A Midpoint Clipping Algorithm

An Honors Thesis (ID 499)

by

Kristine E. Boerger

Thesis Director

Dr. Milton M. Underkoffler

Ball State University

Muncie, Indiana

May, 1974

This project has been accepted in fulfillment of the requirements for ID499 Senior Honors Project.

May 7, 1974
Date

Milton M. Underhoffer
Advisor

TABLE OF CONTENTS

A MIDPOINT CLIPPING ALGORITHM

INTRODUCTION

The ability to interact with the computer and to dynamically change pictures displayed on a screen has greatly increased the potential of the computer. With the aid of a graphical display device, the computer can be used to design a variety of objects, from buildings to political boundaries. It can be used to gain insight into the structure of molecules and the inner workings of the computer itself. The picture, or design, can be scaled, rotated, and transformed to such an extent that it can be viewed from any angle or perspective. Problems can arise when manipulating the picture in this manner, though. The picture, or parts of it, may be moved to a position which is off the screen.

There are several different ways in which computers cope with this problem. In one method, called "scissoring",[1] only the portion which is on the screen is intensified. However, the computer continues to trace the entire picture, including the invisible portion. This wastes time and could result in flickering even though the visible portion is very simple. A more efficient method is to "clip"[2] the picture.

---

[1] William M. Newman and Robert F. Sproull, Principles of Interactive Computer Graphics, McGraw-Hill, New York, 1973, p. 121.

[2] Newman and Sproull, p. 121.

The clipping algorithm traces and intensifies only the visible portion of the picture, thus saving time and reducing the chances of flickering. One such algorithm, called the midpoint clipping algorithm, performs a logarithmic search for the endpoints of the visible segment by continually subdiving the line segments at their midpoints.

The program was written in PL/1. Its method of handling IF statements and the flexibility offered through DO groups and built-in functions made PL/1 the most suitable language to use. The main concern was determining when to stop the process, that is, determining when the visible segment had been found. This should be when the midpoint coincided with an edge of the screen, but this posed another problem. Since distances on the display graph were measured in integers, the midpoints needed to be integers also. However, the danger existed that rounding would distort the line so much that the visible segment thus determined would bear little relationship to the original line. Because of this, floating-point arithmetic was used for all calculations, and the rounded integer portion for all comparisons. A special rounding function had to be written for this purpose as the PL/1 built-in function, ROUND, did not accept floating-point arguments.

USING THE ALGORITHM

CLIP1

Calling this subroutine initializes the coordinates of the screen edges. CLIP1 expects the origin, (0,0), to be at the center of the screen.

Calling Sequence:

```
CALL CLIP1(XR,YT)
```

- XR is the x-coordinate of the right-hand edge of the screen.
- YT is the y-coordinate of the top edge of the screen.

CLIP

This subroutine clips the given line to the dimensions of the screen, as initialized by CLIP1.

Calling Sequence:

```
CALL CLIP(X1,Y1,X2,Y2)
```

- X1, Y1 are the x- and y-coordinates of one endpoint of the line to be clipped.
- X2, Y2 are the x- and y-coordinates of the other endpoint.

NARRATIVES AND FLOWCHARTS

CLIP1

The first part of this subroutine, entered at CLIP1, initializes the values of the screen edges. The x-value of the right-hand edge, and the y-value of the top, are passed to CLIP1 by the calling procedure. The x-value of the left-hand edge of the screen is then set equal to the negative of the x-value of the right-hand edge. The y-value of the bottom is set equal to the negative of the top.

The logic for the second part of CLIP1, entered at CLIP, follows the narrative set forth in Newman and Sproull's Principles of Interactive Computer Graphics. *The space occupied by the unclipped picture is divided into nine regions by extending the edges of the screen (see figure 1). The endpoints of the line are assigned four-bit binary codes corresponding to the regions in which they lie. (For an explanation of these codes see the narrative for the function procedure CODE, page 10.) If the four-bit codes for both endpoints are zeroes, the line lies entirely within the boundaries of the screen. If the logical intersection of the two codes is not zero, the line must lie entirely off-screen, and is rejected.*

1001	1000	1010
0001	SCREEN 0000	0010
0101	0100	0110

fig. 1

If the line cannot be trivially accepted or rejected, it is subdivided at its midpoint. The algorithm is then applied to each of these segments, with the following restrictions:

1. One endpoint of the original line may have been visible (see figure 2, lines A and B). If so, one of its halves must either be entirely visible (line A), or else a trivial reject (line B). This half is discarded, and the algorithm is applied to the other half.

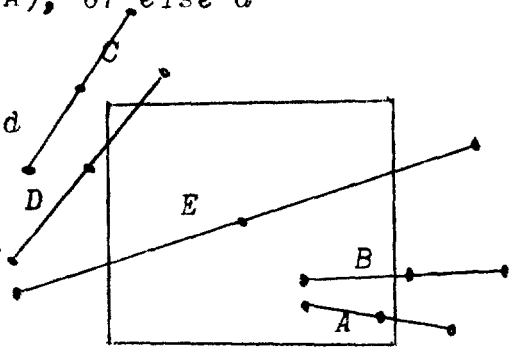


fig. 2

2. If both endpoints of the original line were invisible (lines C, D, and E of figure 2), each of the two halves is treated on its own merits. Both halves may be trivial rejects in which case the entire line is invisible (line C); only one segment may be trivially rejected (line D); or the midpoint may be visible, in which case neither half can be rejected (line E). In this last case the algorithm is applied to both halves.

The search stops when both halves of the line are rejected, or when the midpoint coincides with one of the edges of the screen.

Subroutine Procedures Called:

VINV - Internal, recursive.

DISPLN - External.

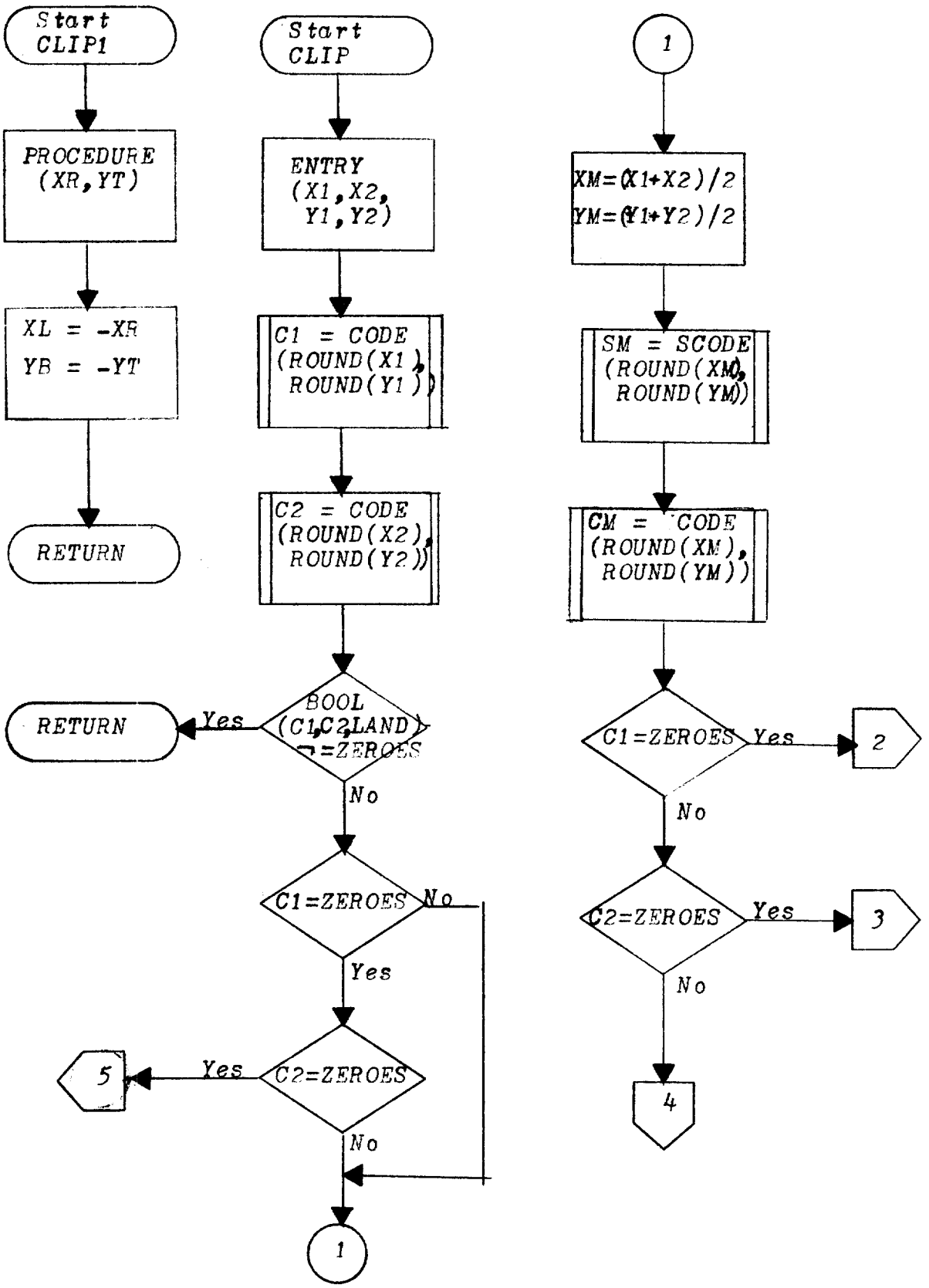
Function Procedures Called:

CODE - Internal.

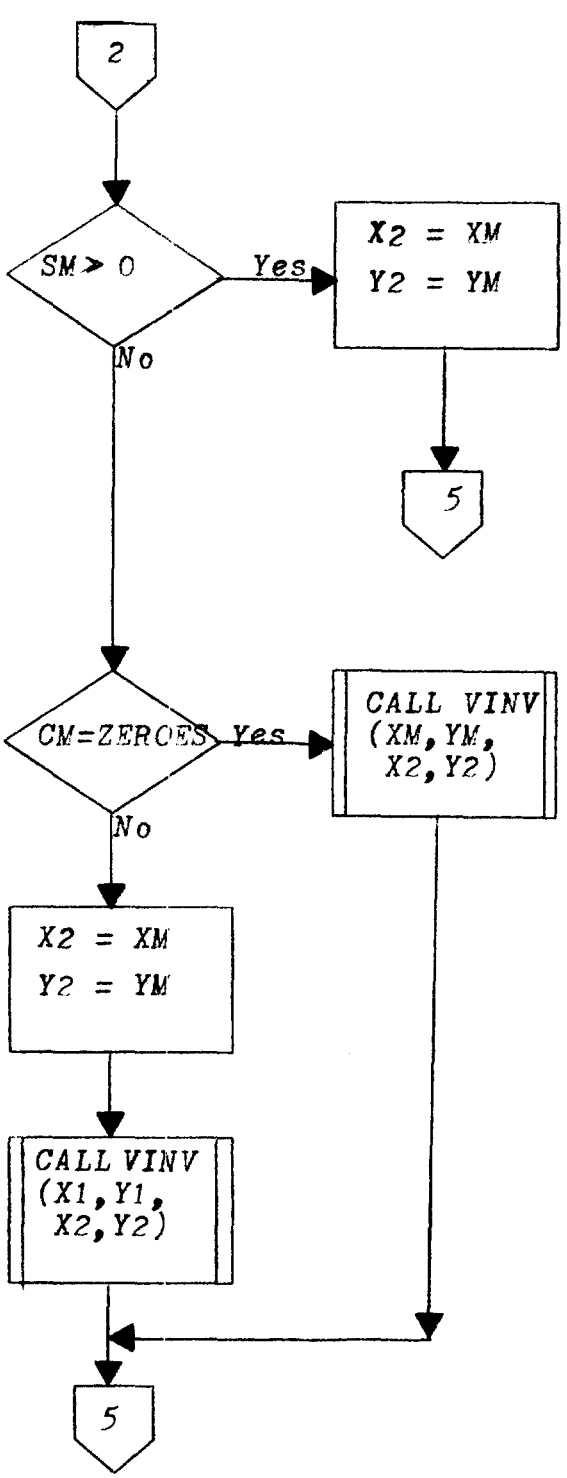
SCODE - Internal.

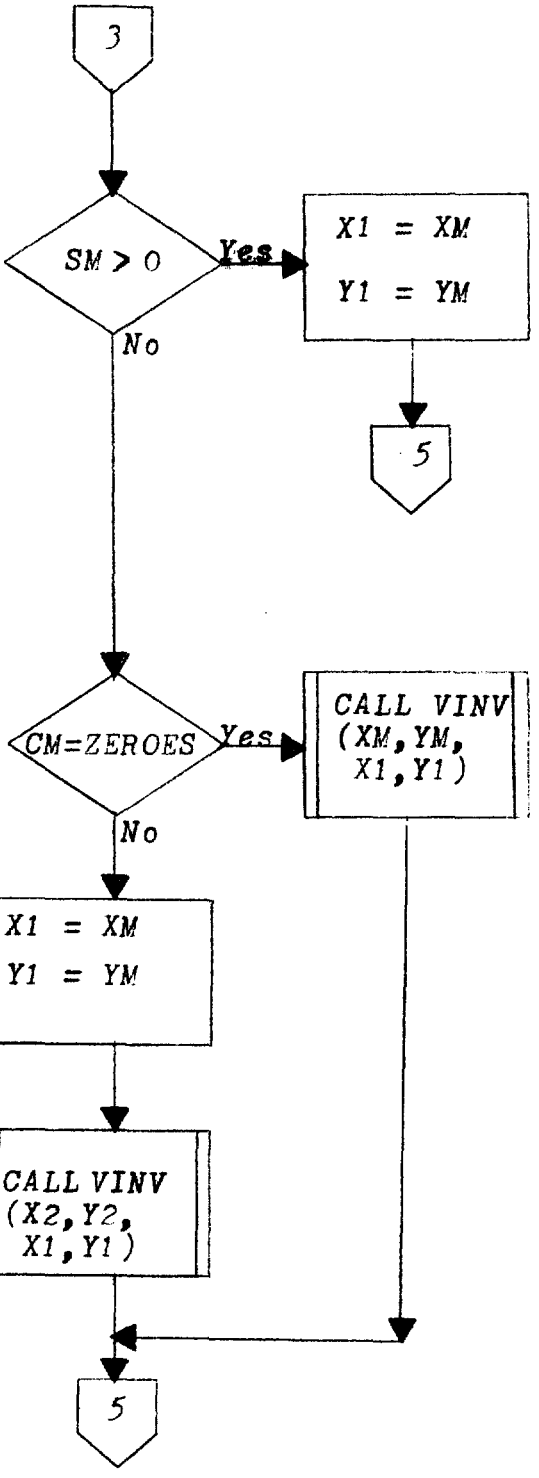
ROUND - Internal.

Flowchart:

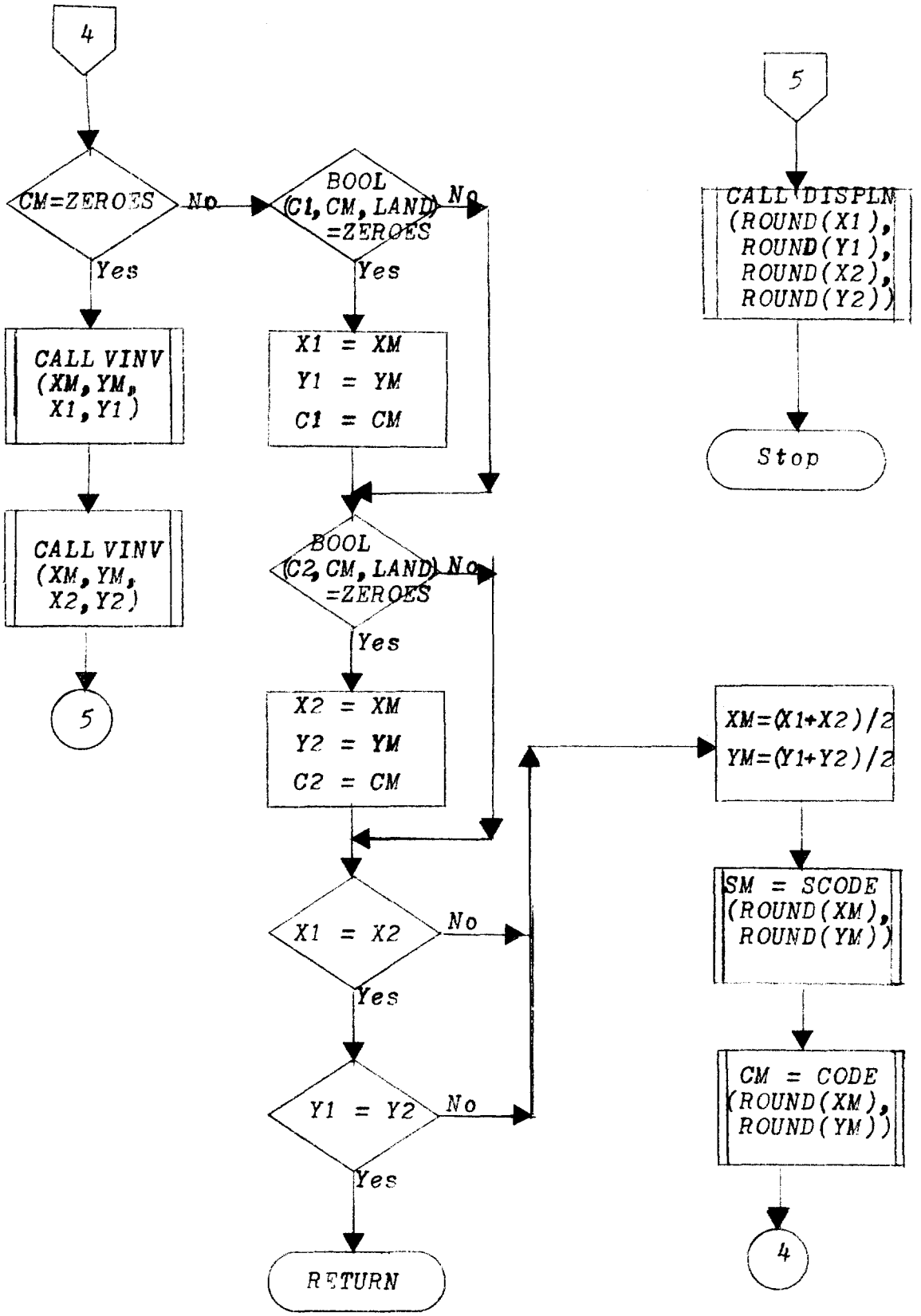

Start CLIP1
PROCEDURE (XR, YT)
XL = -XR
YB = -YT
RETURN
Start CLIP
ENTRY (X1, X2, Y1, Y2)
C1 = CODE (ROUND(X1), ROUND(Y1))
C2 = CODE (ROUND(X2), ROUND(Y2))
BOOL (C1, C2, AND) = ZEROES
Yes
RETURN
No
C1=ZEROES
No
Yes
C2=ZEROES
Yes
5
No
1
1
XM=(X1+X2)/2
YM=(Y1+Y2)/2
SM = SCODE (ROUND(XM), ROUND(YM))
CM = CODE (ROUND(XM), ROUND(YM))
C1=ZEROES
Yes
2
No
C2=ZEROES
Yes
3
No
4

Flowchart (cont.):





Flowchart (cont.):



ROUND

This function procedure rounds a floating-point number to the nearest integer. The number is first truncated at the decimal point. The two numbers are then subtracted. If the absolute value of the difference is less than .5 the truncated value is returned. If it is greater than or equal to .5, a positive or negative 1, depending on the sign of the original number, is added to the truncated value before returning to the calling procedure.

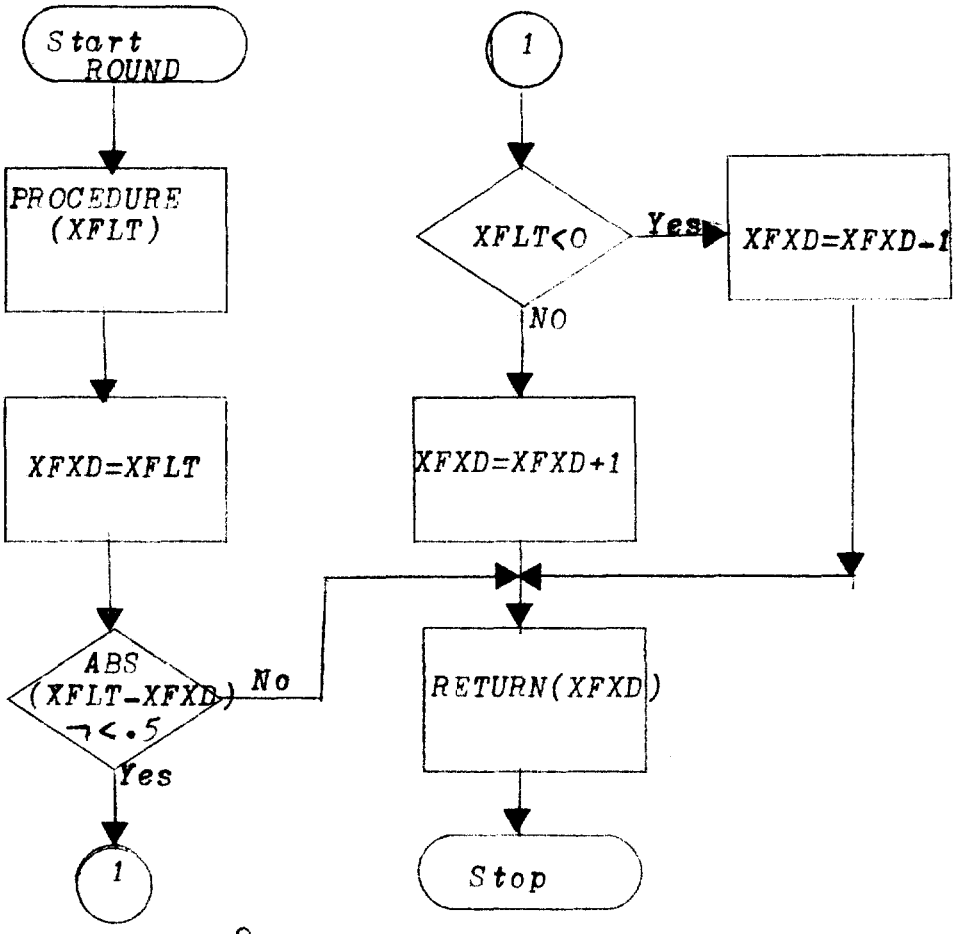
Subroutine Procedures Called:

None.

Function Procedures Called:

None.

Flowchart:



CODE

This function procedure recieves a point from the calling procedure and returns a four-bit code corresponding to the region in which the point lies. The four bits mean the following, if set:

first bit - point is above the top edge of the screen.

second bit - point is below the bottom edge.

third bit - point is to the right of the right-hand edge.

fourth bit - point is to the left of the left-hand edge.

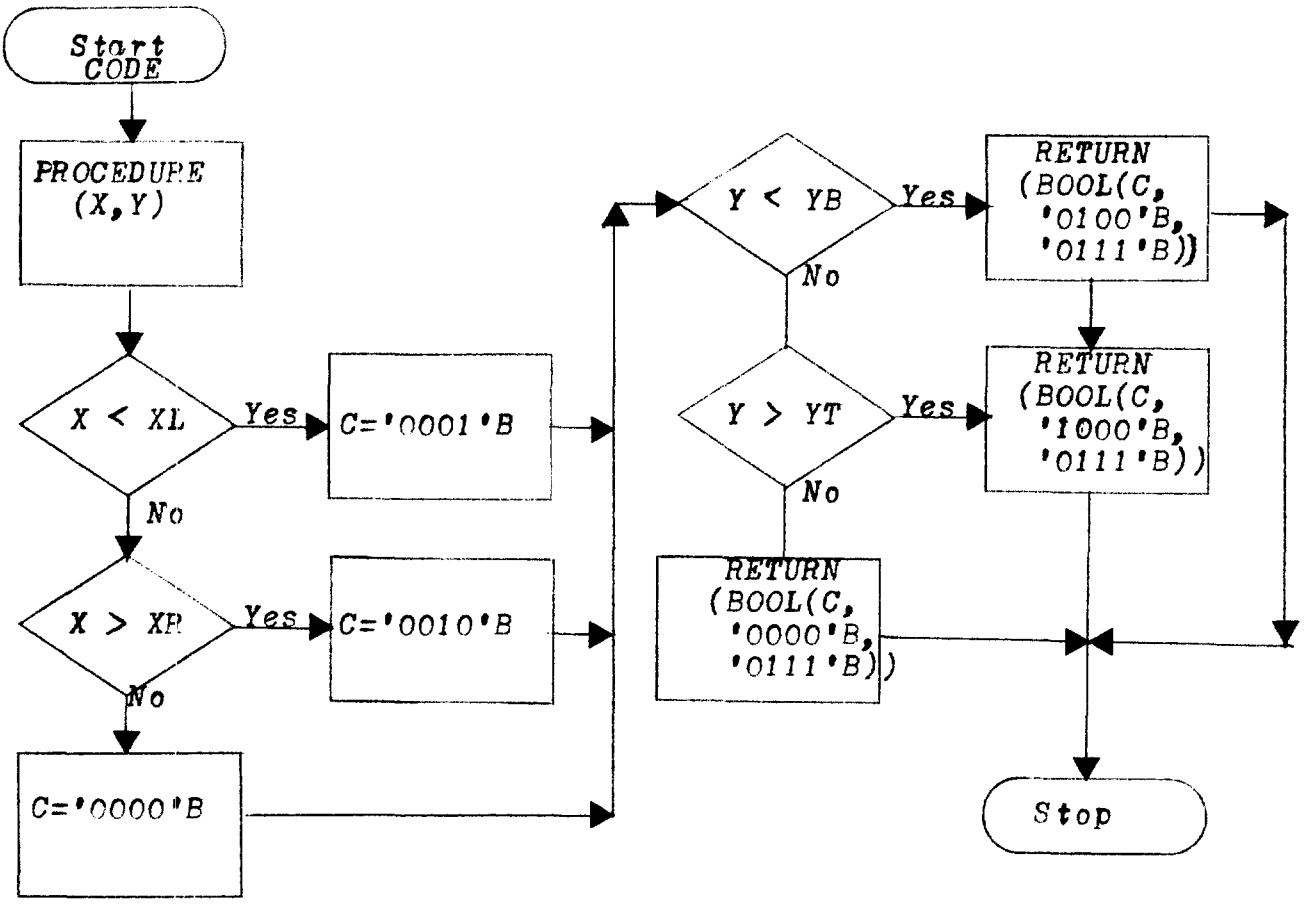
Subroutine Procedures Called:

None.

Function Procedures Called:

None.

Flowchart:



SCODE

This function procedure recieves a point from the calling procedure and returns one of the following codes:

0 - point does not lie on an edge of the screen.

1 - point lies on the top edge.

2 - point lies on the bottom edge.

3 - point lies on the right-hand edge.

4 - point lies on the left-hand edge.

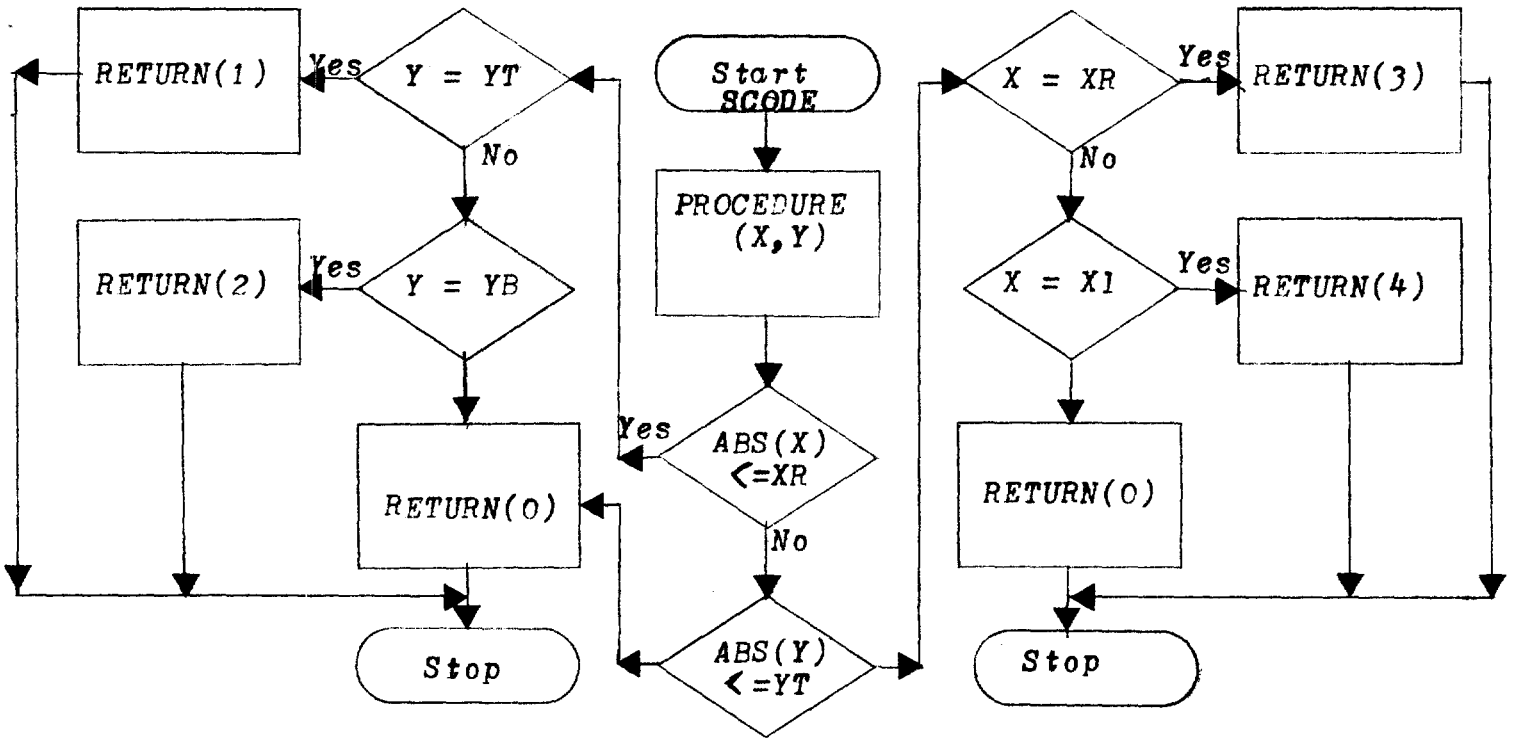
Subroutine Procedures Called:

None.

Function Procedures Called:

None.

Flowchart:



VINV

This recursive subroutine procedure finds the visible portion of a line segment, which has one visible endpoint, and one invisible one. The segment is divided at its midpoint. The half which then meets the criterion of one visible, and one invisible, endpoint is passed to VINV. This cycle is broken when the midpoint coincides with a screen edge.

Subroutine Procedures Called:

VINV - Recursive.

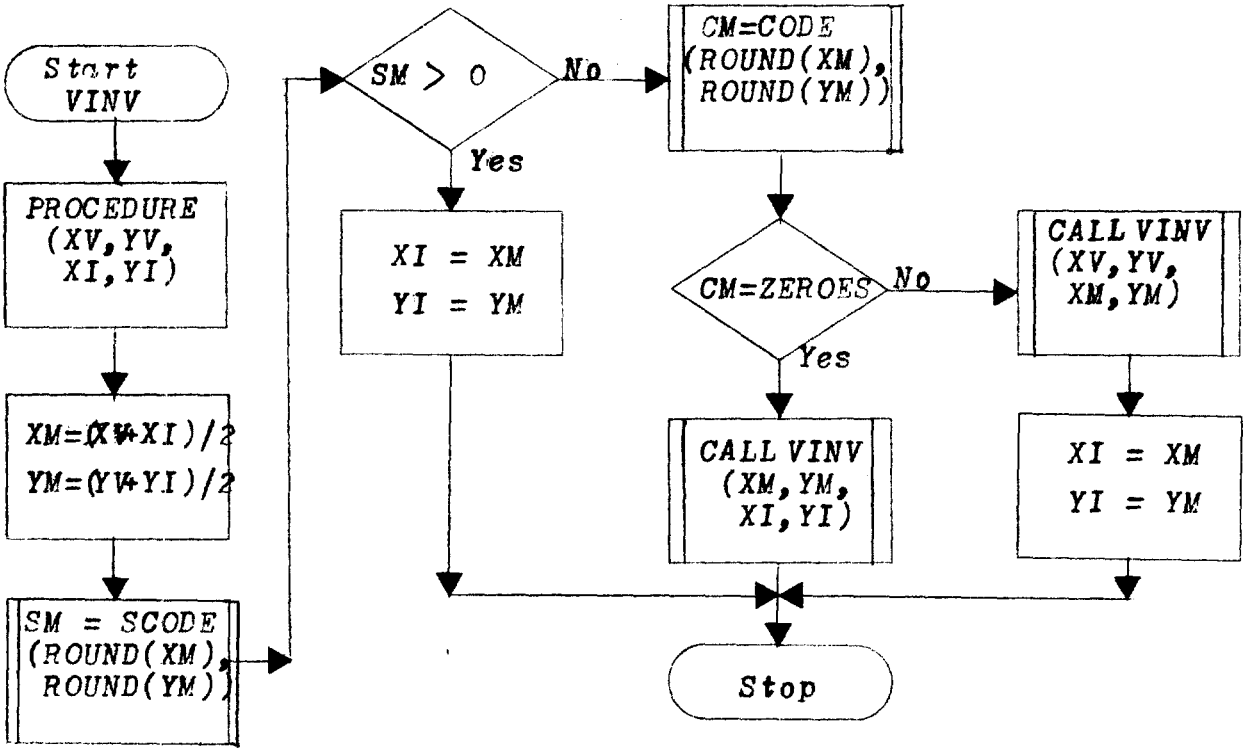
Function Procedures Called:

CODE - External.

SCODE - External.

ROUND - External.

Flowchart:



A SAMPLE RUN

The driver program for this sample run reads the screen size from cards and initializes the screen edges. It then reads from cards the endpoint coordinates for the line which is to be clipped. The coordinates are printed out before being passed to CLIP so the output from the algorithm can be more easily checked. After the call to CLIP, the procedure branches back to read the endpoint coordinates of the next line to be clipped. On end-of-file, the procedure brances to the end and stops execution.

In a real situation, the subroutine procedure DISPLN, called by the clipping algorithm, would display the visible portion of the line segment on the graphical display device. However, for this sample, it displays upon the printer the coordinates of the endpoints.

Figures 3-A and 3-B on pages 15 and 16, show graphically the input and output for this sample. The positive x- and y-coordinates of the screen edges were initialized to 16, with each unit square on the graph representing one unit square on the display screen. The lines have been numbered to correlate the two graphs with the ordering of the following list of output from the sample run:

```
ENDPOINTS OF GIVEN LINE: (    4,   16), (   16,    8)
ENDPOINTS OF VISIBLE SEGMENT: (    4,   16), (   16,    8)

ENDPOINTS OF GIVEN LINE: (    4,    8), (  -12,    6)
ENDPOINTS OF VISIBLE SEGMENT: (    4,    8), (  -12,    6)

ENDPOINTS OF GIVEN LINE: (   12,  -28), (   16,  -20)

ENDPOINTS OF GIVEN LINE: (   10,   -7), (   28,  -13)
ENDPOINTS OF VISIBLE SEGMENT: (   10,   -7), (   16,   -9)
```

```
ENDPOINTS OF GIVEN LINE: (  10,   -8), (  20,  -18)
ENDPOINTS OF VISIBLE SEGMENT: (  10,   -8), (  16,  -14)

ENDPOINTS OF GIVEN LINE: ( -27,   10), (  -9,   26)

ENDPOINTS OF GIVEN LINE: (  -8,   17), ( -28,    7)
ENDPOINTS OF VISIBLE SEGMENT: ( -11,   16), ( -16,   13)

ENDPOINTS OF GIVEN LINE: ( -24,   -6), (  24,    6)
ENDPOINTS OF VISIBLE SEGMENT: ( -16,   -4), (  16,    4)

ENDPOINTS OF GIVEN LINE: (   7,  -20), ( -20,  -11)
ENDPOINTS OF VISIBLE SEGMENT: (  -5,  -16), ( -16,  -12)
```

The first and second lines are trivial accepts; the third, a trivial reject. In the fourth and fifth lines, one endpoint is visible, while the other one is invisible. In the sixth, seventh, and eighth lines both endpoints are invisible. The last line of this sample is an example of a line which does not cross the screen edge at a grid intersection. The endpoints of the visible portion of this last line are rounded to the nearest grid intersection. Even when the line crosses at a grid intersection, some distortion, due to rounding, may occur. This distortion is minor, though, and would be unnoticeable on most displays.

Before Clipping

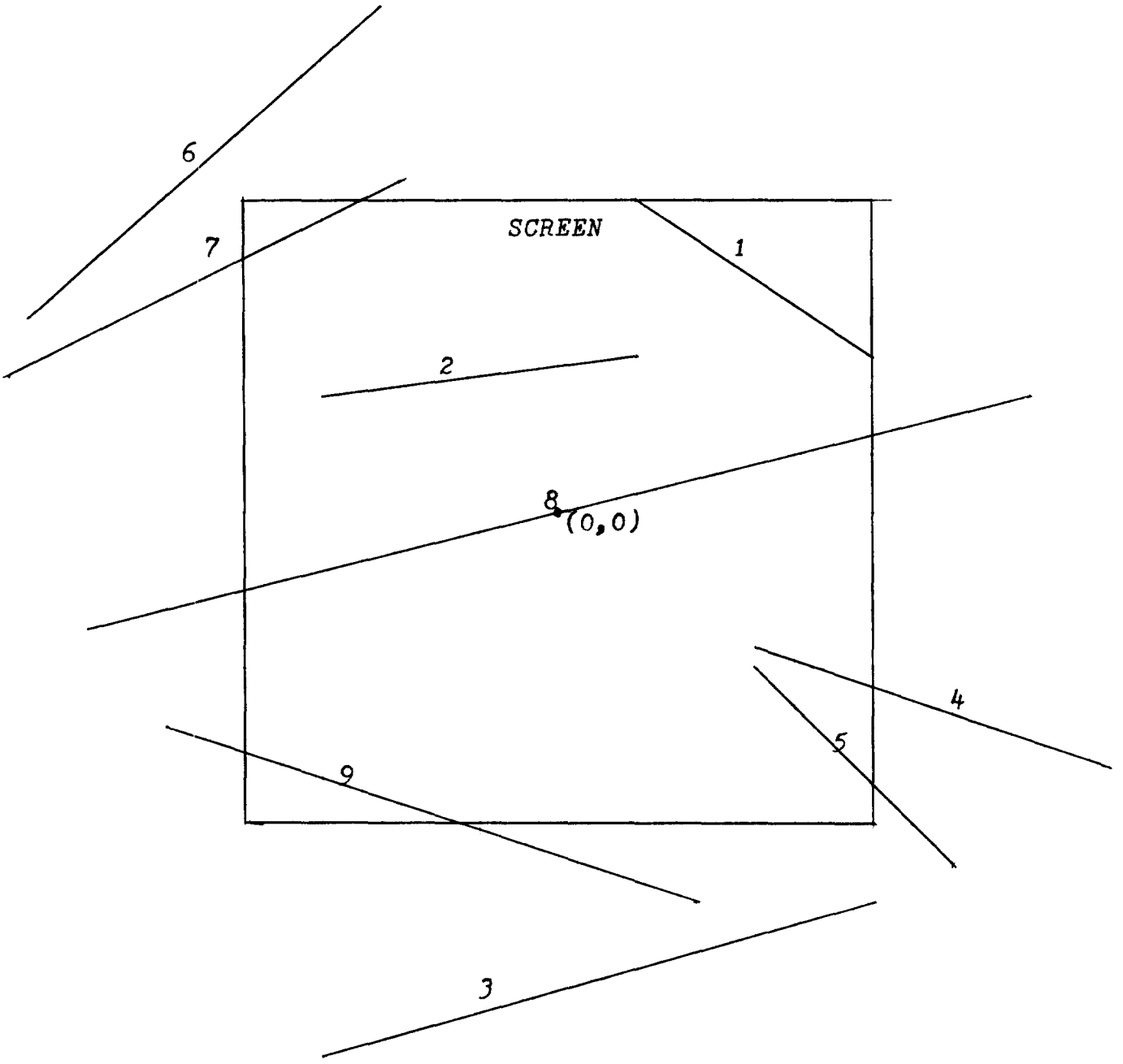


fig. 3-A

After Clipping

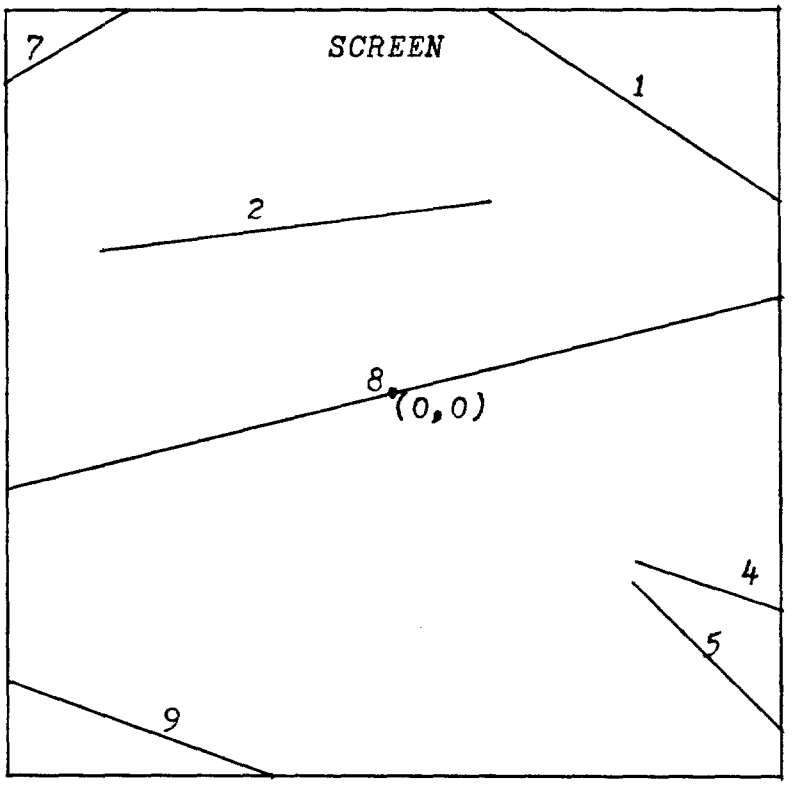


fig. 3-B

BIBLIOGRAPHY

Newman, William M. and Robert F. Sproull. Principles of Interactive Computer Graphics. McGraw-Hill. New York. 1973.

Pollack, S. V. and T. D. Sterling. A Guide to PL/1. Holt, Rinehart, and Winston, Inc. Chicago. 1969.

```
/*                                                                    */
/*                                                                    */
/*                                                                    */
/*                                                                    */
/*                                                                    */
/*                              APPENDIX                              */
/*                                                                    */
/*                                                                    */
/*PROCEDURE MAINP                                                     */
/* THIS PROGRAM READS IN THE ENDPOINTS OF THE ORIGINAL LINE AND       */
/*PASSES THEM TO THE CLIPPING ALGORITHM.  THE CALL TO CLIP1           */
/*INITIALIZES THE COORDINATES OF THE SCREEN EDGES.                    */
/*                                                                    */
MAINP:  PROCEDURE OPTIONS(MAIN);
        DECLARE (X1,Y1,X2,Y2) FLOAT(15),
                (XL,YL) FIXED(5,0);
        GET LIST(XL,YL);
        CALL CLIP1(XL,YL);
READ:   GET LIST(X1,Y1,X2,Y2);
        ON ENDFILE(SYSIN) GO TO THEE;
        PUT SKIP(3) EDIT('ENDPOINTS OF GIVEN LINE: (',X1,',',Y1,
                         '), (',X2,',',Y2,')')
                        (X(1),A(26),F(6,0),A(1),F(6,0),A(4),
                         F(6,0),A(1),F(6,0),A(1));
        CALL CLIP(X1,Y1,X2,Y2);
        GO TO READ;
THEE:   END MAINP;
```

```
/*                                                                    */

   /*                                                                 */
   /*                                                                 */
   /*SUBROUTINE PROCEDURE DISPLN                                      */
   /* THIS SUBROUTINE PRINTS THE COORDINATES OF THE ENDPOINTS         */
   /*OF THE VISIBLE SEGMENT AS COMPUTED BY CLIP.                      */
   /*                                                                 */
DISPLN: PROCEDURE(X1,Y1,X2,Y2);
        DECLARE(X1,Y1,X2,Y2) FIXED(5,0);
        PUT SKIP EDIT('ENDPOINTS OF VISIBLE SEGMENT: (',X1,',',Y1,
                      '), (',X2,',',Y2,')')
                     (X(1),A(31),F(6,0),A(1),F(6,0),A(4),
                      F(6,0),A(1),F(6,0),A(1));
        END DISPLN;
```

```
/*                                                                    */
/*                                                                    */
/*SUBROUTINE PROCEDURE CLIP1                                          */
/* THIS SUBROUTINE DETERMINES THE VISIBLE PORTION OF A LINE           */
/*SEGMENT.  IF THE LINE CANNOT BE ENTIRELY REJECTED OR ACCEPTED       */
/*IT IS DIVIDED AT ITS MIDPOINT.  THE ALGORITHM IS THEN APPLIED       */
/*TO EACH HALF.  THE PROCESS STOPS WHEN THE MIDPOINT COINCIDES        */
/*WITH AN EDGE OF THE SCREEN.  A MORE COMPLETE DISCUSSION OF THE      */
/*ALGORITHM CAN BE FOUND IN PRINCIPLES OF INTERACTIVE COMPUTER        */
/*GRAPHICS BY WILLIAM M. NEWMAN AND ROBERT F. SPROULL.                */
/*                                                                    */
/*SUBROUTINES USED:                                                   */
/*                                                                    */
/*  ROUND - INTERNAL, FUNCTION, ROUNDS A FLOATING POINT NUMBER        */
/*          TO THE NEAREST INTEGER.                                   */
/*                                                                    */
/*   CODE - INTERNAL, FUNCTION, ASSIGNS A FOUR-DIGIT BINARY CODE      */
/*          TO A POINT DEPENDING ON THE REGION IN WHICH IT LIES.      */
/*                                                                    */
/*  SCODE - INTERNAL, FUNCTION, ASSIGNS A ONE-DIGIT DECIMAL CODE      */
/*          TO A POINT DEPENDING ON THE EDGE OF THE SCREEN ON         */
/*          WHICH IT LIES.                                            */
/*                                                                    */
/*   VINV - INTERNAL, RECURSIVE SUBROUTINE, FINDS VISIBLE             */
/*          PORTION OF A LINE WHICH IS PARTIALLY VISIBLE.             */
/*                                                                    */
/* DISPLN - EXTERNAL, SUBROUTINE, DISPLAYS THE VISIBLE PORTION        */
/*          RECIEVED FROM CLIP, ON THE APPROPRIATE DEVICE.            */
/*                                                                    */
CLIP1:  PROCEDURE(XR,YT);
        DECLARE (XR,XL,YT,YB) FIXED(5,0);
/*                                                                    */
/* INITIALIZE THE EDGES OF THE SCREEN.                                */
/*                                                                    */
        XL = -XR;
        YB = -YT;
        RETURN;
CLIP:   ENTRY(X1,Y1,X2,Y2);
        DECLARE ROUND ENTRY(FLOAT(15))RETURNS(FIXED(5,0)),
                CODE ENTRY(FIXED(5,0),FIXED(5,0))
                RETURNS(FIXED BINARY(4,0)),
                SCODE ENTRY(FIXED(5,0),FIXED(5,0))
                RETURNS(FIXED(2,0)),
                DISPLN ENTRY(FIXED(5,0),FIXED(5,0),
                             FIXED(5,0),FIXED(5,0));
        DECLARE (X1,Y1,X2,Y2,XM,YM) FLOAT(15),
                (C1,C2,CM) FIXED BINARY(4,0),
```

```
/*                                                                    */

                ZERCES FIXED BINARY(4,C) INITIAL('CCCC'B),
                LAND FIXED BINARY(4,C) INITIAL('CCC1'B),
                SM FIXED(2,C);
        C1 = CODE(RCUND(X1),RCUND(Y1));
        C2 = CODE(RCUND(X2),RCUND(Y2));
/*                                                                    */
/* TRIVIAL REJECT.                                                    */
/*                                                                    */
        IF BOOL(C1,C2,LAND)¬=ZERCES THEN RETURN;
/*                                                                    */
/* TRIVIAL ACCEPT.                                                    */
/*                                                                    */
        IF (C1=ZERCES) & (C2=ZERCES) THEN GO TO PRT;
/*                                                                    */
/* FIND MIDPOINT.                                                     */
/*                                                                    */
        XM = (X1 + X2)/2;
        YM = (Y1 + Y2)/2;
        SM = SCODE(RCUND(XM),RCUND(YM));
        CM = CODE(RCUND(XM),RCUND(YM));
/*                                                                    */
/* ONE ENDPOINT MAY HAVE BEEN VISIBLE, ONE HALF OF THE LINE CAN       */
/* THEN BE EITHER ACCEPTED OR REJECTED AND THE ALGORITHM APPLIED      */
/* TO THE SECOND HALF.                                                */
/*                                                                    */
        IF C1=ZERCES THEN DO;
          IF SM>C THEN DO;
            X2 = XM;
            Y2 = YM;
            GO TO PRT;
          END;
          ELSE;
          IF CM=ZERCES THEN DO;
            CALL VINV(XM,YM,X2,Y2);
            GO TO PRT;
          END;
          ELSE DO;
            X2 = XM;
            Y2 = YM;
            CALL VINV(X1,Y1,X2,Y2);
            GO TO PRT;
          END;
        END;
        IF C2=ZERCES THEN DO;
          IF SM>C THEN DO;
            X1 = XM;
            Y1 = YM;
```

```
/*                                                                    */

                  GO TO PRT;
                END;
                ELSE;
                IF CM=ZEROES THEN DO;
                  CALL VINV(XM,YM,X1,Y1);
                  GO TO PRT;
                END;
                ELSE DO;
                  X1 = XM;
                  Y1 = YM;
                  CALL VINV(X2,Y2,X1,Y1);
                  GO TO PRT;
                END;
              END;
    /*                                                                */
    /* BOTH ENDPOINTS ARE INVISIBLE.  IF THE MIDPOINT IS VISIBLE      */
    /* EACH HALF IS TREATED SEPERATELY.  IF NOT, ONE HALF OF THE      */
    /* LINE CAN BE DISCARDED AND THE OTHER HALF PASSED TO VINV.       */
    /*                                                                */
TWO:    IF CM=ZEROES THEN DO;
          CALL VINV(XM,YM,X1,Y1);
          CALL VINV(XM,YM,X2,Y2);
          GO TO PRT;
        END;
        IF BOOL(C1,CM,LAND)¬=ZEROES THEN DO;
          X1 = XM;
          Y1 = YM;
          C1 = CM;
        END;
        IF BOOL(C2,CM,LAND)¬=ZEROES THEN DO;
          X2 = XM;
          Y2 = YM;
          C2 = CM;
        END;
MID:    IF (X1=X2) & (Y1=Y2) THEN RETURN;
        XM = (X1 + X2)/2;
        YM = (Y1 + Y2)/2;
        CM = CODE(RCLND(XM),RCLND(YM));
        SM = SCODE(RCLND(XM),RCLND(YM));
        GO TO TWO;
    /*                                                                */
    /*FUNCTION SUBROUTINE RCLND                                       */
    /* RCLNDS A FLOATING POINT NUMBER TO THE NEAREST INTEGER.         */
    /*                                                                */
RCLND:  PROCEDURE(XFLT) RETURNS(FIXED(5,0));
        DECLARE XFLT FLOAT(15),
                 XFXD FIXED(5,0);
```

```
/*                                                                  */

            XFXD = XFLT;
            IF (ABS(XFLT-XFXD)¬<.5) THEN DO;
              IF XFXD<0 THEN XFXD=XFXD-1;
              ELSE XFXD=XFXD+1;
            END;
            RETURN(XFXD);
            END RCLND;
        /*                                                           */
        /*FUNCTION SUBROUTINE CODE                                   */
        /* ASSIGNS A FOUR-DIGIT- BINARY CODE TO A POINT.  THE FOUR   */
        /* DIGITS MEAN THE FOLLOWING, IF SET:                        */
        /*    FIRST BIT - THE POINT IS ABOVE THE TOP EDGE OF THE SCREEN.*/
        /*   SECOND BIT - THE POINT IS BELOW THE BOTTOM EDGE.        */
        /*    THIRD BIT - THE POINT IS TO THE RIGHT OF RIGHT-HAND EDGE.*/
        /*   FOURTH BIT - THE POINT IS TO THE LEFT OF LEFT-HAND EDGE. */
        /*                                                           */
    CODE:   PROCEDURE(X,Y) RETURNS(BINARY FIXED(4,0));
            DECLARE (X,Y) FIXED(5,0),
                    C FIXED BINARY(4,0);
            IF X<XL THEN C='0001'B;
            ELSE
              IF X>XR THEN C='0010'B;
              ELSE C='0000'B;
            IF Y<YB THEN RETURN(BOOL(C,'0100'B,'0111'B));
            ELSE
              IF Y>YT THEN RETURN(BOOL(C,'1000'B,'0111'B));
              ELSE RETURN(BOOL(C,'0000'B,'0111'B));
            END CODE;
        /*                                                           */
        /*FUNCTION SUBROUTINE SCODE                                  */
        /* ASSIGNS A ONE-DIGIT CODE TO A POINT AS FOLLOWS:           */
        /*    0 - THE POINT DOES NOT LIE ON AN EDGE OF THE SCREEN.   */
        /*    1 - THE POINT LIES ON THE TOP EDGE OF THE SCREEN.      */
        /*    2 - THE POINT LIES ON THE BOTTOM EDGE OF THE SCREEN.   */
        /*    3 - THE POINT LIES ON THE RIGHT-HAND EDGE.             */
        /*    4 - THE POINT LIES ON THE LEFT-HAND EDGE.              */
        /*                                                           */
    SCODE:  PROCEDURE(X,Y) RETURNS(FIXED(2,0));
            DECLARE (X,Y) FIXED(5,0);
            IF ABS(X)<=XR THEN
              IF Y=YT THEN RETURN(1);
              ELSE IF Y=YB THEN RETURN(2);
            IF ABS(Y)<=YT THEN
              IF X=XR THEN RETURN(3);
              ELSE IF X=XL THEN RETURN(4);
            RETURN(0);
            END SCODE;
```

```
/*                                                                  */
   /*                                                               */
   /*SUBROUTINE VINV                                               */
   /* THIS RECURSIVE SUBROUTINE DETERMINES THE VISIBLE PORTION OF A*/
   /* LINE SEGMENT, PART OF WHICH IS VISIBLE.  IF THE MIDPOINT     */
   /* COINCIDES WITH AN EDGE OF THE SCREEN (S¬=0) IT, ALONG WITH   */
   /* THE VISIBLE ENDPOINT, MARK THE ENDS OF THE VISIBLE PORTION.  */
   /* IF NOT, ONE-HALF OF THE LINE CAN BE DISCARDED OR SAVED,      */
   /* DEPENDING ON THE VISIBILITY OF THE MIDPOINT, AND THE OTHER   */
   /* HALF PASSED TO VINV FOR FURTHER SUBDIVIDING.                 */
   /*                                                               */
VINV:   PROCEDURE(XV,YV,XI,YI)RECURSIVE;
        DECLARE (XV,YV,XI,YI,XM,YM) FLOAT(15),
                SM FIXED(2,0),
                CM FIXED BINARY(4,0);
        XM = (XV + XI)/2;
        YM = (YV + YI)/2;
        SM = SCODE(RCLND(XM),RCLND(YM));
        IF SM>0 THEN DO;
          XI = XM;
          YI = YM;
          RETURN;
        END;
        CM = CODE(RCLND(XM),RCLND(YM));
        IF CM=ZEROES THEN CALL VINV(XM,YM,XI,YI);
        ELSE DO;
          CALL VINV(XV,YV,XM,YM);
          XI = XM;
          YI = YM;
        END;
RET:    END VINV;
   /*                                                               */
   /* DISPLAY THE VISIBLE PORTION OF THE LINE.                      */
   /*                                                               */
PRT:    CALL DISPLN(RCLND(X1),RCLND(Y1),RCLND(X2),RCLND(Y2));
THE:    END CLIP1;
```