

GODDARD SPAN
N-61-012
142 332
48P



FINAL REPORT

NASA GRANT # NAG5-91

T I T L E

Data Management and Language Enhancement for
Generalized Set Theory Computer Language for
Operation of Large Relational Databases

A B S T R A C T

This report covers the study of the relational database implementation in the NASCAD computer program system. The existing system is primarily used for computer aided design. Attention is also directed to a hidden-surface algorithm for final drawing output.

S U B M I T T E D B Y:

Gail T. Finley
Associate Professor
Computer Science Department
University of the District of Columbia

(NASA-CR-182868) DATA MANAGEMENT AND
LANGUAGE ENHANCEMENT FOR GENERALIZED SET
THEORY COMPUTER LANGUAGE FOR OPERATION OF
LARGE RELATIONAL DATABASES Final Report
(District of Columbia Univ.) 48 p CSCL 09B G3/61

N88-23446

Unclas
0142332

C O N T E N T S

- INTRODUCTION
- RELATIONAL DATABASE STUDY
- HIDDEN-SURFACE ALGORITHM

- APPENDIX:
 - Computer Code for Hidden-surface Algorithm
- BIBLIOGRAPHY

INTRODUCTION

The overall objective of this study is the description of the development of an interactive computer language to handle operations on large relational databases. The language contains features applicable to computer graphics and computer aided design. Additionally, this language is extended to contain commands reflecting the operators of relational algebra to manipulate relational databases.

At the inception of this study, the existing system at NASA, called NASCAD, was complete for primitive graphic commands, for numeric data, and for macros to create more complex graphics. Some work was also complete for creating the relational database. This study, therefore, extends the relational system with additional manipulation commands.

Initial proposals also investigated alternate data structures -- especially some hierarchical structure such as the B-tree. The existing data structure, a roving first fit, was considered adequate in the virtual memory environment; it remained undisturbed.

This report also addresses a hidden-surface algorithm to manipulate the final output of a graphic system -- the drawing. The hidden-surface algorithm has as its objective the removal (from the picture) of surfaces that are not visible when viewed from the planned perspective.

RELATIONAL DATABASE STUDY

Relational Systems

The relational database system derives its theory from the mathematical theory of relations (relational algebra and relational calculus). The term relation is derived as follows. Given the sets D_1, D_2, \dots, D_n (not mutually distinct), R is a relation on these sets if it is a set of ordered n -tuples $\langle d_1, d_2, \dots, d_n \rangle$ such that d_1 belongs to D_1 , d_2 belongs to D_2 , ..., and d_n belongs to D_n , where the sets D_1, D_2, \dots, D_n are called the domains of R . R then is considered a relation of degree n .

In a practical sense, a relational database may be pictured as interrelated flat files or tables. These files can be easily represented by the existing notation available in most current higher level languages, namely, the representation of a 2-dimensional array with rows and columns. Each row is considered a tuple and each column represents a domain. Restrictive properties of a relation are: no two rows (tuple) can be identical; and the order of the rows or columns is not significant.

An example used in standard information processing follows. This example represents a file of students where each row (tuple) represents information about a single student, and each column (domain) represents a particular item of information.

STUDENT RELATION

SS	NAME	ADDR	ZIP	MAJOR
542	Adams	125 A St	20005	EE
543	Jones	136 J St	20003	CE
546	Smith	146 K St	20001	CS

The following example would apply to describing a primitive function in a graphic system.

PRIMITIVE RELATION

OPERATION	VERTEX1 (V1)		VERTEX2 (V2)		COLOR
Line	X1	Y1	X2	Y2	Red
Line	X11	Y11	X22	Y22	Blue
Line	X13	Y13	X23	Y23	Red
Line	X1	Y1	X23	Y23	Green

Relational algebra is a collection of operations on relations. The select, project, and join operators, along with others not mentioned here, constitute the relational algebra. Each operator takes one or two relations as operands and produces another relation.

The select operator takes an existing relation as an input and produces a horizontal subset of that relation - namely that subset of tuples (rows) that satisfy a particular condition specified in the operator statement. That particular condition may be a single attribute or a comparison of attributes within a domain (column). For example: Select lines (defined in tuples)

whose color attribute (defined in the domain, Color) is red. This yields a relation that is a collection of lines whose color attribute is red.

The project operator takes an existing relation as an input and produces, by contrast, a vertical subset of that relation—namely the subset of attributes (columns) specified in a particular order and having any duplicate tuples within the attribute removed. For example: Consider the previous example and lines expressed as vertices V1, V2. After the select produces a relation of red lines, then a project operations over V1 yields a relation showing unique start vertices for all red lines.

The join operator is basically a combination of two relations over a common domain to yield a new relation, generally a wider table containing domains of both predecessor relations with the common domain eliminated. For example: Noting the previous example, consider a join of two relations over the V1 domain. This would yield a relation showing all lines and their attributes emanating from the same starting vertex (regardless of other attributes - color, etc.).

Relational Implementation

Modifications were made to the existing NASA program, NASCAD, to add the commands for the relational operations, project and join. Initial commands for creating and displaying relations as well as SELECT for manipulation of tuples were already

in the system.

The project command (PRJCT) manipulates domains yielding all specified domains in the resulting new relation. Conversely, the command, UNPRJCT, yields all unspecified domains in the resulting new relation. UNPRJCT is desirable when a relatively small number of domains are deleted. The user may specify the domain by name or by number. Domains in the resulting relation are ordered as they were specified in the command string. Thus, PRJCT may be used to reorder domains.

The join command (JOIN) merges two relations over a specified domain. The command string provides for definition of the following: two input relations; a domain (by name or number) in each input relation; and the resulting relation. Theoretically the name of the key domain in each input relation should be the same. However, for flexibility, provision is made for a different "name" to belong to "logically" identical key domains that exist in different relations. (Note that an alias type command could also handle this kind of flexibility.) The key domain may be referred to by name or by number. JOIN provides for operators other than equal such as: greater than and not equal thus allowing the effect of expanded relational operations. The key domain is removed in the resulting relation; this is consistent with traditional join operations.

Each command follows the accepted rules for relational algebra. General command syntax by keyword follows the standard NASCAD form. Identifier names for relations and domains also

follow NASCAD conventions.

At the inception of the study, the NASCAD system, implemented at NASA on a VAX 11/780 Digital Equipment Corporation computer (written in FORTRAN 77) was already in place. The system consists of four basic parts: the command language interpreter, macro interpreter, the editor, and database management system. The command language consists of operating system commands, macro commands, data handling commands, and edit commands. The data base management system is broken into three logical units handling data separately for graphic, numeric, and macro types. The initial commands to create the relational database and the select operator were also in place at that time. These commands as well as additional commands are implemented as macro commands.

In order to provide for change with minimal disruption of existing programs, any modifications are modular -- made as independent segments of existing programs or as separate subroutines. The following program modules are affected by the change: DEFTBL, DEFTAB (with entry points CPYCOL and ADJTAB), and SYSTBL. All work was done from a remote terminal at the university linked to the Vax 11/780 computer at the NASA facility. Testing included runs of expected command variations as well as error conditions. Program listings are filed at NASA.

The following table lists and describes the relational commands.

RELATIONAL COMMANDS

Notation:

R relation by name
 D domain or column by name or by number
 T tuple of row
 d data in domain or column according to specified format
 EQ, NE, GT standard logical operations equal, not equal, greater

Relational Commands Existing Prior to Study

ATTACH	R	retrieves R
SETTABL	R	retrieves R
DEFTBL	number of D, R by name	defines new R
DEFCOL	col number Di, D by name, format for di	defines column or domain of R
DEFROW	d1, d2, ...dn	defines data in each tuple or row
DSPTBL		displays current R
SELECT	Di, logical operator, constant	performs select operation

Relational Commands Implemented in Study

PRJCT	D1, D2, ..., Dn	project operation over D
UNPRJCT	D1, D2, ..., Dn	converse of project; D represents deleted domains
JOIN	R1, R2, D1, logical operator, D2, R3	performs join operation over R1 and R2 according to the logical operation with D1 and D2 to yield R3. Normally D1 and D2 are the same key domain. D1 is in R1 and D2 is in R2.

General comments and restrictions:

- Commas in above command strings are for clarity in this text; NASCAD uses blanks as delimiters in command strings.
- Domains may not be referenced both by name and number in the same command string.

HIDDEN-SURFACE ALGORITHM

Hidden-line/surface Algorithms

One end product of a graphic system is obviously the final drawing. With engineering applications, the drawing of a three dimensional object is made up of precise points, lines, and curved surfaces (represented in this application as flat polygon patches) that are stored in the drawing database to be displayed on a particular graphic device. This drawing is considered viewed from a certain perspective and the (three dimensional) x-y coordinates are projected onto (two dimensional) screen coordinates with the z coordinate representing depth. With the figure completely outlined on the screen (wire frame) all lines and surfaces are shown. The object of hidden-line/surface algorithms is to remove those lines/surfaces from the displayed drawing that are not visible when viewed from the planned perspective. The hidden-line algorithm outlines only the visible lines. The hidden-surface problem renders all visible surfaces on the screen; shading and coloring of the object are considered part of the problem.

The hidden-line/surface problem is one of the oldest in the graphic field. Many algorithms have been developed over the years. All algorithms share one common fault; they are very time consuming. To some extent, the algorithms are suitable for one graphic device versus another. A color raster graphics terminal is the chosen device for this project. For this reason

and because it had been proven effective, the Watkins scan-line algorithm was chosen as the initial algorithm to solve the problem. [Newman & Sproull, 1st edition]

The Watkins algorithm is similar to other scan line algorithms. It operates in image space on the basis of a raster of scan lines. As an image space algorithm, it seeks to compute what the image will be only at each of the resolvable dots on the display screen. The scan lines are assumed to be horizontal-parallel to the x-axis of the image plane coordinate system. The process has few basic steps. The drawing is scanned from the bottom up. Therefore edges are sorted by Y so that only those intersecting the current scan line need be examined. Appropriate sample spans are chosen across the scan-line; this involves a form of X sort. An elimination process of sample spans occurs. Lastly, the segments are searched for visibility - a Z search. The process is repeated at each scan-line.

Implementation and Testing

The program system implements the Watkins hidden-surface algorithm for display of the graphic solids that will be generated by the, NASA based, NASCAD computer aided design system.

Preliminary data processing takes the output from the NASCAD system that represents the three-dimensional object as flat polygon patches projected onto the plane of the screen(x-y coordinates) with the Z coordinate representing depth. The

figure is displayed in outline form on the screen with all parts showing. The algorithm proceeds to scan the figure from bottom to top, one line at a time. Those edges that intersect the current Y scan line are examined for X intersection and lastly for visibility by level of Z depth. The visible components of the figure are filled in at each Y line. The process repeats itself until the top of the screen is reached.

The algorithm as described in the reference [Newman & Sproull, 1st edition] was written in SAIL, a higher level programming language. The implementation is written in FORTRAN IV and is run on the Dec 2060 (Digital Equipment Corporation) computer. Graphic output was produced on the Tektronics 4027 color terminal. Program documentation was completed in a report by Peter A. Brown, a university student, dated March 23, 1981. Test data of scenes typically used in graphic reference materials were produced by NASCAD and used for initial testing.

Final test data generated by NASCAD were figures of the space shuttle pallet and the pallet with the more complex telescope mounted. Tests were successful except those where the telescope was incorporated into the drawing. The point of concern was a center point where multiple lines joined. The figures were broken into individual components and various component combinations to speed test time (the entire figure took 26 minutes to draw on the screen). The case of selected intersecting (or penetrating) planes was the broader problem with the Watkins algorithm itself. [Newman & Sproull, 2nd edition]

A tripod mounted, 35 mm camera was used to present test results as color photographic slides taken of the computer terminal screen at various stages of solution. Extensions were made to provide hard copy visual output on a black and white graphics device. This was done by creating patterns in black and white to represent any of eight colors used simultaneously. This technique was effective where figures had relatively large surface areas. Details were not clearly seen in the black and white pattern, even when patterns were carefully chosen to match the smallest surface area with the smallest pattern.

A P P E N D I X

COMPUTER CODE FOR HIDDEN-SURFACE ALGORITHM

This program is written in the FORTRAN IV programming language for the DEC 2060 computer (Digital Equipment Corporation). The graphic output devices that are supported are:

- Tektronix 4027 Color Terminal
- Tektronix 4013/4015 Series Black And White Terminal
- RAMTEC Terminal (Unimplemented)

C THIS DOCUMENTATION SPECIFIES ALL SUBROUTINES USED
C IN THE WATKINS HIDDEN SURFACE ALGORITHM. IN ADDITION
C A BRIEF DISCRPTION IS GIVEN OF THE SUBROUTINES
C MENTIONED.

C*****

C SUBROUTINE DESCRIPTION
C LOADBX THIS ROUTINE TAKES THE PRESENT SEGMENT, AND LOA
C LOADS IT INTO THE BOX. THE EXTREMITIES OF THE
C SEGMENT ARE REMEMBERED AS THE EXTREMITIES
C OF THE BOX.

C XPANBX THE PRESENT SEGMENT IS ADDED TO THE BOX, IF
C NECESSARY, THE EXTREMITIES OF THE BOX ARE
C EXPANDED TO ENCLOSE THE NEW SEGMENT.

C (FUNCTION)
C BZINT IF ONLY ONE SEGMENT IS IN THE BOX, WE MAY
C HAVE A DESIRE TO COMPUTE THE 'DEPTH' OF
C THAT SEGMENT AT SEVERAL POINTS. THE BZINT
C FUNCTION DOES THIS, GIVEN AN XS
C CO-ORDINATE AS ARGUMENT.

C (FUNCTION)
C ZINT THIS FUNCTION COMPUTES THE DEPTH OF THE
C SEGMENT BEING LOOKED AT, GIVEN AN XS
C COORDINATE AS ARGUMENT.

C RMXSRT THIS ROUTINE REMOVES A SEGMENT FROM THE
C XSORTLIST.

C RETBLK RETURN A SEGMENT BLOCK TO FREE STORAGE.

C GETBLK 1GETS A BLOCK FROM FREE STORAGE AND INITIALIZE
C YLEFT AND YRIGHT ENTRIES TO ZERO.

C PIXSRT THIS ROUTINE PUTS THE SEGMENT AT THE HEAD
C OF XSORT LIST.

C SHWCLS CALL SHWCLS , CALL EFRAME TO END THE FRAME
C AND PUT IT UP ON THE SCREEN.

C STOPIC RECORD DISPLAY DATA IN AN ARRAY.
C STOPIC TAKES TWO ARGUMENTS: THE XS POSITION
C AT WHICH THE SEGMENT STARTS AND THE INDEX
C OF THE VISAIBLE SEGMENT. IF THIS INDEX IS
C ZERO, THEN THIS SECTION OF THE SCAN-LINE
C IS BLANK. STOPIC RECORDS A COLLECTION OF
C PAIRS X1, X2, X3 THESE ARE USED, AT THE END
C OF THE SCAN-LINE TO CREATE SHADING
C COMMANDS FOR THIS SCAN-LINE.

C RECSAM THE RECORD SAMPLE ROUTINE IS CONCERNED WITH
C THE COLLECTION OF SAMPLE POINTS FOR THE
C TRAVERSE OF THE NEXT SCAN-LINE. A SAMPLE POINT
C IS RETAINED IF A SPAN EDGE CORRESPONDS TO THE
C OF THE VISIBLE SEGMENT SAMPLE POINTS ARE
C RECORDED IN A LIST. SAMFST POINTS TO THE FIRST
C ENTRY IN THE LIST, SAMLST TO THE LAST. SAMLINK
C IS AN ARRAY OF POINTERS. SAMX IS THE X VALUE OF TH
C THE SAMPLE POINT.

C PUTSAM PUT A SAMPLE IN THE SAMPLE LIST FOR NEXT SCAN-

C LINE.GET A FREE SAMPLE BLOCK AND FIX UP FREE STORA
C STORAGE.ALSO IT RECORDS X POSITION OF SAMPLE POINT.

C*****

SUBROUTINES WHICH ARE INVOKED BY THE THINKER

C*****

C STOPIC
C RECSAM
C PUTSAM
C STOPIC
C PIXSRT
C RETBLK
C GETBLK

CC*****

SUBROUTINES WHICH ARE INVOKED BY THE LOOKER

C*****

LOADBX
XPANBX
BZINT
ZINT

C*****

SUBROUTINES WHICH ARE INVOKED BY THE CONTROLLER

C*****

RMXSRT
RETBLK
GETBLK
PIXSRT
LOOKER
THINK
SHOW
SHWCLS

* THE FOLLOWING IS A GROUP OF *.CMN

```
C  USED BY CONTROLLER,GETBLK,RETBLK
    INTEGER FRELST
    COMMON/BLK/FRELST
C  COMMON BLOCKS USED FOR INPUT DATA
    COMMON /VERTEX/ XS(MAXPNT),YS(MAXPNT),ZS(MAXPNT)
    INTEGER P1,P2,V1,V2,ENTLST
    INTEGER EDGLST
    INTEGER XRES,YRES
    COMMON/EDG/P1(MAXEDG),P2(MAXEDG),V1(MAXEDG),V2(MAXEDG),
    1  ENTLST(MAXEDG),LINKED(MAXEDG),EDGLST
    INTEGER SHAD
    COMMON/COLOR/ SHAD(MAXPLY)
    COMMON/DEV/IDEV,XRES,YRES
C*****END OF INPUT.CMN*****
C  BOX BLOCK USED BY LOOKER & SUBROUTINES
C
    INTEGER BOXCNT,BOXTYP,BFULL,BSEG1,BSEG2,SFULL,SEG
    COMMON/LOKBX1/ BOXCNT,BOXTYP,BFULL,BSEG1,BSEG2,
    1  BXLEFT,BXRGT,BZLEFT,BZRGT,
    2  BZMIN,BZMAX,DIV,SFULL,SEG
    COMMON/LOKBX2/ SDIV,SXLEFT,SXRGT,SZLEFT,SZRGT
C  POLYGON DATA BLOCKS
    INTEGER CHNGNG,SEGLST
    COMMON/PLYGON/ CHNGNG(MAXPLY),SEGLST(MAXPLY)
C
C  SEGMENT DATA BLOCKS
    INTEGER POLYGN,PLYSEG,XSRTLT,XSRTRT,ACTIVE,YLEFT,YRIGHT
    COMMON/SEGBK1/POLYGN(MAXSEG),PLYSEG(MAXSEG),
    1  ACTIVE(MAXSEG),
    2  XSRTLT(MAXSEG),XSRTRT(MAXSEG),
    3  YLEFT(MAXSEG),YRIGHT(MAXSEG)
C
    COMMON/SEGBK2/ XLEFT(MAXSEG),DXLEFT(MAXSEG),
    1  ZLEFT(MAXSEG),DZLEFT(MAXSEG),
    2  XRIGHT(MAXSEG),DXRGT(MAXSEG),
    3  ZRIGHT(MAXSEG),DZRGT(MAXSEG)
C*****
C  MISCELLANEOUS COMMON ROUTINE 9
    INTEGER VISPOS,VISSEG
    COMMON/MISC1/ VISPOS(MAXSEG),VISSEG(MAXSEG)
C
    INTEGER SAMLNK,SAMX
    COMMON/SAM/ SAMLNK(IMAX),SAMX(IMAX)
C  USED BY CONTROLLER,STOPIC
    INTEGER SEGCNT
    COMMON/PIC/LASSEG,SEGCNT
C  SINGLE VARIABLES USED BY THINKER,CONTROLLER
C  RRECSAM, PSMPLE
    INTEGER SAMFST,SAMFRE,SAMLST
    COMMON/SAMSIN/ SAMFST,SAMFRE,SAMLST
C  USED BY LOOKER,THINKER,CONTROLLER,RECSAM
    COMMON/SPAN/ SPANRT,SPANLT,IMPLFT
C  USED IN CONTROLLER,PIXSRT,RMXSRT
    INTEGER SEGFST
    COMMON /SRT/ SEGFST
C  USED BY CONTROLLER,,THINKER
    INTEGER PREV
    COMMON/THK/ IMPLST,MPLST2,PREV
```

```
C      MODULE USED BY LOOKER.
C*****
C FILE NAME SUB4.FOR
C FUNCTION NAME 'BZINT;'.
C THE BZINT FUNCTION IS DESIGNED TO COMPUTE THE
C 'DEPTH' OF A SEGMENT AT SEVERAL POINTS. GIVEN
C AN XS ARGUMENT.
C*****
```

```
      FUNCTION BZINT (X)
      INCLUDE 'MAIN.PAR'
      INCLUDE 'LOOKBX.CMN'
      IF(BXRGHT .EQ. BXLEFT) GO TO 10
      BZINT = BZLEFT + (BZRGHT - BZLEFT)
1      *(X -BXLEFT)/(BXRGHT-BXLEFT)
      RETURN
10     BZINT = BZLEFT
      RETURN
      END
```

```

SUBROUTINE CONT
C BEGIN ELIMINATE
  INCLUDE 'MAIN.PAR'
  INCLUDE 'MAIN.CMN'
  INCLUDE 'INPUT.CMN'
  INCLUDE 'BLK.CMN'
  INCLUDE 'SRT.CMN'
  INCLUDE 'SAMSIN.CMN'
  INCLUDE 'SPAN.CMN'
  INCLUDE 'THK.CMN'
  INCLUDE 'LOOKBX.CMN'
  INCLUDE 'PIC.CMN'
  INTEGER YENTER(IYRESB)
  INTEGER GETBLK
  REAL LSTUSE, ZFIRST, XSLOPE, RELDLY, ZSLOPE,
  1 XFIRST
  INTEGER CHANGE, SEGLO, CURSEG, SEG1, YLAST,
  1 K, PTR, SEGOUT, Y2, TE1, NEXT, Y1, Y,
  2 YFIRST, P, J, TE2, MX, VV1, DELY, PCHLST,
  3 SEGACT, ITH, IX, SAMPLE, ITEMP,
  4 MXSG, VV2, I
  MXSG = MAXSEG - 1
  DO 10 I = 1, MXSG
    ACTIVE(I) = I+1
10  CONTINUE
    FRELST = 1
    MXSG = MAXSEG * 2 - 1
    DO 20 I = 1, MXSG
      SAMLNK(I) = I + 1
20  CONTINUE
    SAMFRE = 1
C BEGIN HIDDEN-LINE INITIALIZATION
    IMPLST = 0
    MPLST2 = 0
    SEGFST = 0
    PTR = EDGLST
25  IF(PTR .EQ. 0) GO TO 60
    NEXT = ENTLST(PTR)
    IF (((P1(PTR).NE.0).AND.(SHAD(P1(PTR)).NE.
    1 0)).OR.((P2(PTR).NE.0).AND.(SHAD(P2(PTR))
    1 .NE.0))).EQ.0) GO TO 50
    J = V1(PTR)
    K = V2(PTR)
    IF(YS(J).LE.YS(K)) GO TO 30
    ITEMP = V1(PTR)
    V1(PTR) = V2(PTR)
    V2(PTR) = ITEMP
    J = K
30  I=YS(J) +.999999
    IF((I .LT. 1).OR.(YRES.LT.I))GO TO 40
    ENTLST(PTR) = YENTER(I)
    YENTER(I) = PTR
    GO TO 50
40  WRITE(5,1000)
1000 FORMAT(1X,'EDGE OUT OF BOUNDS')
    STOP
50  PTR = NEXT
    GO TO 25
C*****
60  CONTINUE
    CALL SHWINT

```

```

C*****
C  END_HIDDEN_LINE_INITIALIZATION

C  DISPLAY_GENERATION

      DO 730 Y = 1, YRES
C      WRITE(5,789)Y
789    FORMAT(1X,'Y= ',I4)
C  BEGIN PROCESSING BEFOR STEPPING ACROSS SCAN-LINE
      PCHLST = -1
      SEG = SEGFST
70    IF(SEG .EQ. 0) GO TO 100
      XLEFT(SEG) = XLEFT(SEG) + DXLEFT(SEG)
      XRIGHT(SEG) = XRIGHT(SEG) + DXRGHT(SEG)
      ZLEFT(SEG)=ZLEFT(SEG) + DZLEFT(SEG)
      ZRIGHT(SEG)=ZRIGHT(SEG) + DZRGHT(SEG)
      Y1 = YLEFT(SEG) + 1
      YLEFT(SEG) = YLEFT(SEG) + 1
      Y2 = YRIGHT(SEG) + 1
      YRIGHT(SEG) = YRIGHT(SEG) + 1
      IF((Y1.NE. 0).AND.(Y2.NE.0)) GO TO 90
      PTR = POLYGN(SEG)
      IF(PTR.NE. 0) GO TO 80
C*****
      CALL RMXSRT(SEG)
C*****
C*****
      CALL RETBLK(SEG)
C*****
      GO TO 90
80    IF(CHNGNG(PTR).NE.0) GO TO 90
      CHNGNG(PTR) = PCHLST
      PCHLST = PTR
90    SEG = XSRTRT(SEG)
      GO TO 70
100   PTR = YENTER(Y)
110   IF(PTR .EQ. 0) GO TO 260

C  BEGIN ENTERING EDGES
      VV1 = V1(PTR)
      VV2 = V2(PTR)
      YFIRST = YS(VV1)
      YLAST = YS(VV2)
      DELY = YFIRST - YLAST
      RELDLY = YS(VV2) - YS(VV1)
      IF(DELY.GE.0) GO TO 255

C--BEGIN MAKE SEGMENTS FOR THIS EDGE

      XSLOPE = (XS(VV2)-XS(VV1))/RELDLY
      XFIRST = XS(VV1) + XSLOPE * (Y-YS(VV1))
      ZSLOPE = (ZS(VV2)-ZS(VV1))/RELDLY
      ZFIRST = ZS(VV1) + ZSLOPE * (Y-YS(VV1))
      DO 250 P = P1(PTR),P2(PTR)

C--BEGIN LOOK AT BOTH POLYGONS BORDERING THIS EDGE
      IF(P.EQ.0) GO TO 250
C--BEGIN A REAL POLYGON
      IF(CHNGNG(P).NE. 0 ) GO TO 120
      CHNGNG(P) = PCHLST
      PCHLST = P

```

```

120     SEG = SEGLST(P)
        PREV = 0
        J = 3
130     IF(SEG.EQ.0) GO TO 190
C--BEGIN LOOK AT SEGMENTS

        TE1 =(XFIRST.LT.XLEFT(SEG)).OR.(XFIRST.EQ.XLEFT(SEG)
1         .AND.XSLOPE.LT.DXLEFT(SEG))
        TE2=(XFIRST.LT.XRIGHT(SEG)).OR.(XFIRST.EQ.XRIGHT(SEG)
1         .AND.XSLOPE.LT.DXRGHT(SEG))
        Y1 = (YLEFT(SEG).LT.0)
        Y2 = (YRIGHT(SEG).LT. 0)
        I = -(TE1*8)-(TE2*4)-(Y1*2)-(Y2)
        IF((I.LT.0).OR.(I.GT.15))GO TO 170
        IF(I.NE.11)GO TO 140
        J = 0
        GO TO 180
140     IF((I.LT.13).AND.((I.NE.5).AND.(I.NE.10)))GO TO 150
        J = 1
        GO TO 180
150     IF(I.NE. 7) GO TO 160
        J = 2
        GO TO 180
160     J = 3
        GO TO 180
170     WRITE(5,1010)
1010    FORMAT(1X,'CASE I ERROR IN CONTROLLER')
        STOP
180     IF( J .NE. 3 ) GO TO 190
        PREV = SEG
        SEG = PLYSEG(SEG)
        GO TO 130

C--END LOOK AT SEGMENTS
190     IF((J.NE.1).AND.(J.NE.3)) GO TO 220

C--BEGIN INSERT NEW SEGMENT BETWEEN PREV AND SEG
        DUMMY = 0
        SEG1 = GETBLK(DUMMY)
        POLYGN(SEG1) = P
        XLEFT(SEG1) = XFIRST
        DXLEFT(SEG1) = XSLOPE
        ZLEFT(SEG1) = ZFIRST
        DZLEFT(SEG1) = ZSLOPE
        YLEFT(SEG1) = DELY
C*****
        CALL PIXSRT(SEG1)
C*****
        IF(PREV .EQ. 0) GO TO 200
        PLYSEG(PREV) = SEG1
        GO TO 210
200     SEGLST(P) = SEG1
210     PLYSEG(SEG1) = SEG
        GO TO 250
C--BEGIN SPLIT THE SEGMENT
        DUMMY = 0
220     SEG1 = GETBLK(DUMMY)
        POLYGN(SEG1) = P
        XLEFT(SEG1) = XLEFT(SEG)
        DXLEFT(SEG1) = DXLEFT(SEG)
        ZLEFT(SEG1) = ZLEFT(SEG)

```

```

DZLEFT(SEG1) = DZLEFT(SEG)
YLEFT(SEG1)=YLEFT(SEG)
YLEFT(SEG) = 0
XRIGHT(SEG1) = XFIRST
DXRGHT(SEG1) = XSLOPE
ZRIGHT(SEG1) = ZFIRST
DZRGHT(SEG1) = ZSLOPE
YRIGHT(SEG1) = DELY
C*****
CALL PIXSRT(SEG1)
C*****
IF(PREV .EQ. 0) GO TO 230
PLYSEG(PREV) = SEG1
GO TO 240
230 SEGLST(P) = SEG1
240 PLYSEG(SEG1) = SEG
PREV = SEG1

C--END SPLIT THE SEGMENT

C--END A REAL POLYGON

C--END LOOK AT BOTH POLYGONS BORDERING THIS EDGE

250 CONTINUE
C--END MAKE SEGMENTS FOR THIS EDGE
255 CONTINUE
PTR = ENTLST(PTR)
GO TO 110

C--END ENTERING EDGES

260 IF(PCHLST .EQ. -1) GO TO 375

C--BEGIN PROCESS A CHANGING POLYGON

P = PCHLST
PCHLST = CHNGNG(P)
CHNGNG( P ) = 0
PREV = 0
SEG = SEGLST(P)
270 IF(SEG .EQ. 0 ) GO TO 370
C--BEGIN TRANSFORM THE LIST
Y1 = YLEFT(SEG)
Y2 = YRIGHT(SEG)
IF((Y1.GE.0).OR.(Y2.GE.0))GO TO 280
C--BEGIN SCAN FURTHER
PREV = SEG
SEG = PLYSEG(SEG)
GO TO 360
280 IF((Y1.NE.0).OR.(Y2.NE.0)) GO TO 310
C--BEGIN REMOVE THIS SEGMENT
I = PLYSEG(SEG)
IF(PREV .EQ. 0) GO TO 290
PLYSEG(PREV) = I
GO TO 300
290 SEGLST(P) = I
C*****
300 CALL RMXSRT(SEG)
C*****
C*****

```

CALL RETBLK(SEG)
SEG = I
GO TO 360

C*****
310 IF((Y1.NE.0).OR.(Y2.GE.0)) GO TO 320
C--BEGIN MOVE RIGHT TO LEFT
YLEFT(SEG) = YRIGHT(SEG)
YRIGHT(SEG) = 0
XLEFT(SEG) = XRIGHT(SEG)
DXLEFT(SEG) = DXRGHT(SEG)
ZLEFT(SEG) = ZRIGHT(SEG)
DZLEFT(SEG) = DZRGHT(SEG)
GO TO 360

C--BEGIN RIGHT SIDE IS EMPTY-LOOK AT NEXT SEGMENT

320 NEXT = PLYSEG(SEG)
IF(NEXT.NE.0) GO TO 330
WRITE(5,1020)
1020 FORMAT(1X,'NEXT ERROR')
STOP
330 IF(YLEFT(NEXT).GE.0) GO TO 340

C--BEGIN MOVE NEXT'S LEFT TO MY RIGHT

YRIGHT(SEG) = YLEFT(NEXT)
YLEFT(NEXT) = 0
XRIGHT(SEG) = XLEFT(NEXT)
DXRGHT(SEG) = DXLEFT(NEXT)
ZRIGHT(SEG) = ZLEFT(NEXT)
DZRGHT(SEG) = DZLEFT(NEXT)
GO TO 360
340 IF(YRIGHT(NEXT).GE. 0) GO TO 350

C--BEGIN MOVE NEXT'S RIGHT TO MY RIGHT'

YRIGHT(SEG) = YRIGHT(NEXT)
YRIGHT(NEXT) = 0
XRIGHT(SEG) = XRIGHT(NEXT)
DXRGHT(SEG) = DXRGHT(NEXT)
ZRIGHT(SEG) = ZRIGHT(NEXT)
DZRGHT(SEG) = DZRGHT(NEXT)
GO TO 360

C--BEGIN DELETE 'NEXT' ENTIRELY

350 PLYSEG(SEG) = PLYSEG(NEXT)
C*****
CALL RMXSRT (NEXT)
C*****
C*****
CALL RETBLK(NEXT)
C*****

C--END RIGHTSIDE IS EMPTY-LOOK AT NEXT SEGMENT

360 GO TO 270
C--END TRANSFORM THE LIST
370 GO TO 260
C--END PROCESS A CHANGING POLYGON

375 GO TO 385
380 IF(CHANGE .EQ.0) GO TO 430
385 CONTINUE

C--BEGIN SORT THE XSORTLIST

CHANGE = 0
SEG = SEGFST
390 IF(SEG .EQ.0) GO TO 425

C--BEGIN RAMBLE DOWN LIST

I = XSRTRT(SEG)
IF(I .EQ. 0) GO TO 425
IF(XLEFT(SEG).LE. XLEFT(I))GO TO 420

C--BEGIN SWAP

CHANGE = -1
IF(XSRTLT(SEG).EQ. 0) GO TO 400
K = XSRTLT(SEG)
XSRTRT(K) = I
400 K = XSRTLT(SEG)
XSRTLT(I) = K
XSRTLT(SEG) = I
IF(XSRTRT(I) .EQ. 0) GO TO 410
K= XSRTRT(I)
XSRTLT(K) = SEG
410 K = XSRTRT(I)
XSRTRT(SEG) = K
XSRTRT(I) = SEG
IF(SEGFST.EQ.SEG) SEGFST = I
GO TO 425

C--END SWAP

420 SEG = XSRTRT(SEG)
GO TO 390
425 CONTINUE
GO TO 380

C--END RAMBLE DOWN LIST

C--END SORT THE LIST

C-- END PROCESSING BEFORE STEPPING ACROSS SCAN-LINE

430 SEGACT = 0
SEGCNT = 0
440 IF(IMPLST.EQ.0) GO TO 450
IMPLST = XSRTRT(IMPLST)
J = IMPLST

C*****

CALL RETBLK(J)

C*****

GO TO 440
450 IMPLST = MPLST2
MPLST2 = 0
CURSEG = SEGFST
SPANRT = 0
SAMPLE = SAMFST
SAMLST = 0
LSTUSE = 0
455 GO TO 465
460 IF(SPANRT .EQ. XRES) GO TO 700

465 CONTINUE
C--BEGINSAMPLE ACROSS THE SCAN LINE

SPANLT = SPANRT + 1
IF(SPANLT .LE. LSTUSE) GO TO 490

C--BEGIN MOVED TO RIGHT OF LAST SAMPLE SPAN

IF(SAMPLE .EQ. 0) GO TO 470

C--BEGIN MORE SAMPLES LEFT

SPANRT = SAMX(SAMPLE)
IX = SAMPLE
SAMPLE = SAMLNK(SAMPLE)
SAMLNK(IX) = SAMFRE
SAMFRE = IX
LSTUSE = SPANRT
GO TO 480

470 SPANRT = XRES
480 GO TO 500
490 SPANRT = LSTUSE
500 IMPLFT = 0

C--WHILE STATEMENT SIMULATION UNTIL STMT #670
C--WHEN THINKER IS TRUE, THIS WHILE BLOCK WILL

C--BE LEFT.

510 CONTINUE
BOXCNT = 0
SEGOUT = 0
PREV = 0
SEG = SEGACT
520 IF(SEG .EQ. 0) GO TO 590

C--BEGIN ACTIVE SEGMENTS

NEXT = ACTIVE(SEG)
IF(XRIGHT(SEG).GE.SPANRT + 1) GO TO 560

C--BEGIN IT ENDS IN THIS SPAN

IF(PREV.EQ. 0) GO TO 530
ACTIVE(PREV) = NEXT
GO TO 540
530 SEGACT = NEXT
540 ACTIVE(SEG) = SEGOUT
IF(SEGOUT .EQ. 0) SEGLO = SEG
SEGOUT = SEG
IF(XRIGHT(SEG).LT.SPANLT) GO TO 550

C*****
CALL LOOKER

C*****

C---END IT ENDS IN THIS SPAN

550 GO TO 580
560 IF(XLEFT(SEG).GT.SPANRT) GO TO 570
C*****
CALL LOOKER

```

C*****
570     PREV = SEG
580     SEG = NEXT
        GO TO 520
590     IF(CURSEG .EQ. 0) GO TO 640
        SEG = CURSEG
        IF(XLEFT(SEG) .GT. SPANRT) GO TO 640
        CURSEG = XSRTRT(CURSEG)
        IF(POLYGN(SEG) .NE. 0) GO TO 605

C-- BEGIN IMPLIED EDGE BLOCK

        IF(.NOT.((1.LE.XLEFT(SEG)).AND.(XLEFT(SEG)
1  .LE. XRES)).AND.((PLYSEG(SEG)/10000).EQ.
1  LASSEG)) GO TO 600

C BEGIN OK TO KEEP IMPLIED EDGE

        IMPLFT = SEG
        GO TO 630
C BEGIN THROW OUT IMPLIED EDGE
C*****
600     CALL RMXSRT(SEG)
C*****
C*****
        CALL RETBLK(SEG)
C-- END IMPLIED EDGE BLOCK

        GO TO 630
C--BEGIN REAL EDGE BLOCK

605     IF(XLEFT(SEG)+1.GE.XRIGHT(SEG))GO TO 630
        IF(XRIGHT(SEG).GE.SPANRT+1) GO TO 610
        ACTIVE(SEG) = SEGOUT
        IF(SEGOUT.EQ. 0) SEGLO = SEG
        SEGOUT = SEG
        GO TO 620
610     ACTIVE(SEG) = SEGACT
        SEGACT = SEG
C*****
620     CALL LOOKER
C*****
630     GO TO 590
C--END LOOKS GOOD

C--END REAL EDGE BLOCK
C-- END XSORT SEGMENT

640     CONTINUE
C*****
        CALL THINK(ITH)
C*****
        IF(ITH .NE.0 ) GO TO 680
        IF(SEGOUT.EQ. 0) GO TO 650
        ACTIVE(SEGLO) = SEGACT
        SEGACT = SEGOUT
650     I = DIV
        IF(I .LT. SPANRT) GO TO 660
C--BEGIN DIVIDE AT MID POINT
        I = (SPANLT + SPANRT)/2

```

```
        SPANRT = I
        GO TO 670
660     SPANRT = I
670     GO TO 510
```

C--END SUBDIVIDE SAMPLE SPACE

```
680     IF(IMPLFT .EQ. 0) GO TO 690
C*****
        CALL RMXSRT(IMPLFT)
        CALL RETBLK(IMPLFT)
C*****
C _END SAMPLE ACROSS THE SCAN LINE
```

```
690     GO TO 460
700     IF(SAMLST.EQ.0) GO TO 710
        SAMLNK(SAMLST) = 0
        GO TO 720
710     SAMFST = 0
C*****
720     CONTINUE
        IB = Y
        CALL SHOW(IB)
C*****
730     CONTINUE
C--END DISPLAY GENERATION
        CALL SHWCLS
C-- END ELIMINATE
        RETURN
        END
```

C MISCELLANEOUS PROCEDURE.

C*****

C FUNCTION NAME GETBLK.

C THIS FUNCTION GETS A BLOCK FROM FREE STORAGE

C AND INITIALIZE YLEFT AND YRIGHT ENTRIES TO ZERO.

C*****

```

      INTEGER FUNCTION GETBLK(DUMMY)
      INCLUDE 'MAIN.PAR'
      INCLUDE 'MAIN.CMN'
      INCLUDE 'BLK.CMN'
      I = FRELST
      IF(I .NE. 0) GO TO 20
      WRITE(5,4)
4      FORMAT(3X,'NO MORE FREE STORAGE')
      STOP
20     YLEFT (I) = 0
      YRIGHT(I) = 0
      FRELST = ACTIVE(I)
      GETBLK = I
      RETURN
      END
```

```
C      MODULE USED BY LOOKER.
C*****
C      FILE NAME 'SUB2.FOR'
C      SUBROUTINE NAME 'LOADBX'.
C      THIS ROUTINE TAKES THE PRESENT SEGMENT, AND "LOADS"
C      IT INTO THE BOX. THE EXTREMITIES OF THE SEGMENT ARE
C      REMEMBERED AS THE EXTREMITIES OF THE BOX.
C*****
```

```

      SUBROUTINE LOADBX
      INCLUDE 'MAIN.PAR'
      INCLUDE 'LOOKBX.CMN'
      BOXCNT = 1
      BOXTYP = 0
      BXLEFT = SXLEFT
      BXRGT = SXRGT
      BZLEFT = SZLEFT
      BZRGT = SZRGT
      BSEG1 = SEG
      BZMIN = BZLEFT
      BZMAX = BZRGT
      IF(BZMIN.LE.BZMAX) GO TO 10
      CALL SWAP(BZMIN,BZMAX)
10     DIV = SDIV
      BFULL = SFULL
      RETURN
      END
```

C FILE NAME SUB1
C SUBROUTINE NAME 'LOOKER'.

C THE LOOKER IS A SUBROUTINE WHICH EXAMINES THE SEGMENT
C INDEXED BY 'SEG', AND ADDS IT TO THE PRESENT BOX, ECT.

C VARIABLES USED BY THE LOOKER:

C VARIABLE DESCRIPTION

C BXLEFT, BXRGT LEFT AND RIGHT EDGES OF BOX.

C BZMIN, BZMAX NEAR AND FAR EDGES OF THE BOX.

C BZLEFT, BZRGT WHEN ONLY ONE SEGMENT IS IN THE BOX. THESE

C CONTAIN THE ZS COORDINATES OF THE LEFT

C AND RIGHT ENDS OF THAT SEGMENT.

C BOXCNT COUNT OF NUMBER OF SEGMENTS IN THE BOX.

C BOXTYP 1 IF WE HAVE COMPUTED THE INTERSECTION

C OF TWO PENETRATING SEGMENTS (IMPLIED EDGE)

C ELSE 0.

C DIV THE PLACE TO SUBDIVIDE THE SPAN IF NEEDED.

C BFULL TRUE IF THE ONE SEGMENT IN THE BOX IS A SPANNER..

C BSEG1 THE INDEX OF THE FIRST SEGMENT IN THE BOX.

C BSEG2 THE INDEX OF THE SECOND SEGMENT IN THE BOX

C (THIS IS KEPT BECAUSE OF IMPLIED EDGES)

C SXLEFT, SXRGHT XS COORDINATES OF LEFT AND RIGHT ENDS OF THE

C SEGMENT BEING EXAMINED.

C SZLEFT, SZRGHT SAME FOR ZS COORDINATES.

C SFULL TRUE IF SEGMENT BEING LOOKED AT IS A SPANNER.

C*****

C SUBROUTINES CREATED FOR USE BY THE LOOKER.

C*****

C SUBROUTINE FUNCTION

C LOADBX TAKES THE PRESENT SEGMENT, AND LOADS IT INTO

C THE BOX. THE EXTREMITIES OF THE SEGMENT ARE

C REMEMBERED AS THE EXTREMITIES OF THE BOX.

C XPANBX THE PRESENT SEGMENT IS ADDED TO THE BOX. IF

C NECESSARY, THE EXTREMITIES OF THE BOX ARE EXPANDED

C TO ENCLOSE THE NEW SEGMENT.

C BZINT IF ONLY ONE SEGMENT IS IN THE BOX, WE MAY HAVE

C A DESIRE TO COMPUTE THE DEPTH OF THAT SEGMENT

C AT SEVERAL POINTS. THE BZINT FCT DOES THIS, GIVEN

C AN XS COORDINATE AS ARGUMENT.

C ZINT THIS FUNCTION COMPUTES THE DEPTH OF THE SEGMENT

C BEING LOOKED AT, GIVEN AN XS COORDINATE AS ARGUMENT.

C*****

SUBROUTINE LOOKER

INCLUDE 'MAIN.PAR'

INCLUDE 'MAIN.CMN'

INCLUDE 'LOOKBX.CMN'

INCLUDE 'SPAN.CMN'

SXLEFT = XLEFT(SEG)

SZLEFT = ZLEFT(SEG)

SXRGHT = XRIGHT(SEG)

SZRGHT = ZRIGHT(SEG)

SFULL = -1

IF(SXLEFT .GT. SPANLT) GO TO 10

SZLEFT = ZINT(SPANLT)

SXLEFT = SPANLT

GO TO 15

10 SFULL = 0

15 IF(SXRGHT .LT. SPANRT) GO TO 20

SZRGHT = ZINT(SPANRT)

SXRGHT = SPANRT

```

GO TO 21
20 SFULL = 0
21 IF(SXLEFT .LE. SPANLT) GO TO 22
SDIV = SXLEFT
GO TO 23
22 SDIV = SXRGHT
23 IF(BOXCNT .EQ. 0) GO TO 1000
IF(BOXCNT .EQ. 1) GO TO 25
IF((SXLEFT .LE. BXLEFT).AND.(SXRGHT.GE.BXRGHT)
1 .AND.(SZLEFT.LE.BZMIN).AND.(SZRGHT.LE.BZMIN))
1 GO TO 1000
IF((BXLEFT.LE.SXLEFT).AND.(BXRGHT.GE.SXRGHT)
1 .AND.(BZMAX.LE.SZLEFT).AND.(BZMAX.LE.SZRGHT))
1 RETURN
C*****
CALL XPANBX
C*****
RETURN
25 IF((BXLEFT.LE.SXLEFT).AND.(BXRGHT.GE.SXRGHT)
1 .AND.(BZINT(SXLEFT).LE.SZLEFT).AND.
1 (BZINT(SXRGHT).LE.SZRGHT))RETURN
IF((SXLEFT.LE.BXLEFT).AND.(SXRGHT.GE.BXRGHT)
1 .AND.(ZINT(BXLEFT).LE.BZLEFT).AND.
1 (ZINT(BXRGHT).LE.BZRGHT))GO TO 1000
IF((SFULL .NE. 0) .AND. (BFULL .NE. 0)) GO TO 29
C*****
CALL XPANBX
C*****
RETURN
29 TEMP = BXLEFT + (BXRGHT - BXLEFT) *
1 (SZLEFT - BZLEFT) / (BZRGHT - BZLEFT
1 - SZRGHT + SZLEFT)
C*****
CALL XPANBX
C*****
BOXTYP = 1
DIV = TEMP
IF(BZLEFT .LT. SZLEFT)CALL SWAP(BSEG1,BSEG2)
RETURN
1000 CALL LOADBX
9999 RETURN
END

```

```
C MAIN CONTROL PRROGRAM
  INCLUDE 'MAIN.PAR'
  INCLUDE 'INPUT.CMN'
  WRITE(5,9)
9  FORMAT(2X,' OPTIONS: '//,3X,'1-4027',//,3X,'2-4013/16',//,3X,
  1'3-RAMTEC',//)
  READ(5,*)IDEV
  GOTO(4,6,8),IDEV
8  WRITE(5,19)
19 FORMAT(1X,' SWITCH IS NOT AVILABLE ')
  STOP
4  XRES=IXRES1
  YRES=IYRES1
  GO TO 10
6  XRES=IXRES2
  YRES=IYRES2
10 CALL READIN
  CALL CONT
  STOP
  END
```



```
C      MISCELLANEOUS PROCEDURE.
C*****
C SUBROUTINE PIXSRT
C THIS SUBROUTINE PUT THE GIVEN SEGMENT AT THE
C HEAD OF THE XSORT LIST.
C*****
```

```
      SUBROUTINE PIXSRT(SEG)
      INCLUDE 'MAIN.PAR'
      INCLUDE 'MAIN.CMN'
      INCLUDE 'SRT.CMN'
      INTEGER SEG
      IF(SEGFST .EQ. 0) GO TO 20
      XSRTL(T(SEGFST) = SEG
20     XSRTL(SEG) = 0
      XSRTRT(SEG) = SEGFST
      SEGFST = SEG
      RETURN
      END
```

```
C PROC USED BY RECORDSAMPLE.  
C*****  
C MODULE PUTSAMPLE  
C FILENAME SUB12.FOR  
C SUBROUTINE NAME PSMPLE  
C  
C
```

```
          SUBROUTINE PSMPLE(X)  
          INCLUDE 'MAIN.PAR'  
          INCLUDE 'MAIN.CMN'  
          INCLUDE 'SAMSIN.CMN'  
          I = SAMFRE  
          SAMFRE = SAMLNK(I)  
          IF(SAMLST .EQ. 0) GO TO 10  
          SAMLNK(SAMLST) = I  
          GO TO 15  
10         SAMFST = I  
15         SAMLST = I  
          SAMX(I) = X  
          RETURN  
          END
```

```

SUBROUTINE READ2
C MAIN READIN ROUTINE
  INCLUDE 'MAIN.PAR'
  INCLUDE 'INPUT.CMN'
  INTEGER PLYPTR(MAXPLY), PLYEDG(MAXPLY)
  WRITE(5,988)
988  FORMAT(1X,'INPUT SIZE: ')
  READ(5,*)SIZE
C CLEAR P1, P2
  DO 10 I=1,MAXEDG
  P1(I) = 0
  P2(I) = 0
10  CONTINUE
  READ(22,1000) NPTS, NEDGE, NPOLY
1000 FORMAT (3I4)
  IF (NPTS.GT.MAXPNT.OR.NEDGE.GT.MAXEDG.OR.
  1  NPOLY .GT. MAXPLY) GO TO 888
C READ VERTICES
C
  WRITE(5,97)
97  FORMAT(1X,' X,Y,Z ')
  DO 20 I = 1,NPTS
  READ (22,2000) XS(I),YS(I),ZS(I)
2000 FORMAT (3F4.0)
  WRITE(5,*) XS(I),YS(I),ZS(I)
C
C ADJUST SCREEN SIZE
C
  XS(I) = XS(I)/SIZE
  YS(I) = YS(I)/SIZE
20  CONTINUE
  EDGLST = NEDGE
  WRITE(5,876)
876  FORMAT(1X,' NEXT POSITION')
  DO 30 I = 1,NEDGE
  ENTLST (I) = I-1
  READ (22,1000) V1(I),V2(I),LINKED(I)
  WRITE(5,*)V1(I),V2(I),LINKED(I)
C LINKED NOT USED HERE BUT PUT TO BE COMPATABLE WITH CADCOM INPUT
30  CONTINUE
C FORMAT DOES NOT AGREE WITH BRAKE, TAKEN EXACTLY FROM WATKINS
C ON TEXT -- SIMPLE PROGRAM TO GENERATE THIS FORMAT FROM BRAKE
C ONCE FINAL FORMAT IS ESTABLISHED.
  DO 50 I=1,NPOLY
C PLYPTR NOT USED FOR WATKINS, PLYEDG NOT NEEDED TO RETAIN
C
  READ (22,1000) PLYPTR (I),SHAD(I),PLYEDG(I)
  J=PLYEDG(I)
  DO 40 L=1,J
C
C GET EDGE NUMBER FOR EDGE
C DETERMINE WHAT POLYGON/S BORDER EACH EDGE
  READ(22,1000) K
  IF(P1(K).EQ.0) GO TO 35
  P2(K) =I
  GO TO 40
35  P1(K) = I
40  CONTINUE
50  CONTINUE
C TO INITIALIZE SCREEN
C CALL INITSR

```

```

77      CONTINUE
        WRITE(5,7000)
7000    FORMAT (' !WOR 30''/'' !GRA 1,30''/
1        ' !ERA G')
        WRITE(5,931)
931     FORMAT(1X,' HERE ARE THE SHADES')
        DO 90 I=1,NPOLY
        WRITE (5,4000) SHAD(I)
        WRITE(5,*)SHAD(I)
4000    FORMAT (' !COL C',I1)
C ASSUME REPEATED EDGES
        K=PLYPTR(I)
        KK=K+PLYEDG(I)-1
        DO 80 J=K,KK
        IXS=XS(V1(J))
        IYS=YS(V1(J))
        IXS2=XS(V2(J))
        IYS2=YS(V2(J))
        WRITE (5,5000)IXS,IYS,IXS2,IYS2
5000    FORMAT (' !VEC',4I4)
80      CONTINUE
90      CONTINUE
C       CALL CONT
        GO TO 999
888     WRITE (5,8000)
8000    FORMAT (' TOO MUCH DATA')
999     RETURN
        END

```

```

SUBROUTINE READIN
C MAIN READIN ROUTINE
  INCLUDE 'MAIN.PAR'
  INCLUDE 'INPUT.CMN'
  INTEGER PLYPTR(MAXPLY), PLYEDG(MAXPLY)
  WRITE(5,988)
988  FORMAT(1X,'INPUT SIZE: ')
  READ(5,*)SIZE
C CLEAR P1, P2
  DO 10 I=1,MAXEDG
  P1(I) = 0
  P2(I) = 0
10  CONTINUE
  READ(22,1000) NPTS, NEDGE, NPOLY
1000 FORMAT (3I4)
  IF (NPTS.GT.MAXPNT.OR.NEDGE.GT.MAXEDG.OR.
  1  NPOLY .GT. MAXPLY) GO TO 888
C READ VERTICES
C
  DO 20 I = 1,NPTS
  READ (22,2000) XS(I),YS(I),ZS(I)
2000 FORMAT (3F4.0)
C
C ADJUST SCREEN SIZE
C
  XS(I) = XS(I)/SIZE
  YS(I) = YS(I)/SIZE
20  CONTINUE
  EDGLST = NEDGE
  DO 30 I = 1,NEDGE
  ENTLST (I) = I-1
  READ (22,1000) V1(I),V2(I),LINKED(I)
C LINKED NOT USED HERE BUT PUT TO BE COMPATABLE WITH CADCOM INPUT
30  CONTINUE
C FORMAT DOES NOT AGREE WITH BRAKE, TAKEN EXACTLY FROM WATKINS
C ON TEXT -- SIMPLE PROGRAM TO GENERATE THIS FORMAT FROM BRAKE
C ONCE FINAL FORMAT IS ESTABLISHED.
  DO 50 I=1,NPOLY
C PLYPTR NOT USED FOR WATKINS, PLYEDG NOT NEEDED TO RETAIN
C
  READ (22,1000) PLYPTR (I),SHAD(I),PLYEDG(I)
  J=PLYEDG(I)
  DO 40 L=1,J
C
C GET EDGE NUMBER FOR EDGE
C DETERMINE WHAT POLYGON/S BORDER EACH EDGE
  READ(22,1000) K
  IF(P1(K).EQ.0) GO TO 35
  P2(K) =I
  GO TO 40
35  P1(K) = I
40  CONTINUE
50  CONTINUE
C TO INITIALIZE SCREEN
  CALL SHWINT
77  CONTINUE
  DO 90 I=1,NPOLY
  GOTO(100,110,120),IDEV
100  WRITE (5,4000) SHAD(I)
4000 FORMAT (' !COL C',I1)
C ASSUME REPEATED EDGES

```

```
110      K=PLYPTR(I)
        KK=K+PLYEDG(I)-1
        DO 80 J=K, KK
        IXS=XS(V1(J))
        IYS=YS(V1(J))
        IXS2=XS(V2(J))
        IYS2=YS(V2(J))
        IF(IDEV .EQ. 1)GO TO 88
        CALL MOVABS(IXS, IYS)
5000     FORMAT ( ' !VEC', 4I4)
        CALL DRWABS(IXS2, IYS2)
        GO TO 80
88      WRITE(5, 5000)IXS, IYS, IXS2, IYS2
80      CONTINUE
90      CONTINUE
C       CALL CONT
        GO TO 999
888     WRITE (5, 8000)
8000    FORMAT ( ' TOO MUCH DATA')
        GO TO 999
120     WRITE(5, 130)
130     FORMAT(1X, ' SWITCH IS NOT AVILABLE')
999     RETURN
        END
```

```

SUBROUTINE RECSAM(SEG,LEFT,RIGHT)
INCLUDE 'MAIN.PAR'
INCLUDE 'MAIN.CMN'
INCLUDE 'SPAN.CMN'
INCLUDE 'SAMSIN.CMN'
INTEGER SEG,RIGHT
REAL LSAMP
IF(.NOT.((LEFT.NE.0).AND.(IMPLFT.NE.0).AND.(XLEFT(SEG).NE.0)
1 .LE.(SPANLT.NE.0).AND.(SEG.NE.0).EQ.MOD(PLYSEG(IMPLFT)
1 ,10000))) GO TO 10
A = XLEFT(IMPLFT) + DXLEFT(IMPLFT)
CALL PSMPLE(A)
IMPLFT = 0
10 IF(.NOT.((LEFT.NE.0).AND.(YLEFT(SEG).NE.0).LT.-1
1 .AND.(LEFT.EQ.-1.OR.(SPANLT-1.LT.XLEFT(SEG).AND.
1 XLEFT(SEG).LE.SPANLT)))) GO TO 20
IF(.NOT.(SAMLST.EQ.0.OR.LSAMP.NE.SPANLT-1.OR.
1 LEFT.EQ.-1)) GO TO 20
B = (XLEFT(SEG) + DXLEFT(SEG))
CALL PSMPLE(B)
LSAMP = SPANLT-1
20 IF(.NOT.((RIGHT.NE.0).AND.YRIGHT(SEG).LT.-1
1 .AND.(SPANRT.LE.XRIGHT(SEG).AND.XRIGHT(SEG).LT.
1 SPANRT+1))) GO TO 30
C = XRIGHT(SEG) + DXRGHT(SEG)
CALL PSMPLE(C)
LSAMP = SPANRT
30 CONTINUE
RETURN
END

```

```
C          MISCELLANEOUS PROCEDURE.  
C*****  
C SUBROUTINE NAME RETBLK.  
C THIS SUBROUTINE IS USED TO RETURN  
C A SEGMENT BLOCK TO FREE STORAGE.  
C*****
```

```
      SUBROUTINE RETBLK(I)  
      INCLUDE 'MAIN.PAR'  
      INCLUDE 'MAIN.CMN'  
      INCLUDE 'BLK.CMN'  
      ACTIVE(I) = FRELST  
      FRELST = I  
      RETURN  
      END
```



```
C      MISCELLANEOUS PROCEDURE
C*****
C SUBROUTINE NAME RMXSRT
C THIS SUBROUTINE REMOVES A SEGMENT FROM THE
C XSORT LIST.
C*****
```

```
      SUBROUTINE RMXSRT(SEG)
      INCLUDE 'MAIN.PAR'
      INCLUDE 'MAIN.CMN'
      INCLUDE 'SRT.CMN'
      INTEGER SEG
      IF(SEGFST .NE. SEG) GO TO 10
      SEGFST = XSRTRT(SEG)
10     I = XSRTRT(SEG)
      IF( I .EQ. 0 ) GO TO 20
      XSRTL(I) = XSRTL(SEG)
20     I = XSRTL(SEG)
      IF(I .EQ. 0) RETURN
      XSRTRT(I) = XSRTRT(SEG)
      RETURN
      END
```

C DISPLAY SUBROUTINE

SUBROUTINE SHOW(Y)

C

```
INCLUDE 'MAIN.PAR'  
INCLUDE 'MAIN.CMN'  
INCLUDE 'INPUT.CMN'  
INCLUDE 'PIC.CMN'  
INTEGER X,SEG,POLYG,SAMP,Y  
DIMENSION IDASH(7),ICHECK(7),IREPT(7)  
DATA IDASH/1,2,3,4,5,5,2/  
DATA ICHECK/7*-1/  
DATA IREPT/10,30,30,30,1,1,20/  
DATA IFLAG/0/  
SAMP=0  
IF (SEGCNT.LE.1) GO TO 999  
DO 30 I=1,SEGCNT  
SEG = VISSEG(I)  
X = VISPOS(I)  
IF (SEG.EQ.0) GO TO 20  
POLYG = POLYGN(SEG)  
GOTO(4,66),IDEV  
66 ICOL=SHAD(POLYG)  
CALL PNTABS(SAMP,Y)  
CALL PNTABS(X,Y)  
IF(ICHECK(ICOL) .LT. 0)GO TO 5  
IDEL=ICHECK(ICOL)-SAMP  
IF(IDEL .EQ. 0)GO TO 6  
10 IF(MOD(ABS(IDEL),IREPT(ICOL)) .EQ. 0)GO TO 6  
SAMP=SAMP+1  
IF(SAMP .GE. X)GO TO 6  
IDEL=ICHECK(ICOL)-SAMP  
GO TO 10  
5 ICHECK(ICOL)=SAMP  
GOTO 6  
4 WRITE(5,100) SHAD(POLYG)  
100 FORMAT(' !COL C',I1)  
WRITE(5,200) SAMP,Y,X,Y  
200 FORMAT(' !VEC ',4I4)  
GO TO 20  
6 IF(ICOL .NE. 6)GO TO 91  
IFLAG=IFLAG+1  
IF(MOD(IFLAG,5) .EQ. 0)GO TO 91  
GO TO 20  
91 CALL MOVABS(SAMP,Y)  
CALL DSHABS(X,Y,IDASH(SHAD(POLYG)))  
20 SAMP=X  
30 CONTINUE  
GO TO 999  
888 WRITE(5,300)  
300 FORMAT(' SEGCNT =0,NO OUTPUT')  
999 RETURN  
END
```

C
C

```
SUBROUTINE SHWCLS  
CALL FINITT(0,0)  
RETURN  
END
```

C INITIALIZE SCREEN

```
SUBROUTINE SHWINT  
INCLUDE 'MAIN.PAR'  
INCLUDE 'INPUT.CMN'  
GOTO(10,20,30),IDEV  
10 WRITE(5,7000)  
7000 FORMAT(1X,'!WOR 30'/' !GRA 1,30'/  
1 ' !ERA G')  
RETURN  
20 CALL INITT(120)  
RETURN  
30 WRITE(5,8000)  
8000 FORMAT(1X,' SWITCH IS NOT AVILABLE')  
RETURN  
END
```

```
SUBROUTINE STOPIC(X,SEGMENT)  
INCLUDE 'MAIN.PAR'  
INCLUDE 'MAIN.CMN'  
INCLUDE 'PIC.CMN'  
INTEGER SEGMENT  
IF((SEGMENT .EQ. 0).OR.(SEGMENT.NE.LASSEG))GO TO 10  
GO TO 15  
10 SEGMENT = SEGMENT + 1  
LASSEG = SEGMENT  
15 VISPOS(SEGMENT) = X  
VISSEG(SEGMENT) = LASSEG  
RETURN  
END
```

```
C*****  
C FILE NAME "SUB.FOR".  
C SUBROUTINE NAME 'SWAP'.  
C THIS SUBROUTINE IS DESIGNED TO INTERCHANGE THE  
C CONTENTS OF TWO VARIABLES.  
C*****
```

```
SUBROUTINE SWAP(X1,X2)  
TEMP = X1  
X1 = X2  
X2 = TEMP  
RETURN  
END
```

```

SUBROUTINE THINK(ITH)
INCLUDE 'MAIN.PAR'
INCLUDE 'MAIN.CMN'
INCLUDE 'LOOKBX.CMN'
INCLUDE 'THK.CMN'
INCLUDE 'SPAN.CMN'
INCLUDE 'SAMSIN.CMN'
INTEGER SEGSAM
INTEGER GETBLK
INTEGER XRES
XRES = IXRES
C--BEGIN THINKER
  IF(BOXCNT .NE. 0) GO TO 10
C--BEGIN NOTHING VISIBLE
  CALL STOPIC(SPANRT,0)
  ITH = -1
  RETURN
C*****
10  IF(BOXCNT .NE. 1) GO TO 20
C--BEGIN ONLY ONE SEGMENT, DISPLAY DIRECTLY
  IF(BXLEFT .NE. SPANLT) CALL STOPIC(BXLEFT,0)
  CALL STOPIC(BXRGT,BSEG1)
  IF(BXRGT .NE. SPANRT) CALL STOPIC(SPANRT,0)
  CALL RECSAM(BSEG1,1,1)
  ITH = -1
  RETURN
C*****
20  IF(BOXTYP .NE. 1) GO TO 110
C--BEGIN INTERSECTING PLANES CASE
  CALL STOPIC(DIV,BSEG1)
  CALL RECSAM(BSEG1,1,0)
  SEGSAM = BSEG1 * 10000 + BSEG2
  SEG = IMPLST
  PREV = 0
30  IF(SEG .EQ. 0) GO TO 40
  IF(SEGSAM .EQ. PLYSEG(SEG)) GO TO 40
  PREV = SEG
  SEG = XSRTRT(SEG)
  GO TO 30
40  IF(SEG .EQ. 0) GO TO 90
C--BEGIN FOUND A PREVIOUS ONE
  IF(PREV .EQ. 0) GO TO 50
  XSRTRT(PREV) = XSRTRT(SEG)
  GO TO 60
50  CONTINUE
  IMPLST = XSRTRT(SEG)
60  CONTINUE
  DXLEFT(SEG) = DIV - XLEFT(SEG)
  XLEFT(SEG) = DIV
  IF(.NOT.(1.LE.XLEFT(SEG)+DXLEFT(SEG).AND.
  1 XLEFT(SEG)+DXLEFT(SEG).LE.XRES)) GO TO 70
C--BEGIN IMPLIED EDGE WILL BE WITHIN BOUNDS ON NEXT SCANLINE
  CALL PIXSRT(SEG)
  CALL RECSAM(SEG,-1,0)
  GO TO 80
70  CALL RETBLK(SEG)
80  GO TO 100
C--BEGIN DETECTED NEW IMPLIED EDGE
90  J = GETBLK(J )
  PLYSEG(J) = SEGSAM
  I = YLEFT(BSEG1)

```

```

        IF(I .LT. YRIGHT(BSEG1)) I = YRIGHT(BSEG1)
        IF(I .LT. YLEFT(BSEG2)) I = YLEFT(BSEG2)
        IF(I .LT. YRIGHT(BSEG2)) I = YRIGHT(BSEG2)
        YLEFT(J) = I
        POLYGN(J) = 0
        XLEFT(J) = DIV
        XSRTRT(J) = MPLST2
        MPLST2 = J
100     CALL STOPIC(SPANRT,BSEG2)
        CALL RECSAM(BSEG2,0,1)
        ITH = -1
        RETURN
C*****
110     IF(SPANLT .NE. SPANRT) GO TO 120
C--BEGIN MUST NOT SUBDIVIDE FURTHER
        ITH = -1
        RETURN
C*****
120     ITH = 0
        RETURN
        END

```

```

C      MODULE USED BY LOOKER.
C*****
C FILE NAME 'SUB3.FOR'
C SUBROUTINE NAME 'XPANBX'.
C THIS ROUTINE ADDS THE PRESENT SEGMENT TO THE BOX
C IF NECESSARY, THE EXTREMITIES OF THE BOX ARE
C EXPANDED TO ENCLOSE THE NEW SEGMENT.
C*****

```

```

      SUBROUTINE XPANBX
      INCLUDE 'MAIN.PAR'
      INCLUDE 'LOOKBX.CMN'
      BSEG2 = BSEG1
      BSEG1 = SEG
      BOXTYP = 0
      BOXCNT = BOXCNT + 1
      IF(SDIV .LT. DIV)DIV= SDIV
      IF(SXLEFT .LT.BXLEFT) BXLEFT= SXLEFT
      IF(SXRGHT .GT. BXRGT)BXRGT = SXRGHT
      IF(SZLEFT .LT. BZMIN) BZMIN = SZLEFT
      IF(SZRGHT .LT.BZMIN) BZMIN = SZRGHT
      IF(SZLEFT .GT.BZMAX) BZMAX = SZLEFT
      IF(SZRGHT .GT.BZMAX) BZMAX = SZRGHT
      RETURN
      END

```

```

C      MODULE USED BY LOOKER.
C*****
C FILE NAME 'SUB5.FOR'
C FUNCTION NAME 'ZINT'.
C THIS FUNCTION COMPUTES THE DEPTH OF THE SEGMENT
C BEING LOOKED AT, GIVEN AN XS COORDINATE AS ARGUMENT.
C*****

```

```

      FUNCTION ZINT(X)
      INCLUDE 'MAIN.PAR'
      INCLUDE 'LOOKBX.CMN'
      IF(SXRGHT.EQ.SXLEFT) GO TO 10
      ZINT=SZLEFT+(SZRGHT-SZLEFT)*(X-SXLEFT)
1      /((SXRGHT-SXLEFT)
      RETURN
10     ZINT = SZLEFT
      RETURN
      END

```

BIBLIOGRAPHY

Booth, K.S. (ed. and comp.): Tutorial: Computer Graphics, The Institute of Electrical and Electronic Engineers, Inc., New York, N.Y., 1979.

Codd, E.F. : "A Relational Model of Data for Large Data Banks," Communications of the ACM, Vol.13, No.6, 1970, pp.377-387.

Codd, E.F.: "Further Normalization of the Data Base Relational Model," Data Base Systems, Courant Computer Science Symposia Series, Vol.6, Prentice Hall, Englewood Cliffs, N.J., 1972.

Codd, E.f.: "Relational Completeness of Data Base Sublanguages," Data Base Systems, Courant Computer Science Symposia Series, Vol.6, Prentice Hall, Englewood Cliffs, N.J., 1972.

Date, C.j.: An Introduction to Database Systems, Addison-Wesley, 1977.

Comer, D.: "The Ubiquitous B-Tree," ACM Computing Surveys, Vol. 11, No. 2, June 1979.

Eastman, C. and Henrion, M.: Glide-A Language for Design Information Systems, Vol.2, Summer 1977.

Freeman, H. (ed. and comp.): Tutorial and Selected Readings in Interactive Computer Graphics, The Institute of Electrical and Electronic Engineers, Inc., New York, N.Y., 1980.

Gries, D.E.: Compiler Construction for Digital Computers, John Wiley and Sons, Inc., New York, 1971.

Hamlin, G. and Gear, C.W.: "Raster-scan Hidden Surface Algorithm Techniques," Computer Graphics, Vol.2, No.2, Summer 1977.

Harms, E. and Zabinski, M.P.: Introduction to APL and Computer Programming, John Wiley and Sons, Inc., New York, 1977.

Kim, Won: "Relational Database Systems," ACM Computing Surveys, Vol.11, No. 3, September 1979.

Klinger, A.; Fu, K.S.; and Kunii, T.L. (ed. and comp.): Data structures, Computer Graphics, and Pattern Recognition, Academic Press, 1977.

Knuth, D.E.:The Art of Computer Programming Vol. 3: Sorting and Searching, Addison-Wesley Publishing Co., Reading, Mass., 197, pp473-480.

Levy, L.:Discrete Structures of Computer Science, John Wiley and Sons, Inc., New York, 1980.

Lorie, R.A. and Symonds, A.J.: "A Relational Access Method for interactive Applications, Data Base Systems, Courant Computer SCIENCE Symposia Series, Vol.6, Prentice-Hall, Englewood Cliffs, N.J., 1972.

Meissner, L.P. and Organick, E.: "FORTRAN 77 - Featuring Structured Programming," [Appendix I Summary of ANSI Standard X39-1978 (FORTRAN 77)]. Addison-Wesley, 1980.

Newell, M.E.; Newell, R.g; and Sancha, T.L.: "A Solution to the Hidden Surface Problem, Proceedings, ACM National Meetings, 1972.

Newman, W.M. and Sproull, R.:Principles of Interactive Computer Graphics, First Edition, McGraw-Hill, 1973.

Sutherland, I.E., Sproull, R.F., and Schumacker, R.A.: "Sorting and the Hiddenen-surface Problem, Proceeding National Computer Conference 1973, AFIPS Press, pp685-693.

Tremblay, J.P. and Sorenson, P.G.:An Introduction to Data Structures with Applications, McGraw-Hill, Inc., U.S.A., 1976, pp680-4.

Weller, D. and Williams, R.: "Graphic and Relational Data Base Support for Problem Solving, Computer Graphics, Vol. 10, No.2, Summer 1976.

Wimble, M.: "An APL Interpreter for Microcomputers: Part 1," Byte, August 1977 [Part 2, September 1977, Part 3, October 1977].