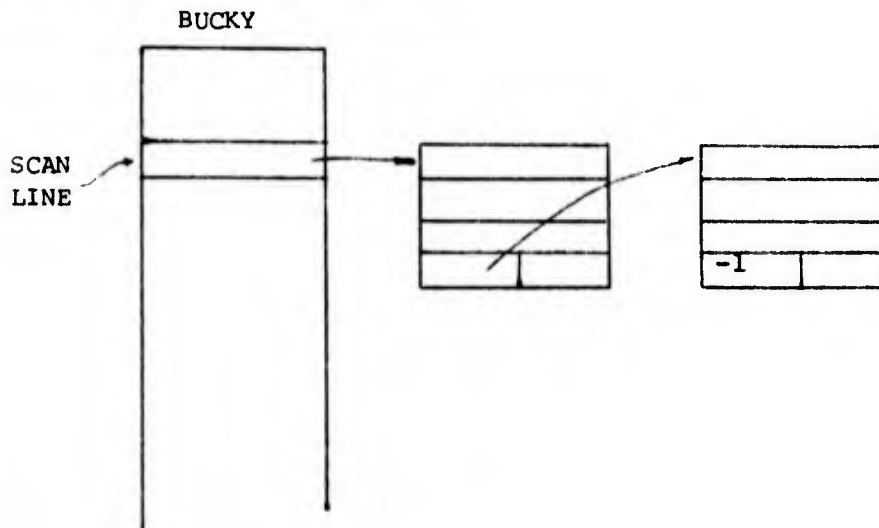
^L

PACK

PACK obtains a four word block from free storage. It then takes all the describing information of this edge and packs it into the four words according to the drawing below.



The bucket list is updated so that the edge is entered at the appropriate scan line location.



```

PACK:  MOVEI 2,4
      JSA 16,GETVAR
      JUMP 3
      JUMP 2
      MOVE IXBEG
      HRL IXSL ;XSLOPE GOES IN TOPHALF
      MOVEM FREE-1(3)
      MOVE 1,IZSL
      LSH 1,6 ;PUSH TO TOP OF WORD
      MOVE POLYPT
      LSHC -6 ;PUT LOW ORDER POLYPT IN TOP OF AC1
      MOVEM 1,FREE(3)
      MOVE 1,IZBEG
      LSH 1,6 ;PUSH TO TOP OF WORD
      LSHC -6 ;PUT HIGH ORDER POLYPT IN TOP OF AC1
      MOVEM 1,FREE+1(3)
      MOVE DELTAY
      MOVE 4,VY(YZ) ;GET Y
      HRL 0,BUCKY-1(4) ;GET NEXT EDGE POINTER
      MOVEM 3,BUCKY-1(4)
      MOVEM 0,FREE+2(3)
JUMPER: JRST 0
P23:  MOVEM FLAGS,FLGLDC ;SAVE FLAGS
      MOVE 7,AC7
      SEIZM VZ+6
LINMOD: JRA 16,0(16)
      MOVE 2,VX+6
      ASL 2,11
      JSA 16,DRAM
      JUMP 2
      JUMP VY+6
      MOVE 2,VX+7
      ASL 2,11
      JSA 16,DRAW
      JUMP 2
      JUMP VY+7
      JRST JUMPER

```

^L

CROSS

CROSS has the responsibility to see if two edges that are visible on a scan line cross on the next scan line. This subroutine will then determine the X point of intersection and draw the edge which is no longer visible. If everything seems to be in order, a flag is set in the segment block stating that the edge is visible on this scan line.

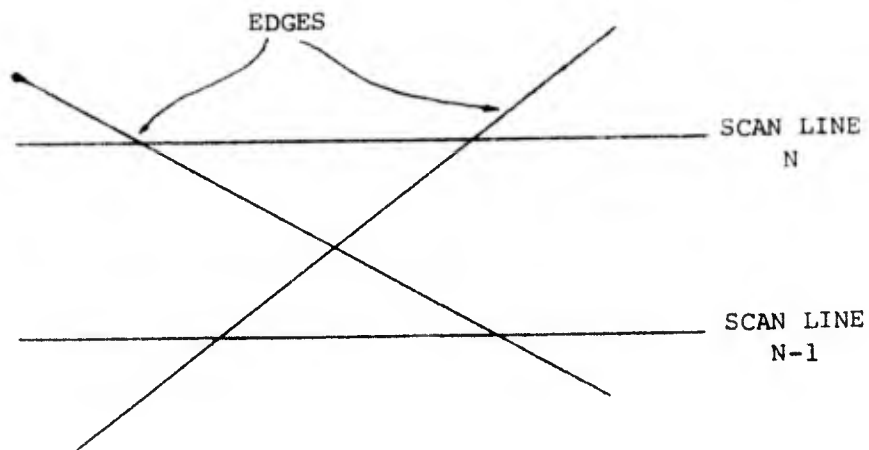
```

EXTERNAL FREE, MOVE TO, VECTO, INTENS, CLPPER
INTERNAL DRAW, DRAWH, DRAWV, SETLIN, CROSS
EXTERNAL OLDEEG
JLDX1=JLDEEG
JLDX2=JLDEEG+1
OLDSEG=JLDEEG+2
CRSS:  Z      ;CHECK TO SEE IF EDGES CROSS
      MOVE    2, @3(16)      ;GET IY
      ADJ     2,
      TRZ     2, 777770      ;ONLY NEED LOW BITS
      MOVE    1, @2(16)
      ADD     1, CLPPER+12
      MOVE    FREE-1(1)      ;GET THE SETLIN
      TRZ     7
      IOR     2
      MOVEM   FREE-1(1)      ;MASK IY+1 INTO WORD
      MOVE    1, @1(16)
      MOVE    2, 1
      ADD     1, CLPPER+12
      MOVE    3, FREE-1(1)    ;GET CURRENT X
      MOVEM   3, VX+6
      CAML    3, JLDX1 ;IS JLDX1 IN FRONT?
      JRST   CRSXIT
      CAIE   2, 3
      JRST   CRSGT ;MUST BE RIGHT EDGE
      HLRE   FREE-2(1)      ;GET YLEFT
      SKIPA

```

^L

When the program is at this point, everything is guaranteed that the two edges do, in fact, cross. The Z value for both of the edges must be calculated for the two edges at the point of crossing to determine which edge will be drawn. The edge with the smallest Z value will be drawn.



```

CRSROT: HRRE      FREE-4(1)      ;GET YRIGHT
        JUMPE     CRSXIT ;JUMP IF EDGE EXITED
        MOVE      5, 10(16)     ;GET PREVIOUS X
        MOVE      6, 5
        SUB       6, OLDX2
        SUB       5, 3
        SUB       5, JLDX2
        ADD       5, JLDX1
        DIV       6, 5          ;AC6_(NEWX2-NEWX1)/(OLDX1-JLDX2-NEWX1+NEWX2)
        ASH      6, -21        ;GET DELTA
        MOVE      OLDX1
        SUB       OLDX2
        MUL       0, 6
        ASHC     0, 21
        ADD      OLDX2 ;GET X POINT AT CROSSING
        HLRZ     1, JLDSEG
        ADD      1, JLDSEG
        MOVE     2, FREE+4(1)    ;GET ZSLOPE FOR OLDEGE
        MUL     2, 6
        ASHC   2, 21
        MOVE   3, FREE+3(1)     ;GET Z FOR OLDEGE
        SUB   3, 2 ;GET Z AT CROSSING FOR OLDEGE
        MOVE  1, 11(16)
        ADD  1, CLPPER+12
        MOVE  4, FREE+4(1)     ;GET ZSLOPE FOR CURRENT EDGE
        MUL  4, 6
        ASHC 4, 21
        MOVE 2, FREE+3(1)     ;GET Z FOR CURRENT EDGE
        SUB 2, 4 ;GET Z AT CROSSING FOR CURRENT EDGE
        CAMG 2, 3
        JRST CRSCUR ;DRAW CURRENT EDGE
        HLRZ 7, JLDSEG
        CAIE 7, 3
        SOJ  7,
        ADD  7, JLDSEG ;GET ADDRESS FOR OLDEGE
        JRST DRNCRS-2
CRSCUR: MOVE 7, 11(16)
        CAIE 7, 3
        SOJ  7,
        ADD  7, CLPPER+12 ;GET ADDRESS FOR CURRENT EDGE
        HRRZ 4, FREE+7(7)
        JUMPE 4, CRSXIT-1 ;JUMP IF ALREADY DRAWN

```

^L

DRAW

DRWCRS gets the X and Y point of intersection for drawing purposes. A YOFFSET value must be calculated since the intersection occurred between scan lines. The end point of the edge is extracted from the segment block and the line is completely drawn.

Subroutine DRAW has the responsibility to draw the line for an edge which has exited or was completely visible on the previous scan line. Note that half of the slope is added so that the line will be defined as existing between scan lines.

```

DRACRS: MOVEM 7,VY+6
        MOVE 4,@3(16)      ;GET Y
        HRRZ 1,DRAW3
        MOVSI 5,1
        SUB  2,6
        IMUL 1,5
        HRRZ 2,DRAW4
        HRRM 2,DRAW5
DRAWB:  ASH  1,0
        ADDI 1,400000      ;ADD 1/2 AND ROUND
        HLRZM 1,VZ+6 ;STORE Y OFFSET
        JSA  16,DRAWM
        JUMP 0
        JUMP 4
        MOVE 7,VY+6
        HLRZ 3,FREE+7(7)
        ASH  3,11
        HRRZ 4,FREE+7(7)
        ASH  4,-3
        SEIZM VZ+6
        SEIZM FREE+7(7)
        JSA  16,DRAWV
        JUMP 3
        JUMP 4
        MOVE 3,VX+6
DRSXIT: MOVEM 3,OLDX1
        MOVE @0(16)
        MOVEM OLDX2
        MOVE  CLPPER+12
        HRL  @1(16)
        MOVEM OLDSEG
        JRA  16,4(16)
DRAW:   Z ;DRAW A LINE FROM HIDDEN LINE BOX
        SEIZM VZ+6
        MOVE 1,@2(16)
        ADD  1,CLPPER+12
        HLRZ 3,FREE-1(1)      ;GET X
        ASH  3,11
        HRRZ 4,FREE-1(1)      ;GET Y
        JUMPE 4,DRAXIT ;DON'T DRAW IF ALREADY DRAWN
        ASH  4,-3
        SEIZM FREE-1(1)      ;CLEAR LINE DATA
        JSA  16,DRAWM
        JUMP 3
        JUMP 4
        MOVE 1,@1(16)
        ADD  1,CLPPER+12
        MOVE 3,FREE-1(1)      ;GET XSLOPE
        SKIPGE 3
        ADJ  3,
        ASH  3,-1
        ADD  3,@0(16)      ;ADD HALF THE SLOPE
        MOVE 4,@3(16)      ;GET Y
        JSA  16,DRAWV
        JUMP 3
        JUMP 4
DRAXIT: JRA  16,4(16)
^L

```

DRAWM - DRAWV - SETLIN

DRAWM and DRAWV expect the X value to be of the form $(N,,M)$, where N is the integer part and M is the fractional part. This results in no loss in precision while scaling and drawing the line. The Y value is right justified in the word. DRAW4 makes certain that the line will not go outside of the specified window viewing area.

SETLIN is called when the hidden surface algorithm determines that the edge of a segment is visible for the first time. This subroutine will store the begin point of the edge for future reference when the line will be actually drawn. Note that half of the slope is subtracted so that the line actually begins between scan lines.

```

DRAWM: Z           ;SCALE DATA AND POSITION BEAM
        MOVEI     1,MOVEIO
        JRST     DRAWI
DRAWV: Z           ;SCALE DATA AND PAINT LINE
        MOVEI     1,VECTO
DRAW1: MOVE       3,0(16)           ;GET X
DRAW2: IMULI     3,0           ;STORED BY DRARES
        ASH      3,-24
        MOVE     4,0(16)           ;GET Y
        ADJ      4,
DRAW3: IMULI     4,0           ;STORED BY INTCLP
DRAW4: ASH      4,0           ;STORED BY INTCLP
        ADD      4,VZ+6
        CAML     3,INTENS+3
        MOVE     3,INTENS+3           ;RESET X
        CAML     4,INTENS+4
        MOVE     4,INTENS+4           ;RESET Y
        JSA     16,(1)           ;CALL MOVEIO OR VECTO
        JUMP     3
        JUMP     4
        JRA     16,2(16)
DRARES: Z           ;DETERMINE SCALING FACTORS
SETLIN: Z           ;STORE BEGIN LOCATION OF LINE
        MOVE     2,0(16)
        ADD      2,CLPPER+12
        MOVN     3,FREE-1(2)
        SKIPGE   3
        ADJ      3,
        ASH      3,-1
        ADD      3,0(16)
        SKIPGD   3           ;SKIP IF NOT LESS THAN ZERO
        SEIZ     3,
        TRZ     3,400
        LSH     3,6           ;MAKE ROOM FOR Y
        IOR     3,03(16)
        ADJ      3,
        LSH     3,3
        MOVE     1,0(16)
        ADD      1,CLPPER+12
        MOVEM    3,FREE-1(1)
        JRA     16,4(16)
        END

```

OUTSTR - LDLPT - LDRPT -
STLPT - STRPT - LOD2HF -
STR2HF - MYMAX0 - MYMIN0

OUTSTR will output a string of text on the teletype using a TTCALL. This was implemented to avoid difficulty between FORTRAN I/O and SAIL I/O.

LDLPT, LDRPT, STLPT, and STRPT all perform half word transfers. Note that the loading routines do a sign extend.

LOD2HF and STR2HF expect the arguments to be located at some address inside the array free. Note that LOD2HF expects the arguments to be at the address plus two. The reason is that the majority of the loads come from this address.

MYMAX0 and MYMIN0 are similar to the FORTRAN functions, except that these functions work on two arguments.

TITLE UTILITY ROUTINES FOR HIDDEN SURFACE PROCESSING
 INTERNAL OUTSTR, LDLPT, LDRPT, LOD2HF, MYMAXO, MYMINO, SILPT, STRPT, STR2HF
 INTERNAL UNPACK, XINTER, ZVALUE
 EXTERNAL FREE, CLPPER

```

OUTSTR: Z      ;TYPE OUT A MESSAGE
          TTCALL 3, 20(16)
          JRA   16, 1(16)
LDLPT:  0      ;LOAD LEFT HALF OF ARG2 INTO ARG1
          HLRE  0, 21(16)
          MOVEM 0, 20(16)
          JRA   16, 2(16)
LDRPT:  0      ;LOAD RIGHT HALF OF ARG2 INTO ARG1
          HLRE  0, 21(16)
          MOVEM 0, 20(16)
          JRA   16, 2(16)
LOD2HF: Z      ;GET LEFT AND RIGHT HALF OF 3RD LOC OF FREE BLOCK
          MOVE  1, 20(16)
          HLRE  0, FREE+1(1)
          MOVEM 0, 21(16)
          HLRE  0, FREE+1(1)
          MOVEM 0, 22(16)
          JRA   16, 3(16)
MYMAXO: Z      ;MAX VALUE OF 2 ARGS
          MOVE  2(16) ;GET 1ST ARG
          CAMGE 21(16) ;SKIP IF .GE. TO 2ND ARG
          MOVE  21(16) ;2ND ARG WINS
          JRA   16, 2(16)
MYMINO: Z      ;MIN VALUE OF 2 ARGS
          MOVE  2(16) ;GET 1ST ARG
          CAMLE 21(16) ;SKIP IF .LE. TO 2ND ARG
          MOVE  21(16) ;2ND ARG WINS
          JRA   16, 2(16)
SILPT:  0      ;STORE ARG1 INTO LEFT HALF OF ARG2
          MOVE  0, 20(16)
          HRLM  0, 21(16)
          JRA   16, 2(16)
STRPT:  0      ;STORE ARG1 INTO RIGHT HALF OF ARG2
          MOVE  0, 20(16)
          HRRM  0, 21(16)
          JRA   16, 2(16)
STR2HF: Z      ;STORE ARG2 AND ARG3 INTO LEFT AND RIGHT LOCS OF FREE
          ** -1(ARG1)
          MOVE  1, 20(16)
          MOVE  2, 21(16)
          HRLM  2, FREE-1(1)
          MOVE  2, 22(16)
          HRRM  2, FREE-1(1)
          JRA   16, 3(16)
  
```

^L

UNPACK

UNPACK extracts the data from the four word edge block created by EDGMAK and places the answers in specific locations in common area EDGARG.

```

:      SUBROUTINE UNPACK(EDGEPT,POLYPT,DELY,
:      X3EG,XSLOPE,ZBEG,ZSLOPE)
EXTERNAL EDGARG
EDGEPT=EDGARG ;POINTER TO EDGE BLOCK
POLYPT=EDGARG+1 ;POINTER TO POLYGON
DELY=EDGARG+2 ;DELTA Y
XSLOPE=EDGARG+3 ;X SLOPE OF EDGE
ZSLOPE=EDGARG+4 ;Z SLOPE OF EDGE
UNPACK: Z
      MOVE      2,EDGEPT          ;EDGEPT
      MOVE      0,FREE+2(2)
      HLREX     0,EDGEPT          ;NEXTEDG (18 BITS)
      SETZ     0,
      MOVE      1,FREE+1(2)
      LSHC     6
      MOVE      1,FREE(2)
      LSHC     6
      HRLZM    0,POLYPT          ;POLYPT (12 BITS)
      HRRE     0,FREE+2(2)
      MOVEM    0,DELY ;DELY
      HRLZ     FREE-1(2)
      LSH      -11 ;MAKE 10 BIT FRACTIONAL
      MOVEM    0,XLEFT ;XBEG (20 BITS)
      HLLZ     FREE-1(2)
      ASH      -11 ;MAKE 10 BIT FRACTIONAL
      MOVEM    0,XSLOPE          ;XSLOPE (20 BITS)
      MOVE     FREE+1(2)
      TLZ      770000
      ASH      10
      MOVEM    0,ZLEFT ;ZBEG (30 BITS)
      MOVE     FREE(2)
      TLZ      770000
      TLNE     4000
      TLJ      770000
      ASH      10
      MOVEM    0,ZSLOPE          ;ZSLOPE (30 BITS)
      JRA      16,(16)

```

^L

XINTER

When the hidden surface algorithm has determined that two potentially visible segments are intersecting, XINTER will evaluate the X value. A subroutine was used for this purpose, since the arithmetic resulted in numbers containing 50 bits and thus created the need for using fractional operators.

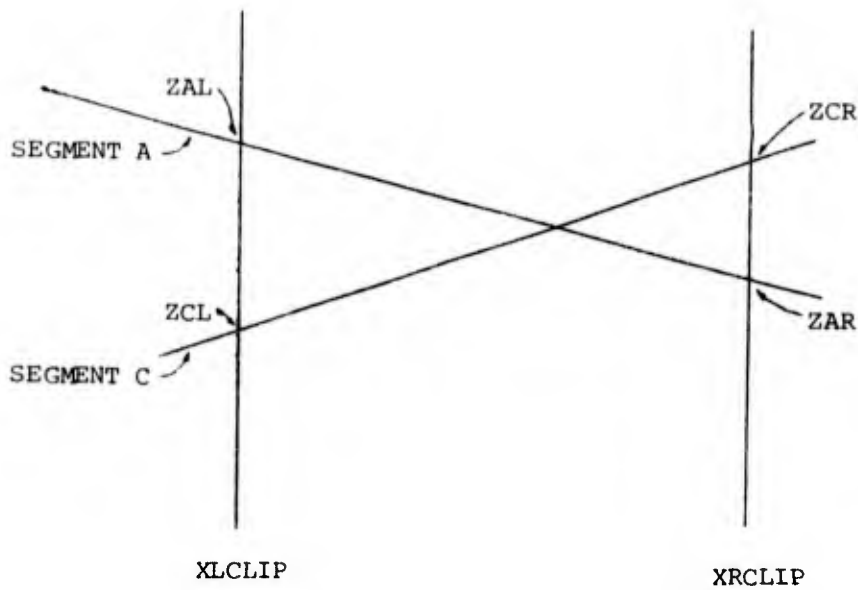
```

ZAL=CLPPER      ;Z LEFT OF LINE A
ZAR=CLPPER+1    ;Z RIGHT OF LINE A
ZCL=CLPPER+2    ;Z LEFT OF LINE C
ZCR=CLPPER+3    ;Z RIGHT OF LINE C
XLCLIP=CLPPER+4 ;X LEFT OF POINT BEING CLIPPED TO
XRCLIP=CLPPER+5 ;X VALUE OF NEW LINE A
XLEFT=CLPPER+6  ;X VALUE OF NEW LINE A
ZRIGHT=CLPPER+7 ;Z VALUE OF NEW LINE A
ZLEFT=CLPPER+10 ;Z VALUE OF NEW LINE A
XRCLIP=CLPPER+11 ;X RIGHT OF POINT BEING CLIPPED TO
ZS=CLPPER+12    ;PREVIOUS LINE C (SEGPT,XL,XR,ZL,ZR)
XAMXL=CLPPER+17 ;POINT OF INTERSECTION OF LINE A AND C
;      ***      Z NUMBERS ARE 30 BITS, X NUMBERS ARE 20 BITS      ***
XINTER: Z      ;THIS ROUTINE COMPUTES THE INTERSECTION OF A AND C
        MOVE   2,ZCL
        SUB    2,ZAL
        SUB    2,ZCR
        AD    2,ZAR
        MOVE   0,ZAR
        SUB    0,ZCR
        MUL    0,XLCLIP
        DIV    0,2
        MOVE   4,ZAL
        SUB    4,ZCL
        MUL    4,XRCLIP
        DIV    4,2
        SUB    0,4
        CAMGE  XLCLIP  ;IS XINTERSECTION LESS THAN XLCLIP?
        MOVE   XLCLIP  ;YES
        CAMLE  XRCLIP  ;IS XINTERSECTION GREATER THAN XRCLIP?
        MOVE   XRCLIP
        MOVEM  0,XAMXL
; XAMXL=(XLCLIP*(ZAR-ZCR)-XRCLIP*(ZAL-ZCL))/(ZCL-ZAL-ZCR+ZAR)
        JRA   16,(16)
^L

```

ZVALUE

Subroutine ZVALUE performs the arithmetic formally described as the clipper in the Watkins' Algorithm. Given a X left clip point and a X right clip point, this routine evaluates the Z values for two segments. This information is passed back to the hidden surface algorithm for comparison.



```

ZVALUE: Z      ;THIS ROUTINE COMPUTES ZAL,ZAR,ZCL,ZCR
            MOVE 0,ZLEFT
            SUB  0,ZRIGHT
            MOVE 2,XLCLIP
            SUB  2,XRIGHT
            MOVE 4,XLEFT
            SUB  4,XRIGHT
            MUL  2,0
            DIV  2,4
            ADD  2,ZRIGHT
            MOVEM 2,ZAL
; ZAL=((XLCLIP-XRIGHT)*(ZLEFT-ZRIGHT))/(XLEFT-XRIGHT)+ZRIGHT
            MOVE 2,XRCLIP
            SUB  2,XRIGHT
            MUL  2,0
            DIV  2,4
            ADD  2,ZRIGHT
            MOVEM 2,ZAR
; ZAR=((XRCLIP-XRIGHT)*(ZLEFT-ZRIGHT))/(XLEFT-XRIGHT)+ZRIGHT
            MOVE 0,ZS+3
            SUB  0,ZS+4
            MOVE 2,XLCLIP
            SUB  2,ZS+2
            MOVE 4,ZS+1
            SUB  4,ZS+2
            MUL  2,0
            DIV  2,4
            ADD  2,ZS+4
            MOVEM 2,ZCL
; ZCL=((XLCLIP-ZS(3))*(ZS(4)-ZS(5)))/(ZS(2)-ZS(3))+ZS(5)
            MOVE 2,XRCLIP
            SUB  2,ZS+2
            MUL  2,0
            DIV  2,4
            ADD  2,ZS+4
            MOVEM 2,ZCR
; ZCR=(XRCLIP-ZS(3))*(ZS(4)-ZS(5))/(ZS(2)-ZS(3))+ZS(5)
            JRA  16,(16)
            EN)

```

INFREE - GETVAR - LSTSET -

GETBLK - RETBLK

INFREE initializes the FREE array. The first argument is the number of locations which must be zeroed - four is the minimum. The second argument is the length of the free array. This value should be the same as the dimension of FREE in the user's program.

GETVAR gets a variable sized block from the array FREE. The first argument is returned to the user and is the index into the FREE array. The second argument is supplied by the user and is the length of the block that is desired.

LSTSET is used by the hidden surface algorithm. The remaining area inside FREE is broken into blocks of size N. These blocks are used as segment storage areas and are linked together.

GETBLK returns to the hidden surface algorithm a block of size N for storing of segment data.

RETBLK returns to free storage a block of segment data. This block is then available for future use.

```

SUBROUTINE INFREE(I,LENGTH)
COMMON/CORE/FREEST,LEN,FREEPT/FREE/FREE(I)
IMPLICIT INTEGER (A-Z)
IF(I.LE.0)GO TO 2
DO 1 K=1,I
FREE(K)=0
LEN=LENGTH
FREEPT=1+I
RETURN
END

```

```

SUBROUTINE GETVAR(INDEX,LENGTH)
COMMON/CORE/FREEST,LEN,FREEPT
IMPLICIT INTEGER (A-Z)
INDEX=FREEPT
FREEPT=FREEPT+LENGTH
IF(FREEPT.LT.LEN)RETURN
CALL OUTSTR('MAX STORAGE')
CALL EXIT
END

```

```

SUBROUTINE LSTSET(N)
COMMON/CORE/FREEST,LEN,FREEPT/FREE/FREE(I)
IMPLICIT INTEGER (A-Z)
FREEPT=0
K=LEN-N+1
IF(K.LT.FREEST)RETURN
FREEPT=FREEPT
DO 1 I=FREEPT,K,N
M=I
FREE(I)=I+N
FREE(M)=0
RETURN
END

```

```

SUBROUTINE GETBLK(INDEX)
COMMON/CORE/FREEST,LEN,FREEPT/FREE/FREE(I)
IMPLICIT INTEGER (A-Z)
IF(FREEPT.EQ.0)GO TO 1
INDEX=FREEPT
FREEPT=FREE(FREEPT)
RETURN
CALL OUTSTR('MAX STORAGE')
CALL EXIT
END

```

```

SUBROUTINE RETBLK(INDEX)
COMMON/CORE/FREEST,LEN,FREEPT/FREE/FREE(I)
IMPLICIT INTEGER (A-Z)
FREE(INDEX)=FREEPT
FREEPT=INDEX
RETURN
END

```

HIDDEN

Following is a description of the code used in solving the hidden surface problem. No attempt is going to be made to explain the reasons or theory behind the algorithm. If the reader is really curious, it is suggested he look into the references at the end of this report.

By the time HIDDEN is called, all the edges should have been clipped and stored into the FREE array. In addition to this, they should have been sorted in Y and entered into the bucket array. The initialization procedure consists of setting IY=FRAMEY and setting up the size of the segment blocks by calling LSTSET. At the beginning of each scan line, the program jumps to label 204 to get the first edge entering on the scan line. Label 210 is the location where the program keeps looping to pick up all edges entering on the current scan line.

```
SUBROUTINE HIDDEN
COMMON/SCOPE/BUCKY(0/255),HBUCKY(256),BB
COMMON/CLPPER/ZAL,ZAR,ZCL,ZCR,XLCLIP,XRIGHT,XLEFT,ZRIGHT
I,ZLEFT,XRCLIP,ZS(5),XAMXR
COMMON/EDGARG/EDGEPT,POLYPT,DELY,XSLOPE,ZSLOPE
COMMON/FREE/EDGE(1)
COMMON/EYES/OI(6),FRAMEX,FRAMEY
COMMON/JLDEDG/JLDX1,JLDX2,OLDSEG
IMPLICIT INTEGER (A-Z)
DIMENSION SEG(1),SAM(4)
EQUIVALENCE (EDGE,SEG)
C      INITIALIZATION.
      IY=FRAMEY
      CALL LISTSET(13)
      SEGXST=0
204     EDGEPT=BUCKY(IY)
210     IF(EDGEPT) GO TO 242
      CALL UNPACK
C      GET POINTER TO FIRST OF SEG LIST
      SEGPT=SEGXST
      PREV=0
      SPLIT=.FALSE.
^L
```

A block is retrieved from free storage and the edge information is stored into the segment as the left edge. The segment list is searched to see where the new segment block should be entered with respect to X. At times, a current segment block must be split into two segments, since the entering segment must be placed between the two edges.

Label 242 is where the scan line count is decremented and the scan processing begins. The XLEFT sample point (SAM(2)) is set to zero and the XRIGHT sample point (SAM(1)) is set to 1024.

```

) STORE EDGE DATA IN SEG BLOCK
  CALL GETBLK(I)
  SEG(I+2)=DELY*262144
  SEG(I+1)=POLYPT
  SEG(I+3)=XLEFT
  SEG(I+4)=XSLOPE
  SEG(I+8)=ZSLOPE
  SEG(I+7)=ZLEFT
  SEG(I+11)=0
C GO SEARCH & SORT LIST TO SEE WHERE NEW EDGE WILL GO
214  IF(SEGPT.EQ.0)GO TO 226
     CALL LJD2HF(SEGPT,YEND1,YEND2)
     TE1=XLEFT-SEG(SEGPT+3)
     TE2=XLEFT-SEG(SEGPT+5)
     IF(POLYPT.NE.(SEG(SEGPT+1).AND..NOT.262143))GO TO 220
     IF(YEND1.GE.0)GO TO 221
C SEE IF EXISTING SEGMENT MUST BE SPLIT FOR NEW EDGE
     IF(TE1.LT.0)GO TO 226
     IF(YEND2.GE.0.OR.TE2.GE.0.OR.SPLIT) GO TO 219
C LOAD RIGHT EDGE OF SEGMENT INTO NEW BLOCK
     SPLIT=.TRUE.
     SEG(I+5)=SEG(SEGPT+5)
     SEG(I+6)=SEG(SEGPT+6)
     SEG(I+9)=SEG(SEGPT+9)
     SEG(I+10)=SEG(SEGPT+10)
     SEG(I+12)=SEG(SEGPT+12)
     CALL STRPT(YEND2,SEG(I+2))
     CALL STRPT(0,SEG(SEGPT+2))
219  PREV=SEGPT
C GET POINTER TO NEXT SEGMENT BLOCK
     CALL LORPT(SEGPT,SEG(SEGPT))
     GO TO 214
220  IF(YEND1.GE.0)GO TO 221
     IF(TE1)226,219,219
221  IF(YEND2.GE.0)GO TO 219
     IF(TE2)226,219,219
226  SEG(I)=SEGPT
C INSERT THIS NEW SEGMENT BLOCK BETWEEN EXISTIN SEGMENTS
     IF(PREV.NE.0)SEG(PREV)=I
     IF(PREV.EQ.0)SEGXST=I
     GO TO 210
C DECREMENT SCAN LINE COUNT
242  IY=IY-1
     INTSCT=.FALSE.
     JLDX1=0
     SE3S2=0
     SE3L2=0
     SAM(2)=0
     SEGACT=0
C GET NEXT LEFT SAMPLE POINT
281  SAM(1)=SAM(2)+1024
     ZS(1)=0
     FROM=0
     SEGPT=SEGACT
     SEGACT=0
     EXTEND=.TRUE.

```

Due to edges entering and exiting, the X sort segment list must be updated so that all edges are linked together as segments belonging to polygons. This is one of the critical areas in the algorithm, since a great deal of time could be consumed in re-sorting the list. However, it turns out that an extremely small percent of the time is spent here, since a picture changes very little between scan lines.

```

301  IF (SEGPT.EQ.0) GO TO 304
      NEXT=SEG(SEGPT+1).AND..262143
      XLEFT=SEG(SEGPT+3)-SEG(SEGPT+4)
      XRIGHT=SEG(SEGPT+5)-SEG(SEGPT+6)
      ZLEFT=SEG(SEGPT+7)-SEG(SEGPT+8)
      ZRIGHT=SEG(SEGPT+9)-SEG(SEGPT+10)
      IF(XRIGHT-SAM(1)-345,345,315
304  SEGPT=SEGXST
      IF(SEGPT.EQ.0)GO TO 350
      RETURN TO FREE LIST IF BLOCK IS EMPTY
      IF(SEG(SEGPT+2).NE.0)GO TO 235
      CALL LDRPT(SEGXST,SEG(SEGPT))
      CALL RETBLK(SEGPT)
      GO TO 304
235  IF(SEG(SEGPT+2).LT.0)GO TO 236
      C MOVE RIGHT EDGE OF SEGMENT TO LEFT EDGE OF SEGMENT
      CALL STR2HF(SEGPT+2,SEG(SEGPT+2),0)
      SEG(SEGPT+3)=SEG(SEGPT+5)
      SEG(SEGPT+4)=SEG(SEGPT+6)
      SEG(SEGPT+7)=SEG(SEGPT+9)
      SEG(SEGPT+8)=SEG(SEGPT+10)
      SEG(SEGPT+11)=SEG(SEGPT+12)
236  CALL LDRPT(YEND2,SEG(SEGPT+2))
      IF(YEND2.LT.0)GO TO 305
      POLYPT=SEG(SEGPT+1).AND..NOT.262143
      CALL LDRPT(NEXT,SEG(SEGPT))
237  IF(NEXT.EQ.0)GO TO 241
      IF((SEG(NEXT+1).AND..NOT.262143).NE.POLYPT)GO TO 239
      CALL LDLPT(YEND1,SEG(NEXT+2))
      IF(YEND1.GE.0) GO TO 238
      C MOVE LEFT EDGE OF SEGMENT TO RIGHT EDGE OF SEGMENT
      CALL STRPT(YEND1,SEG(SEGPT+2))
      CALL STLPT(0,SEG(NEXT+2))
      SEG(SEGPT+5)=SEG(NEXT+3)
      SEG(SEGPT+6)=SEG(NEXT+4)
      SEG(SEGPT+9)=SEG(NEXT+7)
      SEG(SEGPT+10)=SEG(NEXT+8)
      SEG(SEGPT+12)=SEG(NEXT+11)
      GO TO 305
238  CALL LDRPT(YEND2,SEG(NEXT+2))
      IF(YEND2.GE.0) GO TO 239
      CALL STRPT(YEND2,SEG(SEGPT+2))
      SEG(NEXT+2)=0
      SEG(SEGPT+5)=SEG(NEXT+5)
      SEG(SEGPT+6)=SEG(NEXT+6)
      SEG(SEGPT+9)=SEG(NEXT+9)
      SEG(SEGPT+10)=SEG(NEXT+10)
      SEG(SEGPT+12)=SEG(NEXT+12)
      GO TO 305
239  CALL LDRPT(NEXT,SEG(NEXT))
      GO TO 237
241  SEG(SEGPT+5)=SEG(SEGPT+3)
      SEG(SEGPT+6)=SEG(SEGPT+4)
      CALL STRPT(-1,SEG(SEGPT+2))
^L

```

This page of code performs the updating of the segments. Since each segment contains the slope information for both the left and right edge, this data is added to the current values of X and Z, giving the new X and Z for the next scan line.

```

300  XLEFT=SEG(SEGPT+3)
      XRIGHT=SEG(SEGPT+5)
      IF((.NOT.FXEND.OR.ZS(1).NE.0).AND.XLEFT.GE.SAM(2))GO TO 347
      FRJM=-1
      CALL LDRPT(SEGXST,SEG(SEGPT))
      ZLEFT=SEG(SEGPT+7)
      ZRIGHT=SEG(SEGPT+9)
C  UPDATE XLEFT,XRIGHT,ZLEFT,ZRIGHT
      SE3(SEGPT+3)=SEG(SEGPT+3)+SEG(SEGPT+4)
      SE3(SEGPT+5)=SEG(SEGPT+5)+SEG(SEGPT+6)
      SE3(SEGPT+7)=SEG(SEGPT+7)+SEG(SEGPT+8)
      SE3(SEGPT+9)=SEG(SEGPT+9)+SEG(SEGPT+10)
      INT=0
C  UPDATE YLEFT AND YRIGHT SCAN LINE COUNT
      CALL LOD2HF(SEGPT,YEND1,YEND2)
      YEND1=YEND1+1
      YEND2=YEND2+1
      CALL STR2HF(SEGPT+2,YEND1,YEND2)
C
3091  BACK POINTERS NEEDED ON NEW LIST.
      IX=SEG(SEGPT+3)
      IF(YEND1.GE.0)IX=SEG(SEGPT+5)
      S2=0
      S1=SEGL2
3092  IF(S1.EQ.0)GO TO 3094
      IX1=SEG(S1+3)
      IF(SEG(S1+2).GE.0)IX1=SEG(S1+5)
      IF(IX.GE.IX1)GO TO 3094
      S2=S1
      CALL LODLPT(S1,SEG(S1))
      GO TO 3092
3094  IF(S2.NE.0)SEG(SEGPT)=S2
      CALL STLPT(S1,SEG(SEGPT))
      IF(S2.NE.0)CALL STLPT(SEGPT,SEG(S2))
      IF(S2.EQ.0)SEGL2=SEGPT
      IF(S1.NE.0)CALL STRPT(SEGPT,SEG(S1))
      IF(S1.EQ.0)SEGS2=SEGPT
312  IF(XRIGHT.LE.SAM(1))GO TO 345
^L

```

When the program has reached this point, it is ready to perform the depth processing on two segments to see which is visible for a span. The Z values for the two segments are evaluated at the X left clip point and the X right clip point. The program calls ZVALUE to obtain this information. If an intersection is determined, then XINTER is called to evaluate the point of crossing.

```

315    CONTINUE
C      ADDITION TIME ONE.
      INTSCF=.FALSE.
      ABLLI=SAM(1).GE.XLEFT
      ABRLI=XRIGHT.LT.SAM(2)
C GET X LEFT CLIP POINT
      XLCLIP=SAM(1)
      IF(.NOT.ABLLI)XLCLIP=XLEFT
C GET X RIGHT CLIP POINT
      XRCLIP=SAM(2)
      IF(ABRLI)XRCLIP=XRIGHT
      JOBOX=.FALSE.
      JBOXES=.TRUE.
      IF((ZS(1).EQ.0).AND..NOT.ABLLI)GO TO 335
      JBOXES=.FALSE.
      IF((ZS(1).EQ.0).AND.ABLLI)GO TO 3311
323    CONTINUE
C GET Z VALUES FOR NEW AND OLD LINES AT CLIP POINTS
      CALL ZVALUE
      ABACKL=ZCL.GE.ZAL
      ABACKR=ZCR.GE.ZAR
      JOBOX=ABACKL.AND.ABACKR
C JUMP IF AB BACK ON LEFT AND RIGHT
      IF(JOBOX)GO TO 335
      JIBOX=ABLLI.AND..NOT.ABACKL.AND..NOT.ABACKR
C JUMP IF AB NOT BACK ON LEFT AND RIGHT
      IF(JIBOX)GO TO 3311
      JINTER=(ABACKL.AND..NOT.ABACKR).OR.(.NOT.ABACKL.AND.ABACKR.AN
**).
      IABLLI)
C JUMP IF THE TWO SURFACES INTERSECTED
      IF(JINTER)GO TO 326
      JBOXES=.TRUE.
C      JBOXES=.NOT.ABLLI.AND..NOT.ABACKL BY DEFAULT
      GO TO 335
C GET THE X INTERSECTION POINT
326    CALL XINTER
      INTSCF=.TRUE.
330    SAM(2)=XAMXR
      SAM(3)=SAM(2)/256
      SAM(4)=0
      EXTEND=.FALSE.
      IF(ABACKR)GO TO 3312
      GO TO 335
^L

```

The visible span has been determined, and now the program must evaluate the next potential visible segment which should be compared. When the program has reached label 350, a visible segment is available for display. If an intersection is involved, then the procedure will display a dot.

```

: MAKE A ONE ELEMENT BOX.
3311 IF(ZS(1).NE.0.AND.ABRLT)EXTEND=.FALSE.
    IF (.NOT.ABRLT.AND..NOT.EXTEND) GO TO 3312
    SAM(2)=XRIGHT
    SAM(3)=XRIGHT/256
    CALL LDLPT(SAM(4),SEG(SEGPT+6))
3312 ZS(1)=SEGPT
    ZS(2)=XLEFT
    ZS(3)=XRIGHT
    ZS(4)=ZLEFT
    ZS(5)=ZRIGHT
335 CONTINUE
    IF(J)BOX.AND..NOT.(XRIGHT.LE.SAM(2))EXTEND=.FALSE.
    IF(J)BOX.AND.(XRIGHT.LE.SAM(2)) GO TO 345
    CALL STRPT(SEGACT,SEG(SEGPT+1))
    SEGACT=SEGPT
    IF (.NOT.JBOXES) GO TO 345
    EXTEND=.FALSE.
    SAM(2)=XLEFT
    SAM(3)=XLEFT/256
    CALL LDLPT(SAM(4),SEG(SEGPT+4))
345 SEGPT=NEXT
    IF(FROM.EQ.0)GO TO 301
    GO TO 304
347 IF((SAM(2).AND..NOT.262143).NE.(XLEFT.AND..NOT.262143))GO TO
**350 IF(SAM(4)*262144.NE.(SEG(SEGPT+4).AND..NOT.262143)) SAM(4)=0

C OUTPUT SEGMENTS.

350 CONTINUE
    IF(EXTEND)SEGACT=0
    ENDDFSCAN=(SEGPT.EQ.0).AND.EXTEND
    IF(.NOT.INTSCT) GO TO 355
    CALL DRAWH(XAMXR,IY)
    CALL DRAWV(XAMXR,IY)
^L

```

Since all segment data has been updated to the next scan line, it is necessary to evaluate the data for the current scan line by subtracting the slope from the X values. If the X value of the segment equals that of the sample point, then subroutine CROSS is called, which will update a flag, indicating that the edge is visible for this scan line. Subroutine SETLIN will be called if the edge is visible for the first time. At this time, the beginning X and Y coordinate values of the edge are stored. If the edge is going to exit on the next scan line, then it is drawn, since the edge will be deleted on the re-sorting of the segment list. The above procedure is performed for both the left and right edges of the segment.

Label 17 through label 15 handle the horizontal edge problem. If there are any horizontal edges on the current scan line, then the visible segment is checked to see if it has the same polygon which has a horizontal edge within the visible span. If so, then the line is drawn. Otherwise, the rest of the horizontal edge list for this scan line is searched for a match.

Finally, the edges in the segment list are checked to see if they were visible on the previous scan line but not this scan line. If so, then a line is drawn. The program checks to see if the end of the scan line has been reached, and then jumps to either process another scan line or get another visible segment.

```

355     IF (ZS(1).EQ.0) GO TO 15
C UPDATE LEFT EDGE IF VISIBLE SEGMENT EQUALS THE REAL SEGMENT.
      INC=SEG(ZS(1)+3)-SEG(ZS(1)+4)
      IF(SAM(1).NE.INC+1024) GO TO 16
      IF(SEG(ZS(1)+11).EQ.0) CALL SETLIN(INC,4,11,IY)
C DETERMINE IF THE PREVIOUS LINE AND CURRENT LINE CROSS
      CALL CROSS(INC,3,11,IY)
C CHECK TO SEE IF LEFT OF SEG WILL EXIT ON NEXT LINE
16     IF(SEG(ZS(1)+2)/262144.EQ.0) CALL DRAW(INC,4,11,IY)
C UPDATE RIGHT EDGE IF VISIBLE SEGMENT EQUALS THE REAL SEGMENT.
      INC=SEG(ZS(1)+5)-SEG(ZS(1)+6)
      IF(SAM(2).NE.INC) GO TO 17
      IF(SEG(ZS(1)+12).EQ.0) CALL SETLIN(INC,6,12,IY)
C DETERMINE IF THE PREVIOUS LINE AND CURRENT LINE CROSS
      CALL CROSS(INC,5,12,IY)
C CHECK TO SEE IF RIGHT OF SEG WILL EXIT ON NEXT LINE
17     IF((SEG(ZS(1)+2).AND.262143).EQ.0) CALL DRAW(INC,5,12,IY)
      CALL LDRPT(J,HBUCKY(IY+2))
      CURLIN=.TRUE.
C ARE THERE HORIZONTAL LINES ON THIS SCANLINE?
13     IF (J) GO TO 14
      CALL LOD2HF(J-1,HXL,HXR)
      CALL LOD2HF(J-2,POLYPT,J)
C DOES THIS POLYGON HAVE HORIZONTAL LINES
      IF (POLYPT.NE.SEG(ZS(1)+1)/262144) GO TO 13
      SAML=SAM(1)/262144-1
      SAMR=SAM(2)/262144
C IS THE LINE IN THE VISIBLE SEGMENT REGION?
      IF (SAML.GE.HXR) GO TO 13
      IF (SAMR.LE.HXL) GO TO 13
C DRAW HORIZONTAL LINE
      CALL DRAWV(MYMAXO(SAML,HXL)*262144,IY-CURLIN)
      CALL DRAWV(MYMINO(SAMR,HXR)*262144,IY-CURLIN)
      GO TO 13
14     CURLIN=.NOT.CURLIN
      J=HBUCKY(IY+2)/262144
      IF (.NOT.CURLIN) GO TO 13
15     IF (.NOT.ENDOFSCAN)GO TO 281
C BACK POINTER NOT NEEDED NOW.
      IF(SEGL2.NE.0)SEG(SEGL2)=0
      SEGXST=SEGS2
C GO THROUGH ALL SEG BLOCKS TO SEE IF AN EDGE WAS
C VISIBLE ON THE LAST SCAN LINE BUT WAS NOT DISPLAYED.
      ZS(1)=SEGXST
40     IF(ZS(1).EQ.0) GO TO 50
C CHECK LEFT EDGE
      IF((SEG(ZS(1)+11).AND.7).EQ.((IY+2).AND.7))
      I CALL DRAW(SEG(ZS(1)+3)-2*SEG(ZS(1)+4),4,11,IY+1)
C CHECK RIGHT EDGE
41     IF((SEG(ZS(1)+12).AND.7).EQ.((IY+2).AND.7))
      I CALL DRAW(SEG(ZS(1)+5)-2*SEG(ZS(1)+6),6,12,IY+1)
C GET NEXT SEGMENT BLOCK
42     ZS(1)=SEG(ZS(1)).AND.262143
      GO TO 40
50     IF (IY.GE.0) GO TO 204
      RETURN
      END
^L

```

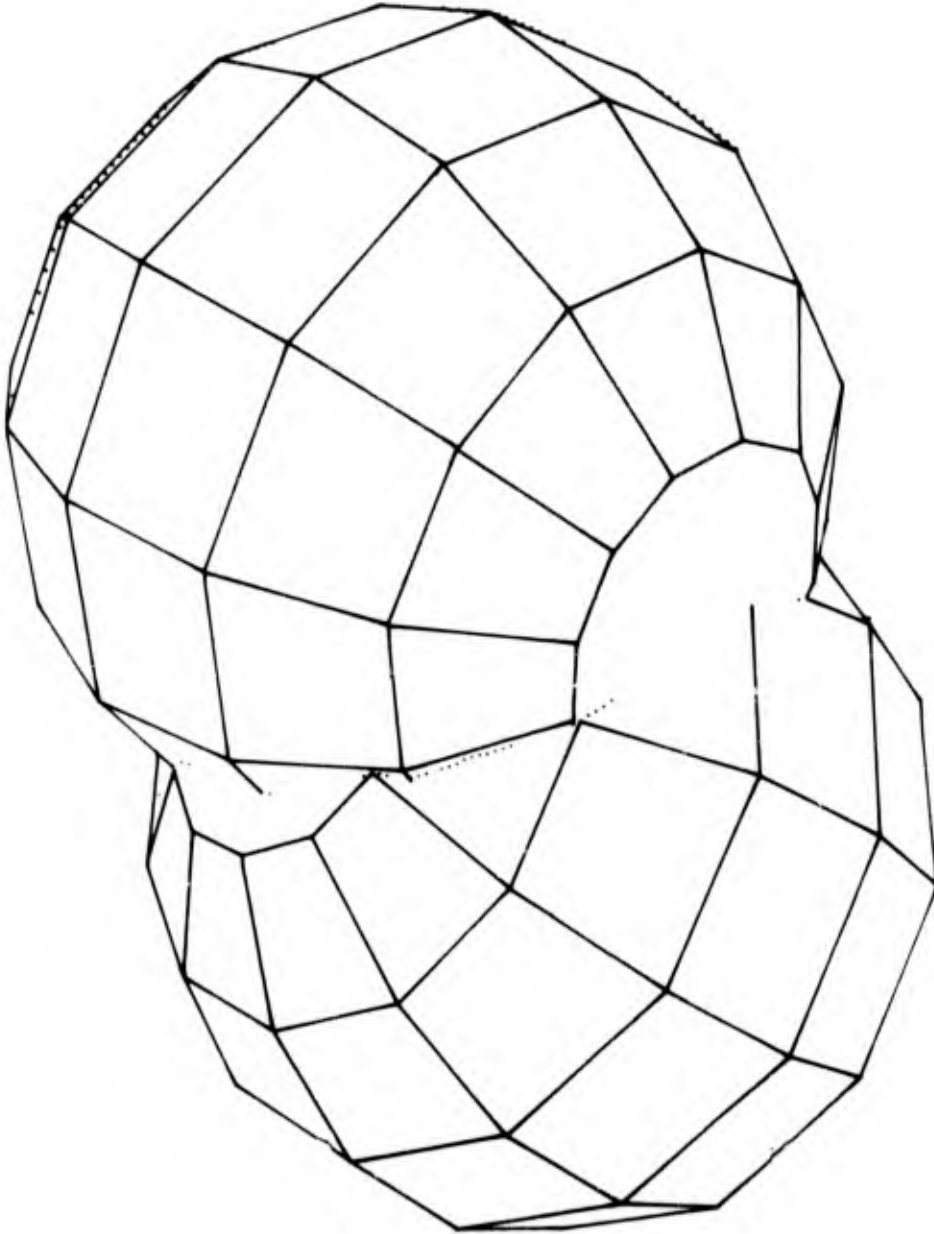


Figure 1:
Intersection of two spheres at 256 scan line resolution.

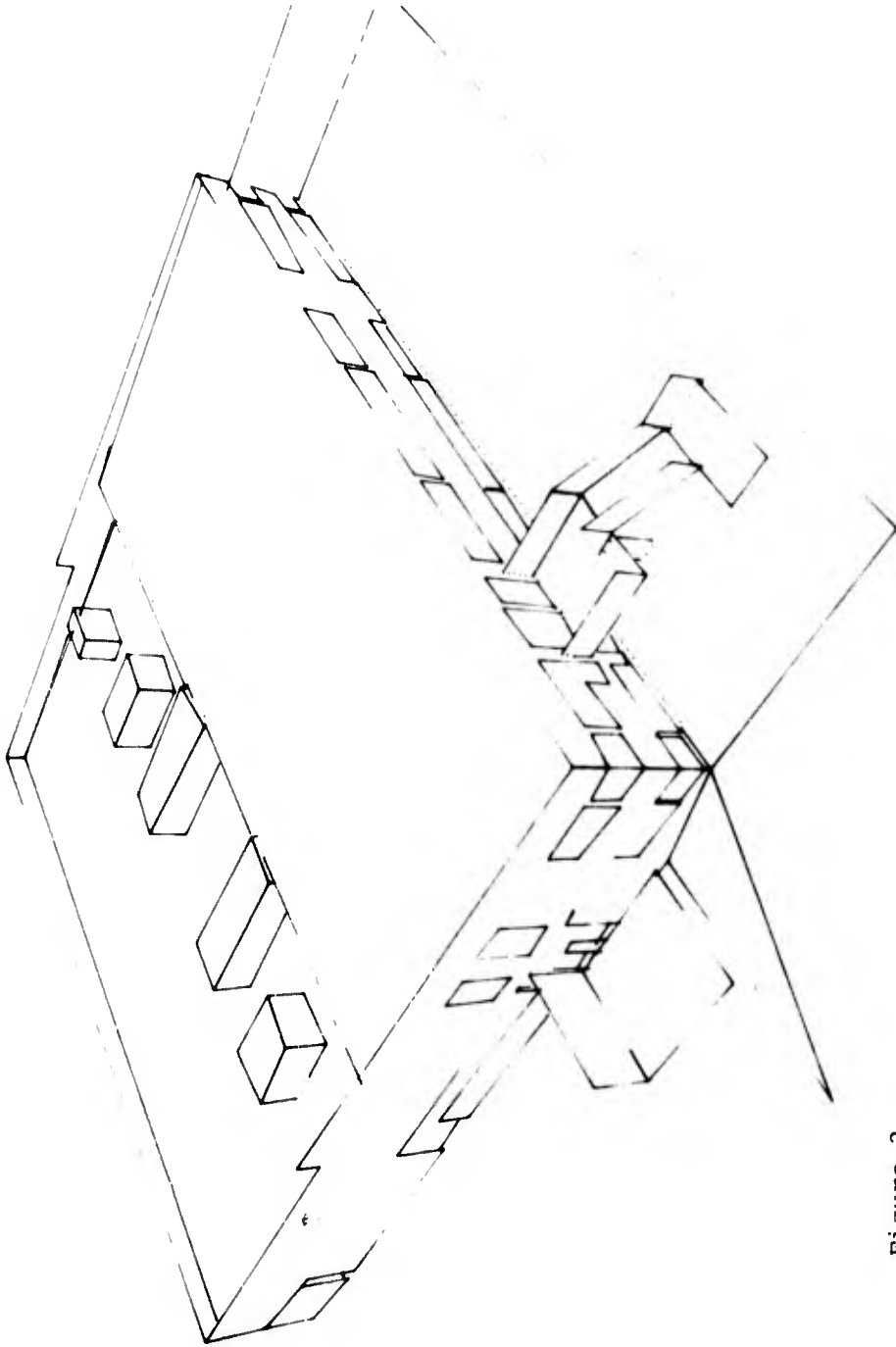


Figure 2:

This picture was processed at 256 scan line resolution. The perspective transformation distorts some of the objects. Processing time on the 295 edges which compose this picture was 7.2 seconds.

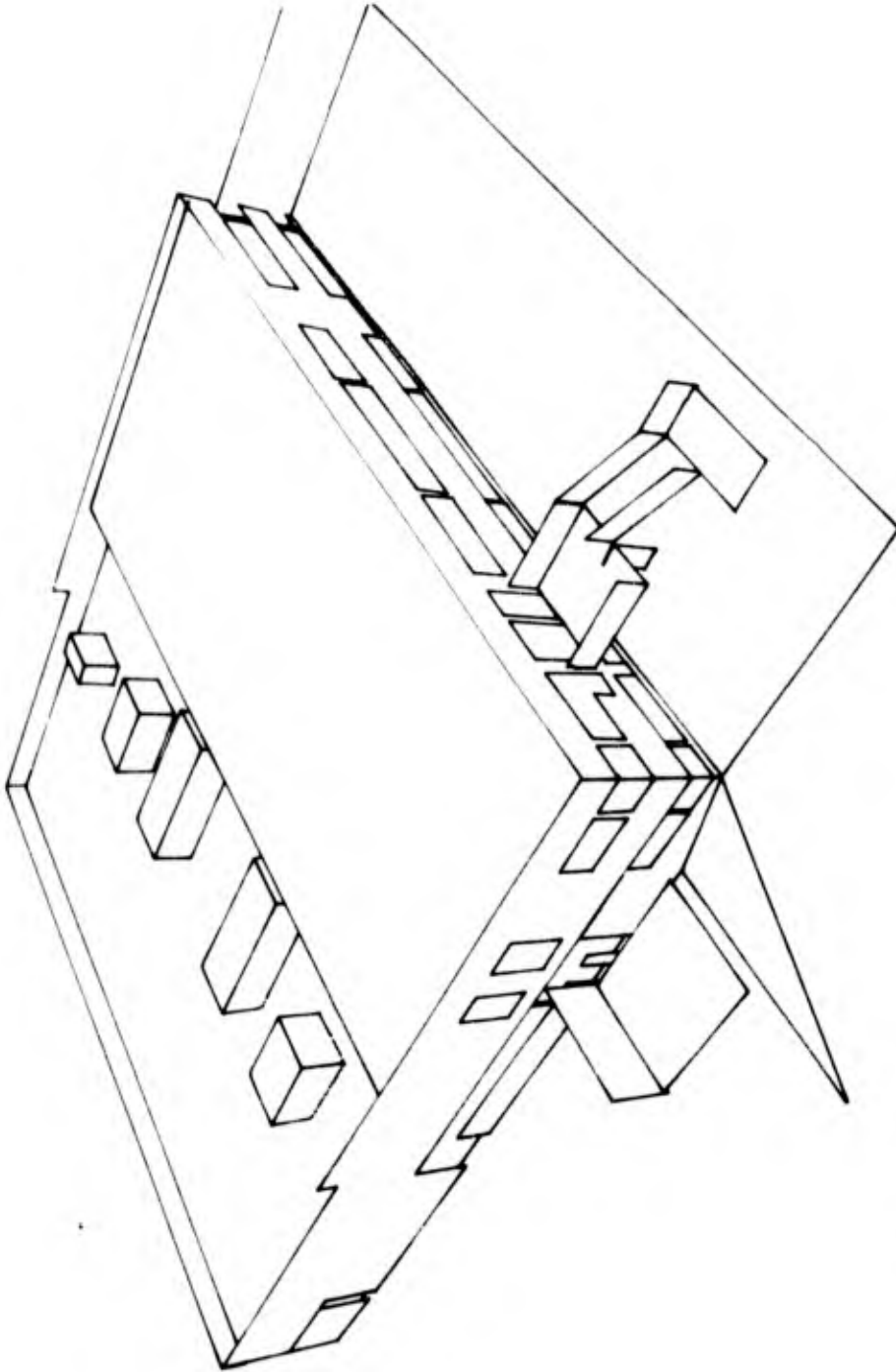


Figure 3:

This picture was processed at 1024 scan line resolution. Quality of the picture is greatly enhanced as a result of the increase in resolution. Processing time increased to 23.7 seconds. Notice that some lines are composed of dots. These are a result of intersection of surfaces.

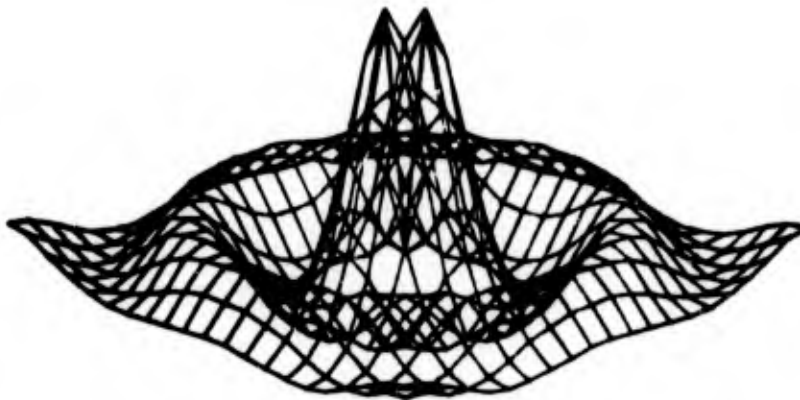
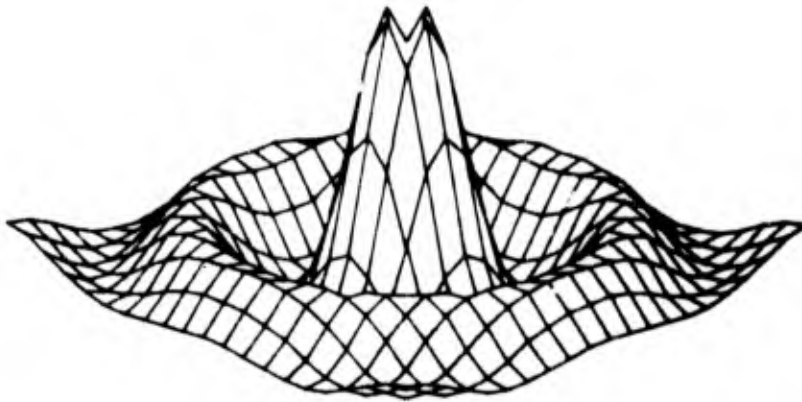


Figure 4:

1600 edge object with and without hidden surface removal at 256 scan line resolution. Computation time is 16.3 seconds.

REFERENCES

Romney, G. W., "Computer Assisted Assembly and Rendering of Solids," *Computer Science Technical Report 4-20*, Computer Science Division, University of Utah, Salt Lake City, August 1969.

Watkins, G. S., "A Real-time Visible Surface Algorithm," *Computer Science Technical Report UTEC-CSc-70-101*, Computer Science Division, University of Utah, Salt Lake City, July 1970.