NBSIR 85-3121

SOLID - A FORTRAN Program for Displaying Three-Dimensional Surface Topographies

David E. Gilsinn

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Manufacturing Engineering
Gaithersburg, MD 20899

June 1985

U.S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

NBSIR 85-3121

SOLID - A FORTRAN PROGRAM FOR DISPLAYING THREE-DIMENSIONAL SURFACE TOPOGRAPHIES

David E. Gilsinn

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Manufacturing Engineering
Gaithersburg, MD 20899

June 1985

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, ***Secretary***
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, ***Director***

Disclaimer

Certain commercial equipment, instruments, or materials are identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

TABLE OF CONTENTS

Page

Page

1.0 Introduction

The techniques for image analysis depend on the distribution of the "grey levels" or pixel intensity levels of a picture displayed on an image processor. Combinatorial, statistical or Fourier techniques can be used to find picture boundaries, adjust contrast or modify the picture in frequency space. An entirely different approach to picture enhancement is to look at the picture from a geometric point of view and assume that the intensity value at a pixel represents a surface height, where brighter pixel intensities represent higher points and lower pixel intensities represent lower points. Then, if a viewer looks at the picture from various perspectives, the geometric image displayed can yield interesting information from a visual perspective. This was the background motivation for writing this program. The simulated solid generated by the program on the display monitor is sometimes called a pseudosolid since a nonreal three-dimensional effect can be used to enhance a two-dimensional picture such as a medical x-ray.

The program, documented in this manual, was modeled on an earlier program written by Dr. Roger L. Nagel of Lehigh University and discussed in Weber, Nagel [7]. This program is a redesign of Dr. Nagel's original code that takes advantage of the image processor features available to the author. Both programs implement an algorithm that generates a three dimensional solid image on the viewer's screen by using a technique that follows light rays from a light source to the reflecting surface and then to the viewer's eyes. This type of an algorithm is referred to as a ray tracing algorithm, see Goldstein, Nagel [5].

A large body of literature exists (see Foley and Van Dam [4]) describing computational techniques to display the images of three dimensional objects by a) eliminating hidden surfaces, b) applying shading or c) casting shadows. Most of the techniques require initial processing of the displayed scene. This usually requires that the program user describe in fairly great detail the geometric structure of the scene in such a manner that the objects in the scene can be decomposed into adjoining polygons. These techniques, however, become very unwieldy when the scene displayed is very rough, such as a very mountainous area. It is this situation that a user encounters when displaying the texture of machined metal parts. For this reason a point-by-point shadowing technique was used, see Appel [1].

The problem addressed by the progam documented here is as follows: Given a randomly rough surface, such as a machined metal surface, we assume that the topography of that surface has been digitized in such a way that the digital scale is proportional to the surface height. The specific technique used to acquire surface topographic maps is briefly described in Appendix C. Furthermore, we assume that the digitized points are distributed in such a way that they form an evenly spaced square grid. Each record in the digitized data file represents an amplitude trace across the surface with data values taken at, say, N evenly spaced points. There are N records in the data file. We wish to first simulate the illumination of the surface in order to highlight the rough topography and then to view the surface from various orientations.

This problem lends itself to the ray tracing technique. It does not require preprocessing of the data file in order to establish connectivity relations since they are inherent in the data file and the mode of acquisition. Ray tracing is a procedure that defines the light intensity at every point on the monitor screen by tracking a ray of light from the light source to the rough surface and then to the monitor as if it were a viewing window.

The ray tracing algorithm used in this program is a two-pass algorithm, see Crow [2], in the sense that the surface data file is processed twice, first to generate shadows and then to construct a solid projection of the illuminated surface. In the first pass the object surface is divided into two classes of areas, those that are shadowed and those that are not. The result is displayed on the user monitor in a form called a shadowgraph. The effect shown simulates a viewer looking down on a scene with light illuminating the surface from some specified direction. In the second pass of the algorithm the hidden surfaces relative to the viewer are removed and the resulting image is projected onto a window that represents the monitor screen. The advantage of this two-pass approach is that the shadow generation is separated from the picture generation process. This allows the viewer to look at the surface from various angles without changing the light source.

This manual approaches the documentation from the point of view of refinement in the sense that as one proceeds through this document, one begins with a global picture as a user in Section 2. This section also includes an interactive scenario. Next, from the point of view of the analyst, the document discusses in more detail the analytic geometry and algorithms that are implemented in the programs. This is contained in Sections 3 and 4, respectively. Section 5 includes listings of the main program and subroutines along with their flow charts. Finally, the appendices broadly describe 1) the general architecture of the image processor on which the algorithm was implemented, 2) the image processor specific and host system subroutine calls required and 3) the 3-D stylus data profilometer used to acquire the data.

The author feels strongly that a documentation of this nature serves a useful purpose. A fundamental myth must be abandoned by those who think that using a standard language on the host computer makes transportability possible. It is true that, since the algorithm was implemented in FORTRAN, the program could be transferred by tape and most likely compiled on another system. Any connection with portability stops there. The architecture and controlling software for image processors are all different. No standards for interfacing host driven software and image processor hardware exist. The implementation of any algorithm becomes an ad hoc exercise in communicating a mathematically described algorithm, by way of a possibly standard language, through specialized non-standard control programs to a unique device. Although an algorithm may be stated in a general form, the implementation of that algorithm on a specific device or combination of devices is usually a nontrivial undertaking. The author feels that it is worthwhile for those both familiar and unfamiliar with implementing graphics algorithms to see how a general algorithm is tailored to a particular system. The author hopes that the detailed discussion of the algorithm will encourage others to modify this program or rewrite it as necessary in order to implement the algorithm on another system. Dunham [3] has also voiced similar sentiments.

2.0 Program Operation

2.1 Algorithm Overview

Since the nature of the process used to generate a solid is accomplished in two applications of the same essential algorithm the program was structured in such a manner that the user can run it from the beginning to the final solid generation or terminate it after creating a shadowgraph. In either case the program assumes that a surface topographic image is available as a sequential file with 512 records of 512 bytes each. Furthermore, the program assumes that the image processor being used is enabled and that the display monitor is on.

The topography of a surface can be interpreted as a grid of impulse spikes with amplitudes ranging from 0 to 255. This image is limited by the hardware only. Each spike is referred to as a pixel or picture element. It is displayed on the monitor as a dot, whose intensity is controlled by the values 0 to 255. 0 is the lowest intensity (black) and 255 is the highest (bright white). The entire topography when stored in the image processor is specified by 512 x 512 dots. The intensity of the dots represent the digitized amplitude of the surface at that point. The higher the point, the higher the intensity. The lower the surface point, the lower the intensity. For a discussion of the digital data acquisition techniques used to acquire surface topographies, the reader is referred to the Appendix C.

If a shadowgraph of an image is not available then one must be created. This is done on the first pass. Once the program initializes the image processor, it asks the user for the file

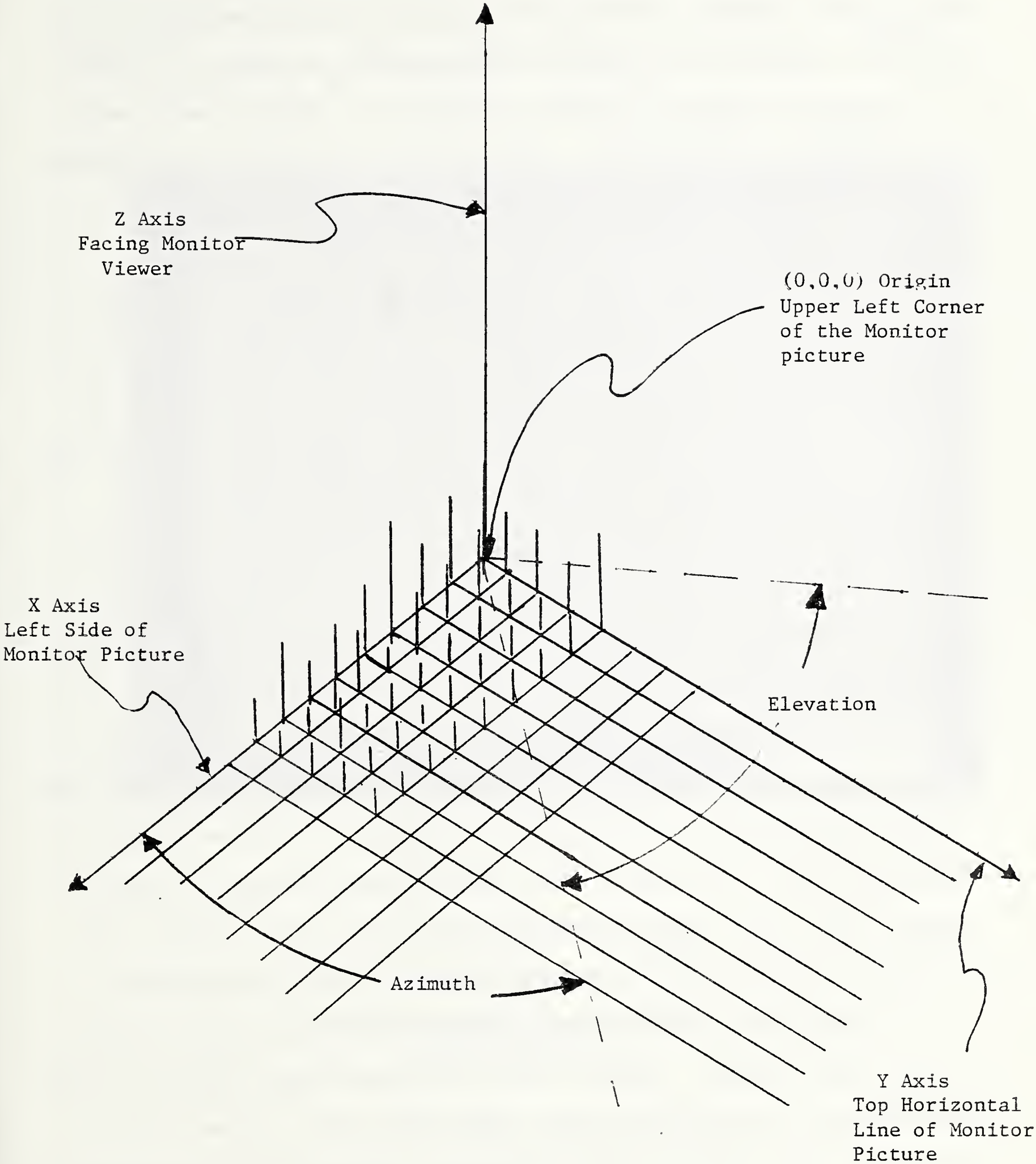


Figure 1

Conceptual Picture of a
Topographic Map

name of the topographic image. The program displays the image on the graphics monitor and asks the user for the azimuth and elevation angles of the desired light source as shown in Figure 1. A typical surface is shown in Figure 2. This is what would be displayed on the monitor. The surface in Figure 2 represents approximately 1mm x 1mm of stainless steel with a roughness height of about 1 µm.

In the geometric model used for this program the image is assumed to form a solid within a box (called the bounding box) of sides 511 units in length (512 grid points per side) and height 255 units (256 grid points in height). The 256 height grid points represent the intensity levels for the image processor. The origin of the right hand coordinate system, referred to as the world coordinate system, lies at point (0,0,0) of the box in Figure 3. The origin is sometimes referred to as 0. Referring to Figure 3, consider the vector $\vec{OA}$. The azimuth angle, AZ, of the vector $\vec{OA}$ is measured from the positive X axis in a counterclockwise manner. The elevation, EL, is measured from the XY-plane vertically. In Figure 3, if the point A were taken as (511, 511, 255), the angle, EL, would be approximately 19.5 degrees. For most cases the elevation of the light source used is usually greater than 20 degrees and less than 90 degrees.

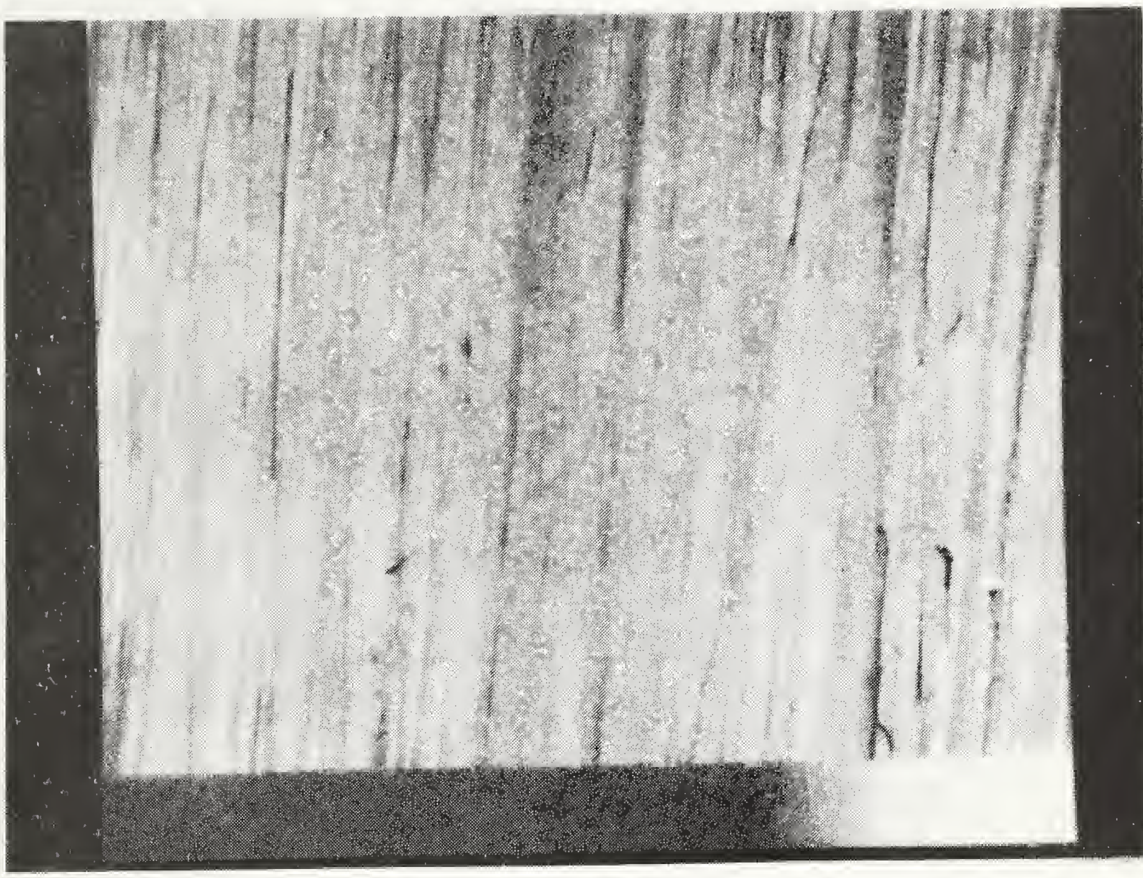
The idea behind the general algorithm used in this program is relatively simple. Every line in the Euclidean 3-space can be represented by an equation that associates the Z value on the line with a pair (X,Y) on the XY-plane. These (X,Y) values fall on a line that is the projection of the line in space onto the XY-plane. This line is called the <u>scan line</u>. As one steps along

Origin (0,0)

(0,511)

Y Axis

X Axis



(511,0)

(511,511)

Figure 2

A Surface Topographic Image with Grey Level Bar

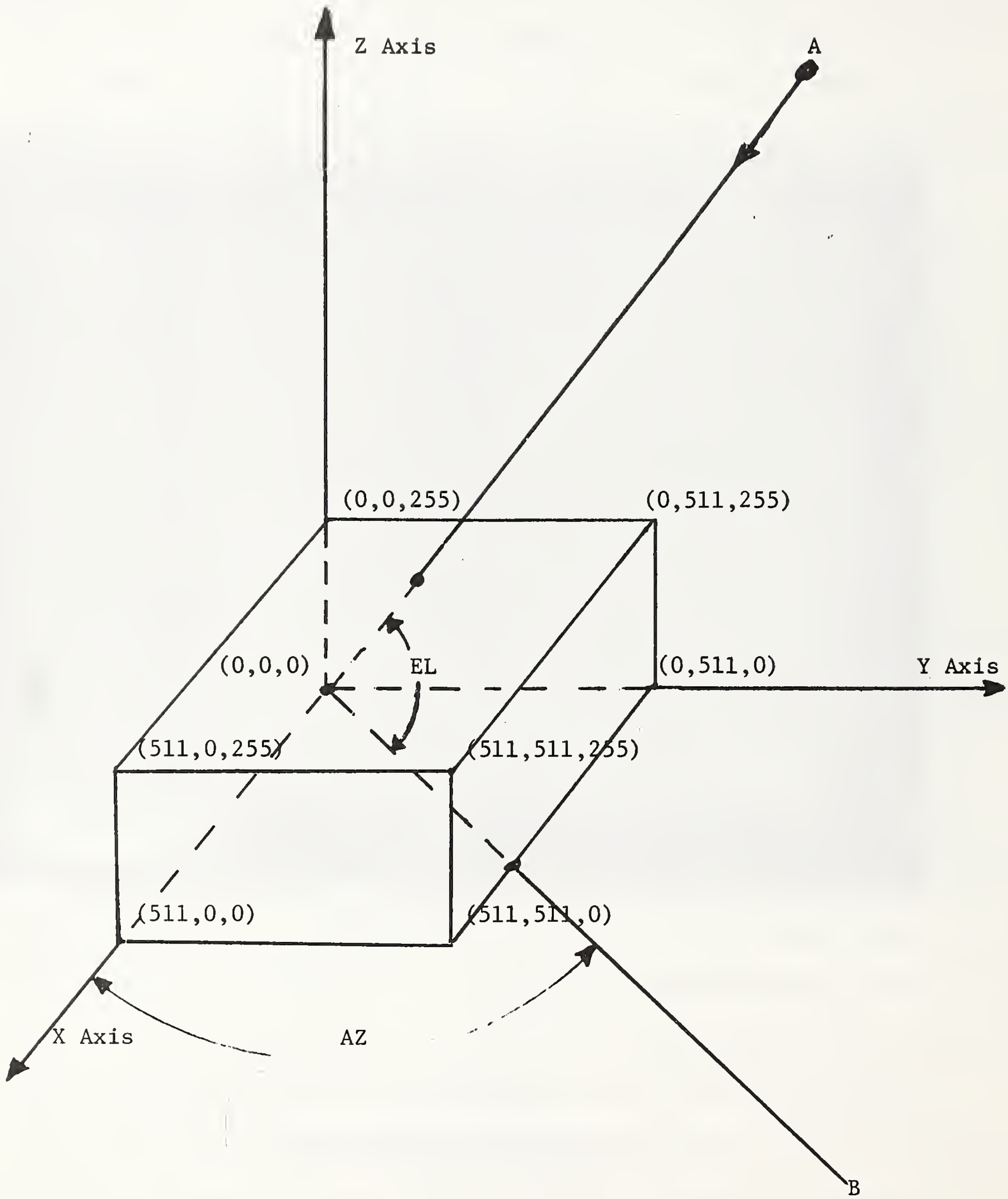


Figure 3

Bounding Box with Azimuth
and Elevation Shown

the scan line, one can always find the associated value Z in space. Therefore, the line in space can be "traced" by sequentially selecting (X,Y) points along the scan line.

In order to trace rays by computer from a source to the topographic surface some bound must be put on the set of all rays in order to establish program limits for those rays that will be traced to the surface. Consider Figure 4. Assume that the light source lies at infinity so that all light rays are parallel. Let the azimuth and elevation of the light source be AZ and EL, respectively, as already shown in Figure 3. Notice that the only possible rays that could impinge on the surface lie between the two planes P_1 and P_2. These are vertical planes intersecting the XY-plane at the lines L_1, L_2, where L_1 and L_2 are lines through the points EX1 and EX2 in Figure 4. These points will be called <u>extreme points</u>. L_1 and L_2 are parallel to the projection of the vector $\vec{OA}$ onto the XY-plane. This projection is the line from (0,0,0) to B in Figure 3. Once the azimuth and elevation are given, the program can look up the extreme points from a prespecified table. These are then used as bounds on where the program begins and ends.

Once the extreme points for the image have been identified ray tracing from the light source to the surface can begin. The procedure traces rays beginning at extreme point EX1 up planes parallel to P_1 and P_2 moving incrementally to a new plane P after each cycle of the shadowgraph portion of the algorithm is completed. The program stops when extreme point EX2 has been encountered. On each plane P, rays are traced beginning with the

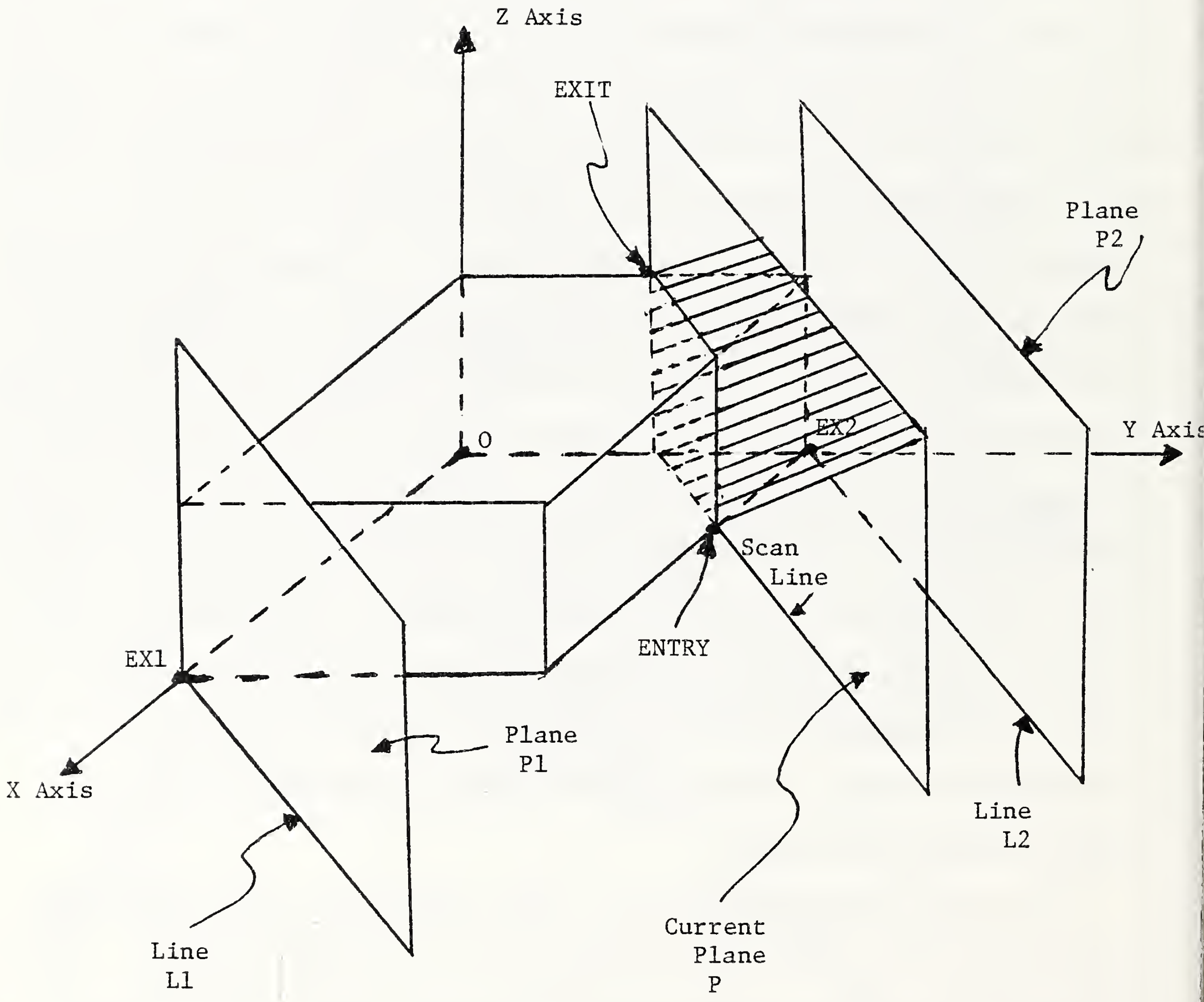


Figure 4

Extreme and Current Planes

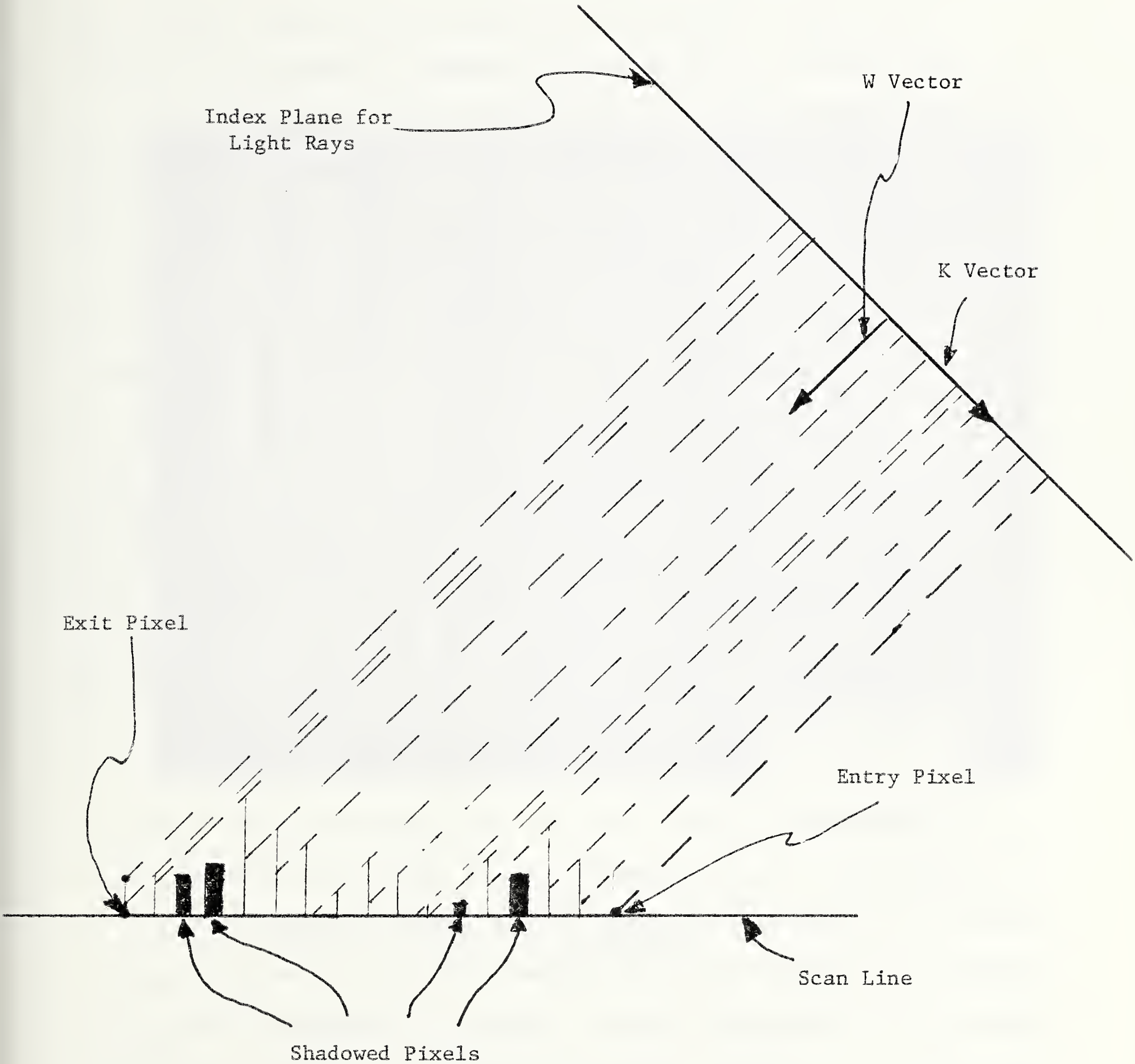


Figure 5

Geometry for the
Shadowgraph

one passing through the ENTRY point in Figure 4 and terminating with the one through the EXIT point.

In each plane P parallel to P_1 and P_2, rays are traced from the light source to a point on the surface. Referring to Figure 5, if the height of a pixel falls below a ray then nothing is entered into the shadowgraph map, which is displayed as the host program creates it; that is, a black pixel is displayed. If the height of the pixel is greater than or equal to the ray height, the original pixel value is written to the shadowgraph. The set of all points whose values fall below all rays traced are those in shadow. The shadowgraph for a sample calculation, with AZ=45 and EL=75, is given in Figure 6. The black areas of the picture are the points of the original image that fall within the shadow. The shadowgraph then is a second image that, along with the original image, is used to shadow the solid image in the second part of the program. Once the shadowgraph has been generated the user may save the image, if desired, before proceeding to generate a solid.

In generating a solid the user can designate a portion of the shadowgraph for solid generation by using the interactive trackball and function buttons of the image processor. The trackball is used to move the screen cursor in order to specify vertices of a rectangle called a <u>region of interest</u>. The user then enters the azimuth angle and elevation angle for the viewer rays and a percentage value used to reduce the intensity of those pixels designated for shadowing. From experience a reasonable number is 45%.

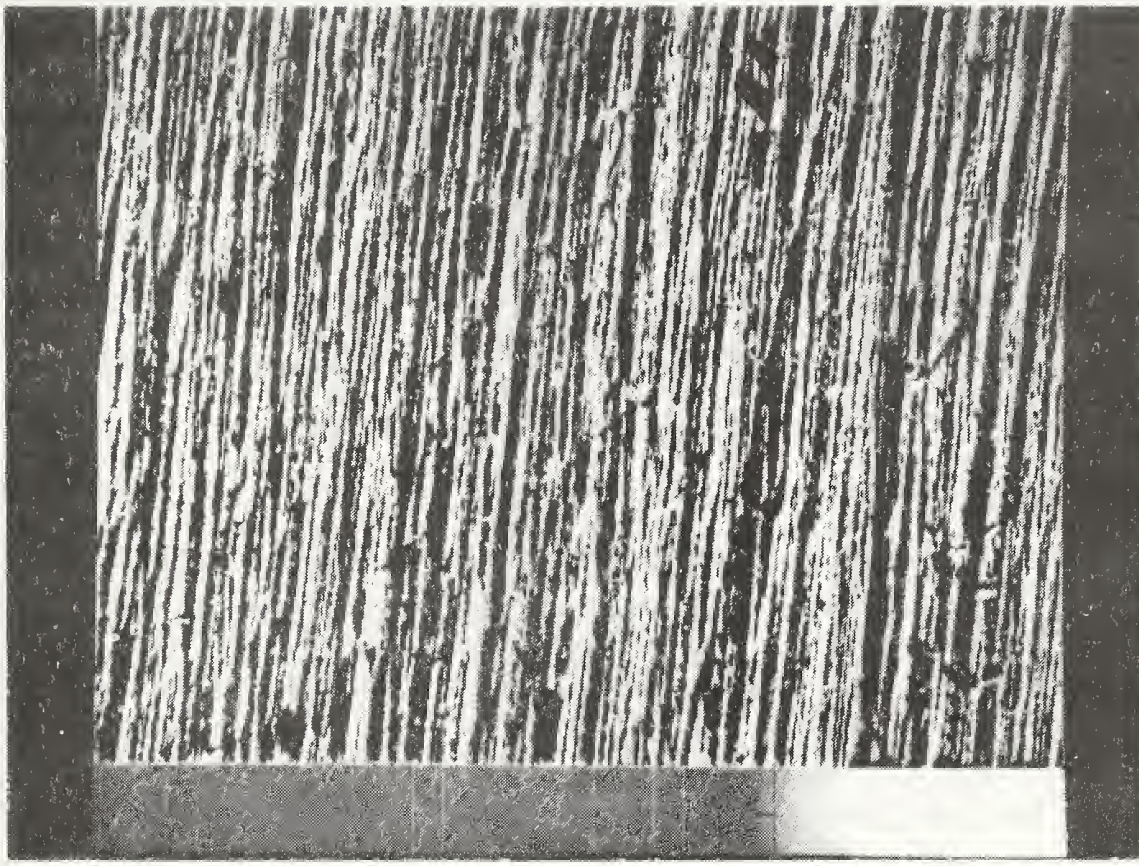


Figure 6

Sample Shadowgraph

From the azimuth and elevation angles the viewer frame of reference is generated. The viewer frame of reference, represented by unit vectors $\vec{K}$, $\vec{W}$, and $\vec{K}\times\vec{W}$ in Figure 7, is an orthonormal set of vectors used to trace points along rays from the viewer's eyes to the surface and to index points on the viewing screen. The screen can be considered a viewport opening in an extended plane in front of the viewer called the <u>viewplane</u>.

Given the selected rectangular area (region of interest) of the shadowgraph and the viewer coordinate system, the extreme points are selected, as in the shadowgraph pass of the algorithm. The viewing screen is indexed by the two coordinate vectors, $\vec{K}$ and $\vec{K}\times\vec{W}$. $\vec{K}$ points vertically down the screen and $\vec{K}\times\vec{W}$ points to the user's right. The program computes the multiples of the $\vec{K}\times\vec{W}$ vector that project down to the extreme points. These multiples are added to the projection of the center of the region of interest onto the viewplane. They are used to index planes P1, P2 as in Figure 4. These planes bound the computations.

For the sake of terminology, vertical lines on the screen are referred to as <u>columns</u> and horizontal lines as <u>rows</u>. The program starts at the column containing the left most projection of an extreme point and then moves to the next column. In each column it processes pixels from the bottom of the screen to the top. The column it works on is called the <u>current column</u>. The program first finds the initial viewscreen row for that current column. Then the row is selected that is the projection of the entry point of the viewing rays into the solid, see Figure 4. Since each pixel falls on a screen row, that row is referred to as the current row.

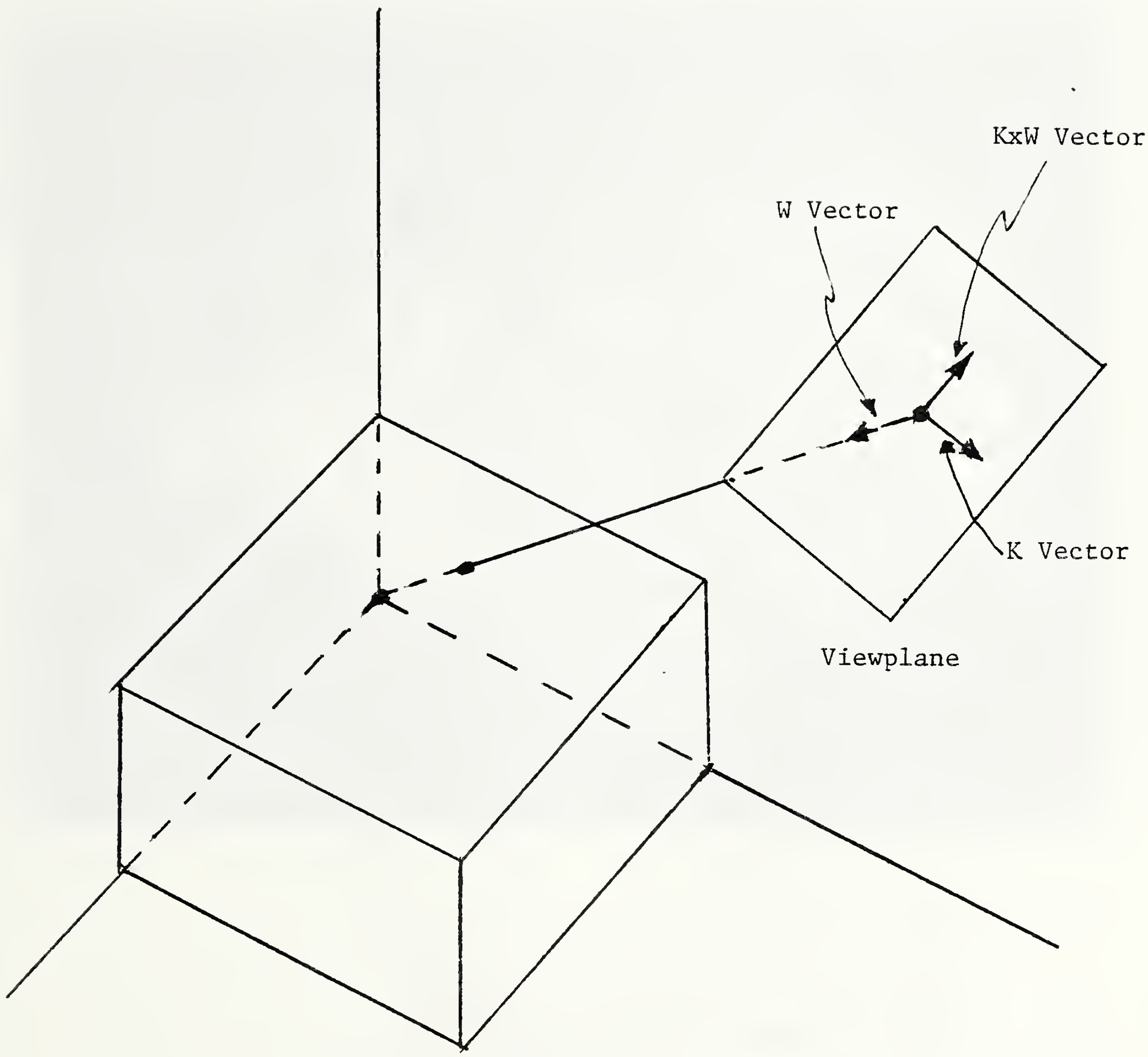


Figure 7

Viewplane Coordinate System

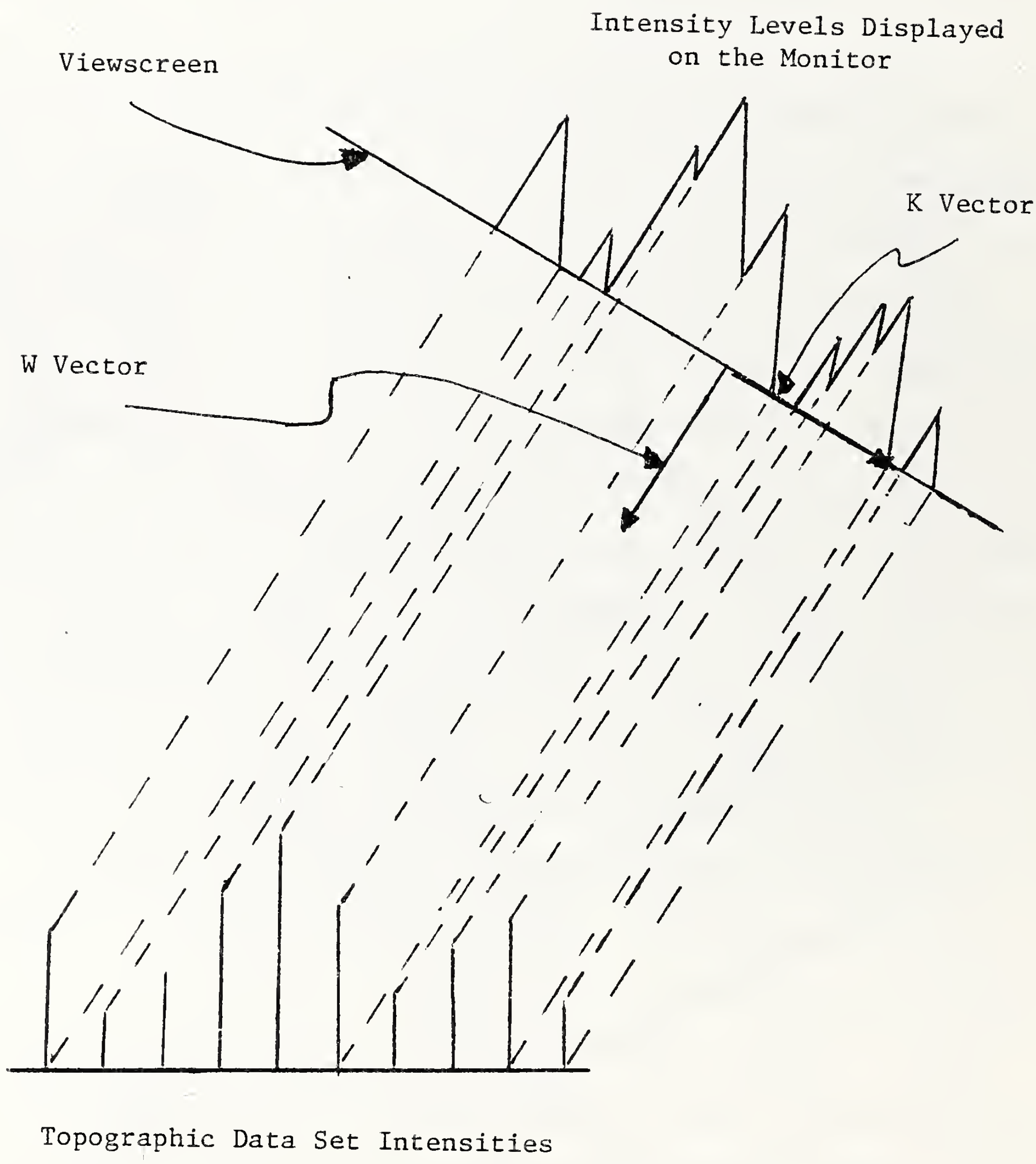


Figure 8

Correspondence
of Displayed Intensities to
Surface Amplitude Values

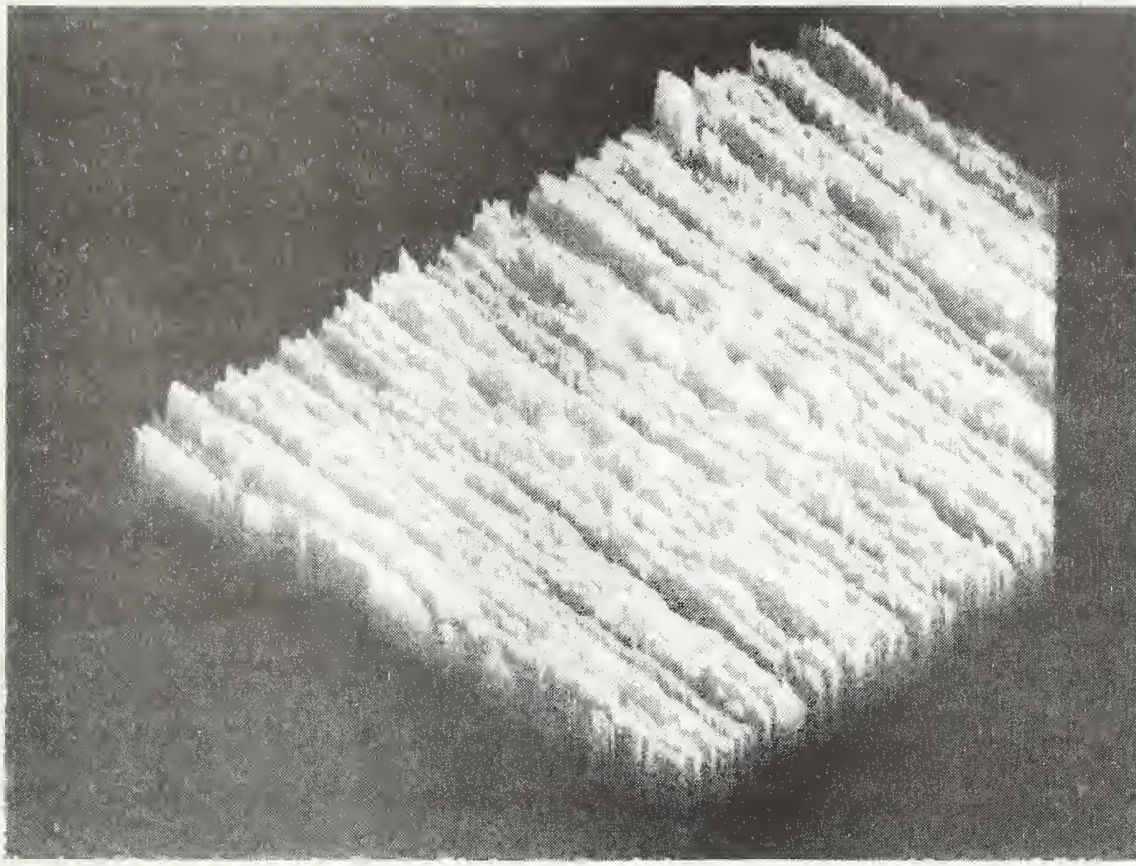


Figure 9

Projected Image of a
Surface Sample

For the sake of a mental picture, one can think of image pixels as spikes sitting above the base X,Y-plane (Figure 1). The height of each spike represents the intensity value of the image pixel. From the point of view of a spectator looking at the solid from an angle, more than just the top of the spike is seen, as would be the case if one were looking directly down on the image. The first ray, selected for processing, is that which encounters the base point of a pixel sitting at the boundary of the solid area. The program then traces rays through each pixel in the current column until it hits the point representing the top of the boundary pixel. The intensity of the points displayed represent the height of the viewing ray above the XY-plane when it encounters the spike representing the pixel, see Figure 8. This is why at the boundary the intensities in Figure 9 rise in value from 0 to the full value of the pixel. If the pixel lies in a shadow, e.g. the darker section of the projected surface in Figure 9, all of the values displayed for that spike are reduced by the selected percentage. The right hand side of Figure 9 shows the computer processing a column vertically.

Once a ray misses the top of a pixel in a column it is traced along until it hits another pixel spike. The height of the ray at that point is then written to the screen. The process continues up the column until a ray leaves the area selected. This is shown in Figure 8.

After a column has been processed from bottom to top the program moves to the next column on the right and starts at the bottom again. This continues until that column is reached that contains the projection of the second extreme point. This is the last column processed.

2.2 An Interactive Session

In this section the user is introduced to the interactive dialogue used by the program. Before beginning, the user is assumed to have read privileges for the desired image data file, which consists of 512 logical records, each consisting of 512 contiguous bytes.

Assume then that the user has signed on and verified access to the required data file and the host system has returned the prompt character. On the host system used by the author, this is an *. The user enters SOLID followed by a carriage return. It would look like this

*SOLID<CR>

where <CR> stands for the non-printing character for carriage return. This calls a user created command file that loads the program task, assigns the appropriate peripheral devices to the job, and then starts the job.

The program first prints

```
IF YOU WISH TO SHADOW A PICTURE TYPE 0.
IF YOU WISH TO CREATE A PSEUDO-SOLID TYPE 1.
***NOTE: TO PSEUDO-SOLID AN IMAGE A SHADOWGRAPH
         MUST HAVE PREVIOUSLY BEEN CREATED.
```

followed by a program prompt character. For the author's system this is a > character. The program user must enter something at this point.

Assume that a shadowgraph does not exist. Then the user types a 0 and a carriage return. It would look like this

```
>0<CR>
```

where 0 represents the zero. The program then prints

```
***************************
*      PSEUDO-SOLID       *
*        PHASE 1          *
*      SHADOW GRAPH       *
***************************

     THIS PORTION OF THE PSEUDO-SOLID GENERATION
SIMULATES THE EFFECT OF A DISTANT SOURCE OF LIGHT
SHINING ON THE SURFACE.  THOSE AREAS OF THE SURFACE
THAT WOULD BE SHADED ARE DARKENED.  NO THREE-
DIMENSIONAL EFFECT IS CREATED IN THIS PART.
     THE MAIN PICTURE IS WRITTEN TO CHANNEL 1
AND THE SHADOW GRAPH IS GENERATED ON CHANNEL 2.
```

This is followed by

```
ENTER NAME OF IMAGE FILE YOU WISH TRANSFERRED
MAX OF 16 CHAR.
>
```

Assume for the sake of this example that an image resides on a disk with disk name IMG: and the image file is SURFACE.DAT. Then after the > the input would look like

```
>IMG:SURFACE.DAT<CR>
```

At this time the host would transfer the image to refresh memory 1 of the image processor. For a description of the general architecture of the image processor used, see Appendix 1. After the picture has been transferred, the host returns the message

```
IF THE PICTURE HAS BEEN PROPERLY GENERATED,
TYPE 1, OTHERWISE 0 TO GET ANOTHER PICTURE.
>
```

If the user types 0, the host asks for the file name again. If 1 is entered as in

```
>1<CR>
```

the program next asks for the azimuth and elevation angles of the light source. See Figures 1 and 4.

```
ENTER AZIMUTH ANGLE AND ELEVATION ANGLE
IN DEGREES FOR THE LIGHT SOURCE.
AZIMUTH ANGLE LIMITS ARE 0 TO 360
ELEVATION ANGLE LIMITS ARE 0 TO 90
ENTER AS AZ , EL.
>
```

Suppose, for example, that the user would like to shadow the surface with an azimuth of 45 degrees and elevation of 75 degrees, then the input sequence would look like

```
>45.,75.<CR>
```

If the user enters any value outside of the limit, the message and prompt will appear again. If the user enters the first value and misses the second the host ordinarily will return with the prompt >, expecting the second value.

As soon as these data values have been entered the host and image processor start generating the shadowgraph. In the case above, the user would see tracing beginning on the monitor at the left hand lower corner and proceed along the diagonals beginning at the bottom of the right hand side and tracing to the left or top side at a 45 degree angle. Figure 6 is the resulting shadowgraph for Figure 2.

After the shadowgraph has been generated the user is given an option to save the shadowgraph with the message

```
IF YOU WISH TO SAVE THIS SHADOWGRAPH TYPE 1.
OTHERWISE 0.
>
```

If the user enters 1, such as

```
>1<CR>
```

the program returns the message

```
ENTER THE NAME OF THE FILE YOU WISH TO CREATE.
MAX OF 16 CHARACTERS.
>
```

to which the user would supply a file name with extension .SHW, to designate a shadowgraph file, in the form

>IMG:SURFACE.SHW<CR>

The program then enters the second pass of the algorithm. In this part a projection of a selected portion of the image is generated on the viewplane. If at the beginning of the program the user had selected to bypass the shadowgraph generation the following message is written. It is also written after the shadowgraph is generated.

```
IF THE ORIGINAL PICTURE IS IN CHANNEL 1 AND
ITS SHADOWGRAPH IS IN CHANNEL 2 THEN TYPE 1
OTHERWISE TYPE 0 TO TRANSFER THE PICTURES.
>
```

If a shadowgraph previously exists and the user wishes to generate a solid image projection then enter 0 in the form

>0<CR>

If 0 has been selected then the following is printed

```
******** LOADING ORIGINAL IMAGE ********
```

followed by

```
ENTER NAME OF IMAGE FILE YOU WISH TRANSFERRED,
MAX OF 16 CHAR.
>
```

At this point the user types the image data file name and the file is transferred to the image processor and displayed on the monitor. The program then prints the message

```
IF THE PICTURE HAS BEEN PROPERLY GENERATED,
TYPE 1, OTHERWISE 0 TO GET ANOTHER PICTURE.
>
```

If a 1 is typed, this message is followed by

```
******** LOADING THE SHADOWGRAPH ********
```

followed by

```
ENTER THE NAME OF IMAGE FILE YOU WISH TRANSFERRED,
MAX OF 16 CHAR.
>
```

Here the user must enter the shadowgraph file name for the image displayed on the monitor. After the image is transferred it remains visible on the monitor and the message

```
IF THE SHADOWGRAPH HAS BEEN PROPERLY GENERATED,
TYPE 1, OTHERWISE 0 TO GET ANOTHER SHADOWGRAPH.
>
```

appears on the user console. If the user types 1 then the program moves to an interactive mode in which the user must identify a rectangular region of interest on the shadowgraph that will be used to project a solid onto the display monitor. The user interacts with the image processor by way of a trackball with function buttons. For an illustration of the system configuration see Figure 10. The first message printed on the user console is

```
******** IDENTIFY THE REGION FOR PSEUDO-SOLID
******** ENHANCEMENT BY USING THE TRACKBALL.

THE USER MUST IDENTIFY TWO DIAMETRICALLY OPPOSITE
CORNERS OF A RECTANGLE USING THE TRACKBALL BUTTONS.
MOVE THE CURSOR WITH THE TRACKBALL TO THE FIRST
CORNER OF THE RECTANGLE OF INTEREST. PUSH BUTTON A.
```

The user then selects the upper left corner of the desired rectangle with the cursor by way of the trackball. After selecting the point the user presses button A on the trackball housing. Once the processor has selected the point the host computer displays the message

```
NOW MOVE THE CURSOR TO THE DIAMETRICALLY OPPOSITE
CORNER OF THE RECTANGLE OF INTEREST. PUSH BUTTON A.
```

After moving the cursor by way of the trackball to the diametrically opposite corner of the desired rectangle the user

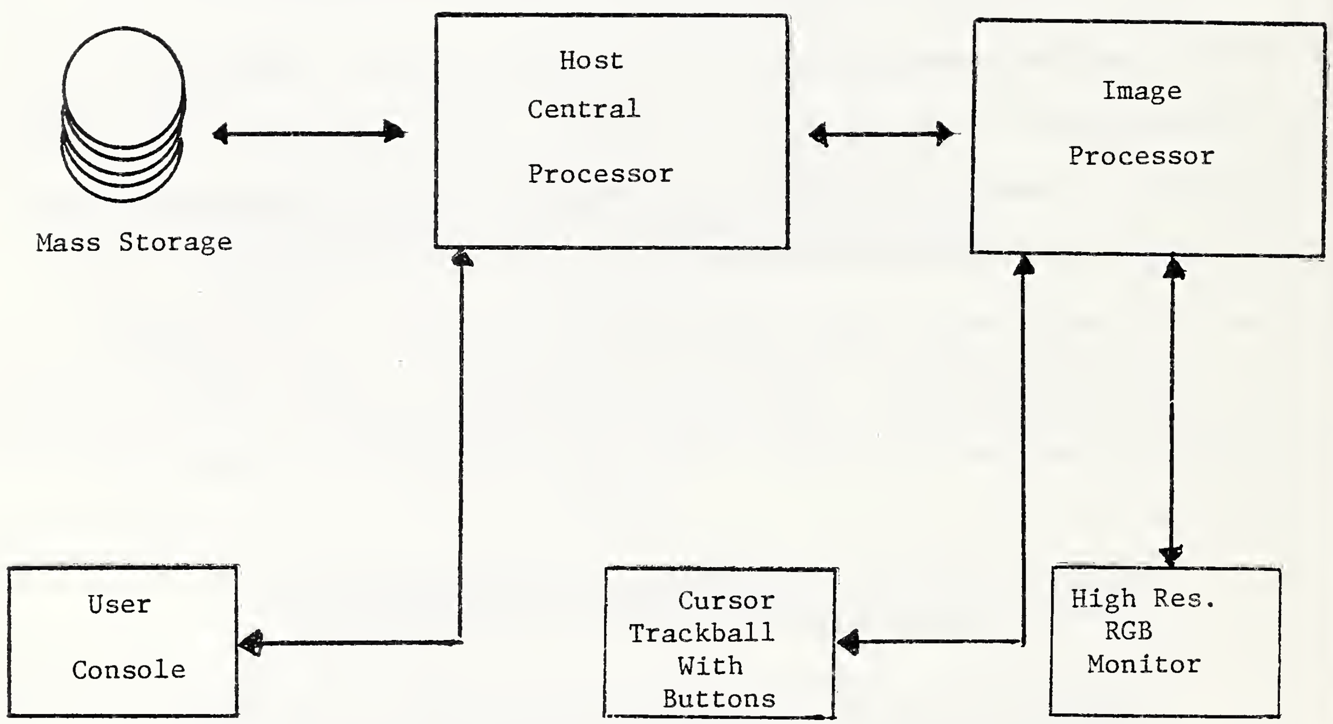


Figure 10

System Configuration

again pushes button A. The image processor then outlines the rectangle selected by drawing boundary lines and places a plus sign in the center to indicate the central point that the viewer will be seeing when the solid is projected. If the user has made an error in selecting the rectangle and wishes to select a new rectangle the processor allows this with the message

```
IF YOU WISH TO CHANGE YOUR MIND ON THE
RECTANGLE OF INTEREST PUSH BOTTON B, OTHERWISE
PUSH BUTTON A
```

Assuming that the user presses button A, the host then prints the message

```
ENTER AZIMUTH ANGLE AND ELEVATION ANGLE
IN DEGREES FOR THE VIEWER. AZIMUTH ANGLE
LIMITS ARE 0 TO 360. ELEVATION ANGLE LIMITS
ARE 0 TO 90.
ENTER AS AZ, EL.
>
```

If a user wishes to view the solid from an azimuth of 315 degrees and elevation angle of 75 degrees, the following would be entered

```
>315.,75.<CR>
```

The host computer then returns with

```
ENTER THE PERCENT REDUCTION IN INTENSITY DESIRED
FOR SHADOWING. ENTER FROM 0. TO 100.
>
```

Since the viewer will in general look at a surface from a direction other than that of the light source, some of the points seen would normally fall into shadow. From ordinary experience areas that are shadowed, say by trees or houses, are still visible but with reduced intensity. The reduced intensity comes from any diffuse lighting of the scene. In order to simulate this effect the user can enter a percentage value that will be used by the program to reduce the intensity of pixels seen by the viewer but are cast into shadow. From user experience percentage values of 40 to 45 percent reduction give an adequate shadow simulation. Therefore for a 45 percent reduction the user would enter

>45.<CR>

The program immediately starts generating the solid projected image moving from left to right on the screen tracing vertically from bottom to top. Some sample solids are shown in Figures 11 and 12. These pictures represent two views of the same region of interest of the surface in Figure 2. In particular, the region of interest is a portion of the upper right quadrant of the picture.

At the end of the solid generation the program prints

```
IF YOU WISH TO SAVE THE PSEUDOSOLID IMAGE TYPE
1, OTHERWISE 0
>
```

If the user types 1 then the message

```
ENTER THE NAME OF THE FILE YOU WISH TO CREATE,
MAX OF 16 CHARACTERS.
>
```

appears after which the user types the file name desired followed by the extension .SOL to indicate that this is a solid image, as for example, IMG:SURFACE.SOL. This is followed by a carriage return. The following message appears

```
IF YOU WISH TO GENERATE ANOTHER SOLID TYPE 1,
OTHERWISE 0
>
```

If the user types 0 the program terminates, and if the user types 1 the same shadowgraph will be used and the processor prints the next message to the user console

```
IF YOU WANT THE SAME REGION-OF-INTEREST TYPE 1,
OTHERWISE 0
>
```

If the user types 1, then the same rectangle as earlier outlined would be used but the user can look at it from a different viewpoint by selecting a new viewing azimuth and elevation. If a 0 is entered, the program returns to the shadowgraph and allows the user to select a new rectangle for solid projection. Processing then continues as before.

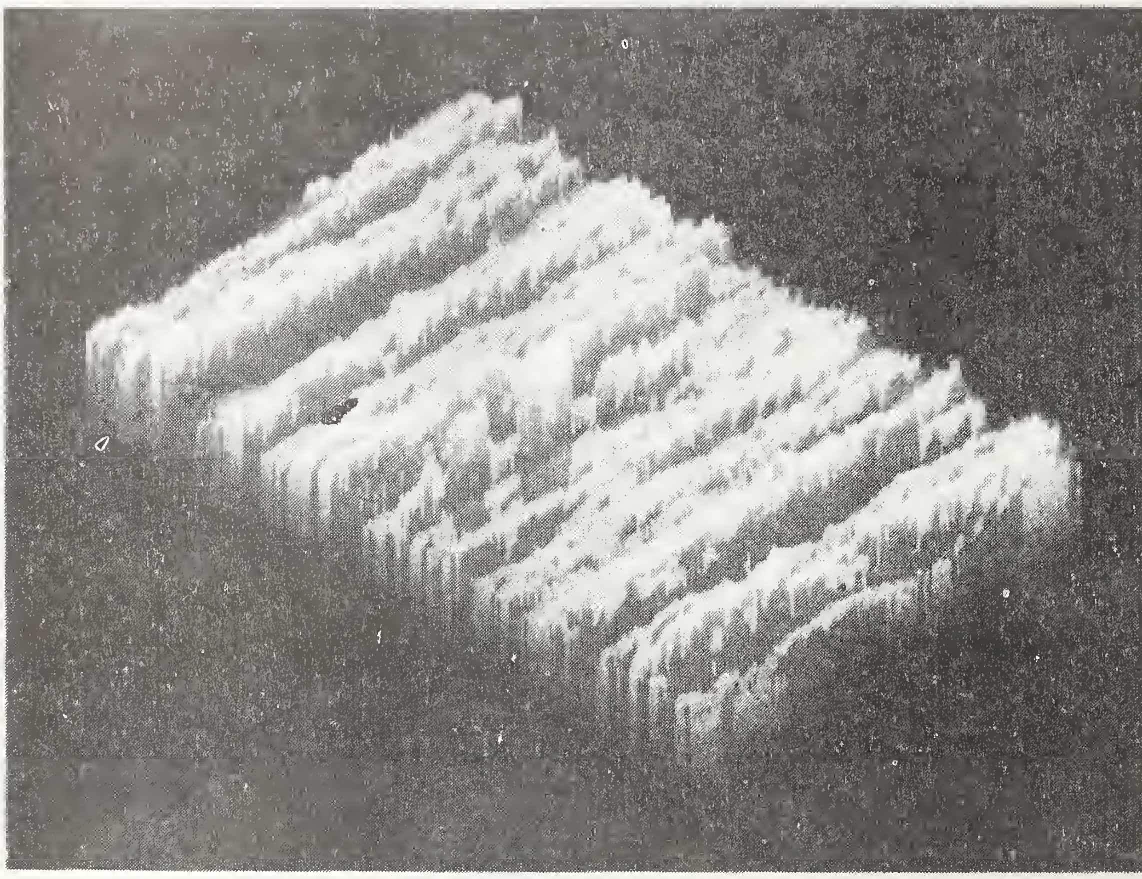


Figure 11

Solid Projection of a Portion
of the Upper Right Corner of Figure 2

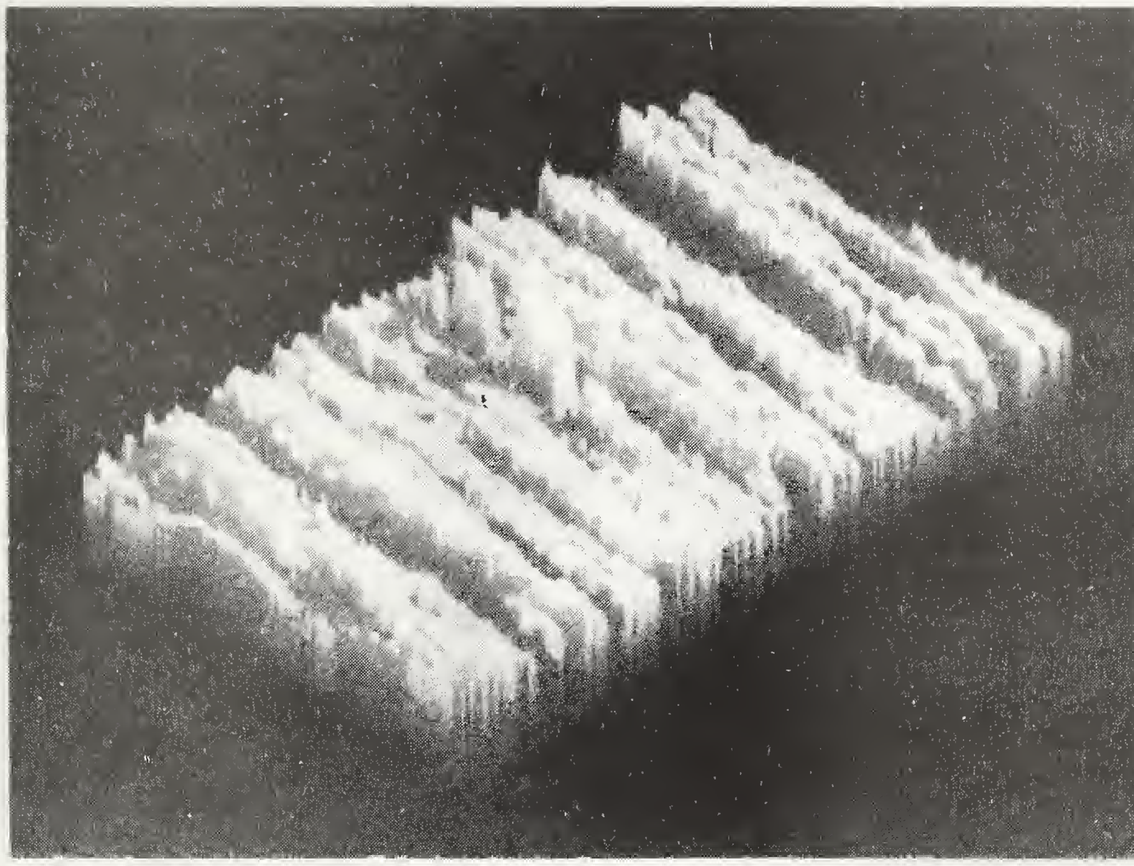


Figure 12

Figure 11 Rotated 180 Degrees

3.0 Three-Dimensional Geometric Considerations

In this section, the necessary vector geometry techniques will be described. The mathematical tools developed will be used in the graphics algorithms to locate points in three-dimensional space in such a way that they can be uniquely traced to points on a viewer's screen. This requires defining special coordinate systems and linking them properly.

3.1 World Coordinate System

The application or user oriented coordinates are generally referred to as world coordinates. The world coordinate system in this application will be a right-handed three-dimensional Cartesian coordinate system. For a surface image the world coordinate system will be placed so that if a person were looking straight down on the top of the surface as in Figure 2 the origin would appear in the upper left hand corner. The positive world coordinate X-axis would then point vertically downwards and the positive world-coordinate Y-axis would point horizontally to the right. The positive world-coordinate Z-axis would point directly at the viewer. See Figure 13. The Z-axis units represent digitized intensity levels of 0-255, lower values represent low intensities, the XY-coordinate ranges are 0-511.

3.2 Device Coordinate Space

The user of image processors must be aware of their device's specific coordinate system. Thus, for example, in the image processor used the coordinate system used on the device reverses

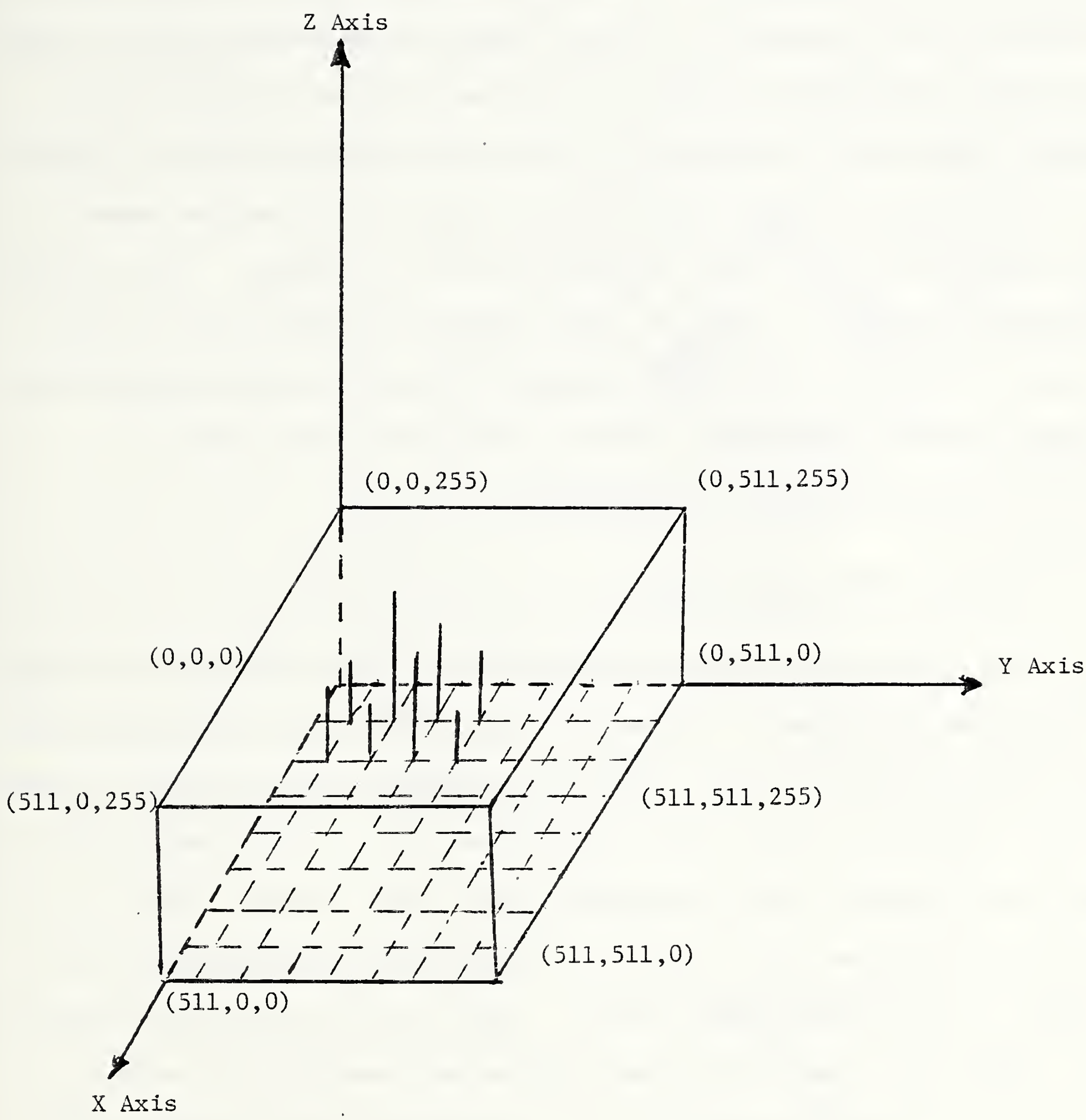


Figure 13

World Coordinate System

the X and Y axes so that the device X-axis is the world coordinate Y-axis and the device Y-axis is the world coordinate X-axis. This confusion is overcome sometimes by calling points along the world coordinate Y-axis sample indices and points along the world coordinate X-axis as traverse indices. This device system comes about because the image processor performs a raster scan from the left to right and top down of the refresh memory, the same way a television screen picture is scanned. This device coordinate system is used in many graphics systems and can lead to some confusion. We shall attempt to use the world coordinate system defined throughout and note the differences when explaining the software references.

3.3 Viewer Coordinate System

The general approach to generating a three-dimensional image used in this program is to define a portion of three-dimensional space and project it onto the viewing screen. The viewing screen can be thought of as a window to the world. A two-dimensional coordinate system can be constructed on this window. The coordinate system that identifies points on this window will be called the V-H coordinate system. With the V-H system defined on the viewplane, specifying the minimum and maximum V-H values defines the viewing window in the viewplane. The viewplane is orthogonal to the viewing rays to the surface. Viewing rays can be thought of as lines along which viewers sight as they look at an object. The portion of the world projected onto the window is called the view volume. In the present case, since orthogonal

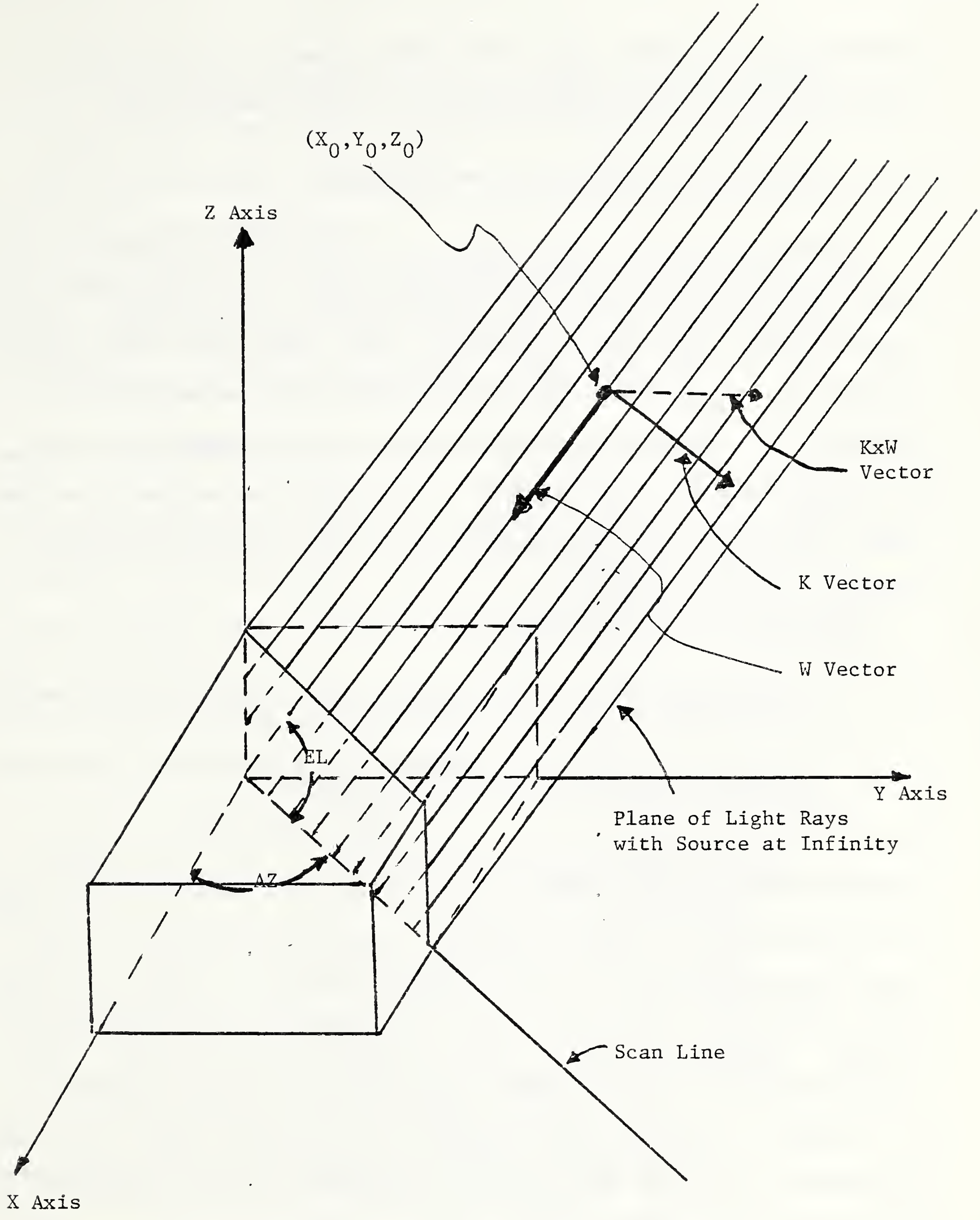


Figure 14

Light Ray Coordinates

projections are being used, the view volume is an infinite parallelepiped with sides parallel to the viewing rays.

3.4 Indexing the Light and Viewing Rays

Assume that the light source is a point at infinity and all rays impinging on the surface are parallel. See Figure 14 for an illustration. Let the direction of the light source be given by two angles, an azimuth and an elevation. The azimuth AZ is measured in a positive sense beginning at the positive X axis. It ranges from 0 to 360 degrees. The elevation angle EL of the light source is measured upwards from the XY-plane and falls between 0 and 90 degrees.

Now set up two unit vectors:

1) $\vec{W}$ - This unit vector points along the light rays and toward the origin.

2) $\vec{K}$ - This unit vector is orthogonal to $\vec{W}$ and points downwards across a plane made of light rays.

We use the same terminology as that used for the light source because the algorithm used is essentially the same for the light and viewing rays. Given an azimuth and an elevation for the light source one can think of a plane formed by rotating the XZ-plane by the azimuth angle. Now fill up this plane with light rays that point in the direction of the W-vector. Consider a unit vector in this plane, called $\vec{K}$, orthogonal to $\vec{W}$. $\vec{K}$ is then orthogonal to all of the light rays pointing in the direction $\vec{W}$. Each point on a fixed light ray in the plane can be indexed from a fixed point on the light ray by adding some multiple of the vector $\vec{W}$. Each light ray can be indexed from a fixed point on the plane by adding some multiple of $\vec{K}$. Finally, all light rays in the direction $\vec{W}$ fall on some plane parallel to the rotated plane. If one takes the cross product of $\vec{K}$ and $\vec{W}$ one gets a vector that can be used to access any plane parallel to the rotated plane, as in Figure 14.

This same procedure can be used to define viewing rays. In this latter case, $\vec{K}$ and $\vec{K}x\vec{W}$ index points on the viewing plane. This is orthogonal to the viewing rays, indexed by $\vec{W}$.

3.5 Vector Representations of the Ray Vector System

Let CE be the cosine of the elevation angle, CA the cosine of the azimuth, SE the sine of the elevation angle, and SA the sine of the azimuth angle. Then $\vec{W}$, $\vec{K}$, and $\vec{K}x\vec{W}$ can be represented in vector triple form as

$$\vec{W} = (-CE*CA,\ -CE*SA,\ -SE)$$

$$\vec{K} = (CA*SE,\ SA*SE,\ -CE)$$

$$\vec{K}x\vec{W} = (-SA,\ CA,\ 0)$$

where * is multiplication. These are developed as follows:

1) Refer to Figure 15 for $\vec{W}$. From simple formulas the distance from A to B is -SE since $\vec{W}$ has unit length. The magnitude of the length from A to O is CE. Then the length from D to A is -CE*SA and from C to A is -CE*CA. The components of $\vec{W}$ are then (-CE*CA, -CE*SA, -SE). Note that $\vec{W}$ as constructed is a unit vector since

$$\begin{aligned}(-CE*CA)^2+(-CE*SA)^2+(-SE)^2 &= CE^2(CA^2+SA^2)+SE^2 \\ &= CE^2+SE^2 \\ &= 1.\end{aligned}$$

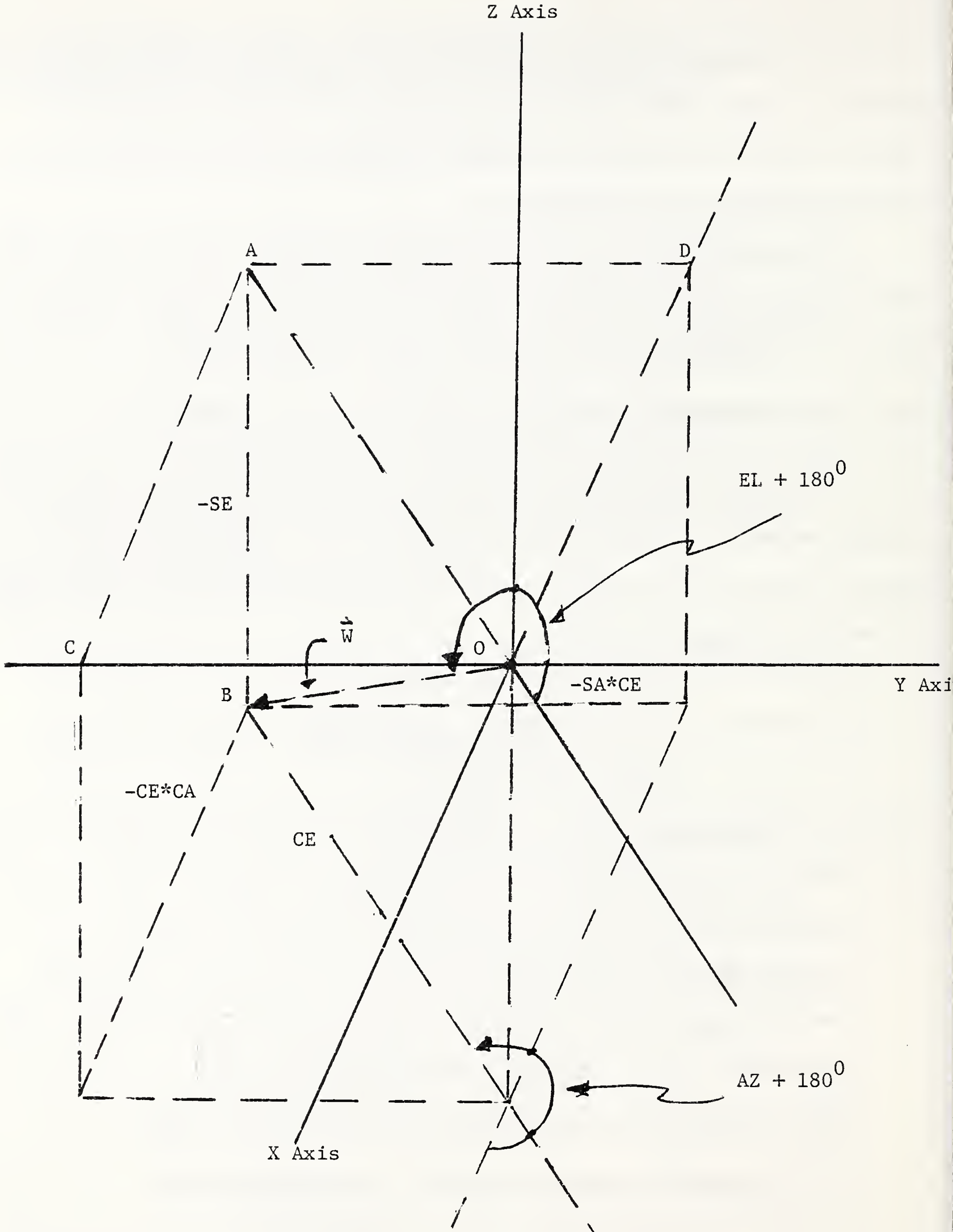


Figure 15

Coordinate Representation
for W Vector

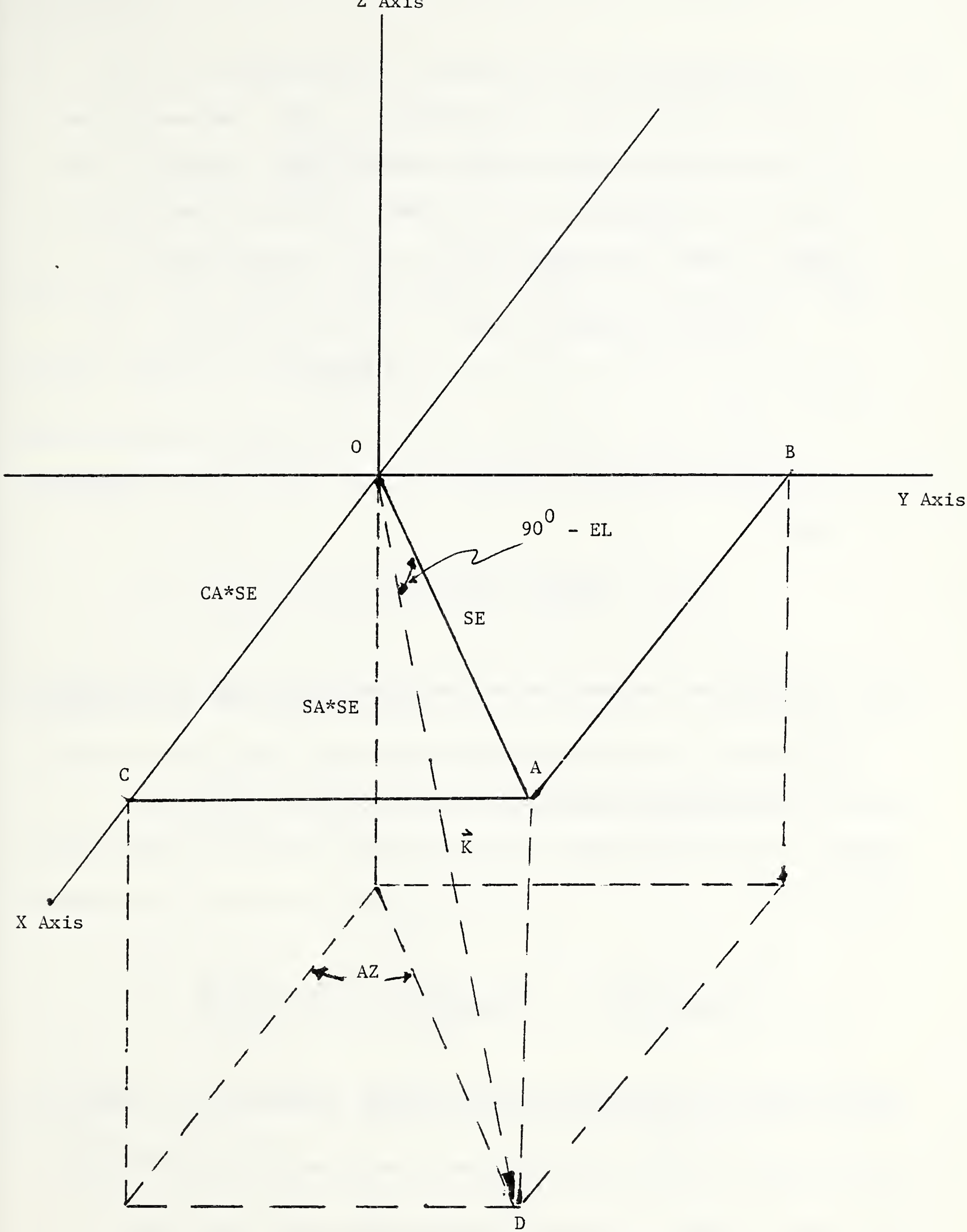


Figure 16

Coordinate Representation
for K Vector

2) The development for $\vec{K}$ is similar. See Figure 16. The distance from A to D = −sin(90°−EL) = −CE. The magnitude of the distance from O to A is cos(90°−EL) = sin(EL) = SE. Then the X coordinate of $\vec{K}$ is CA*SE and the Y coordinate is SA*SE. Again $\vec{K}$ as specified is a unit vector since

$$\begin{aligned}(CA*SE)^2+(SA*SE)^2+(-CE)^2 &= SE^2(CA^2+SA^2)+CE^2\\ &= SE^2+CE^2\\ &= 1.\end{aligned}$$

3) The definition of the standard cross product of two vectors yields

$$\vec{K}\times\vec{W} = (-SA,\ CA,\ 0).$$

3.6 The Relation Between World Coordinates and Ray Coordinates

Any point in the three-dimensional world coordinate system can be represented uniquely by two orthonormal systems of vectors. The first system is the ordinary system of coordinates given by

$$\hat{X} = \begin{pmatrix}1\\0\\0\end{pmatrix}, \quad \hat{Y} = \begin{pmatrix}0\\1\\0\end{pmatrix}, \quad \hat{Z} = \begin{pmatrix}0\\0\\1\end{pmatrix}$$

and the other is given by the orthonormal system $\vec{W}$, $\vec{K}$, $\vec{K}\times\vec{W}$.

Given a point $\begin{pmatrix}X\\Y\\Z\end{pmatrix}$ in the standard coordinate system, then one can write uniquely, as long as the origins are identified,

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = R\vec{W} + V\vec{K} + H(\vec{K}x\vec{W})$$

where (R, V, H) are the coordinates of $\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$ in the $\vec{W}$, $\vec{K}$, $\vec{K}x\vec{W}$ system. Given a point $\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$ in the standard coordinate system, one can always compute R, V, H by the simple inner product relations

$$R = \left(\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \vec{W} \right) = X*W(1) + Y*W(2) + Z*W(3) \ ,$$

implemented in subroutine GETR,

$$V = \left(\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \vec{K} \right) = X*K(1) + Y*K(2) + Z*K(3) \ ,$$

implemented in subroutine GETV, and

$$H = \left(\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \vec{K}x\vec{W} \right) = X*((KxW)(1)) + Y*((KxW)(2)) \ ,$$

implemented in subroutine GETH. For the application of R, V, and H, see Figures 17-20.

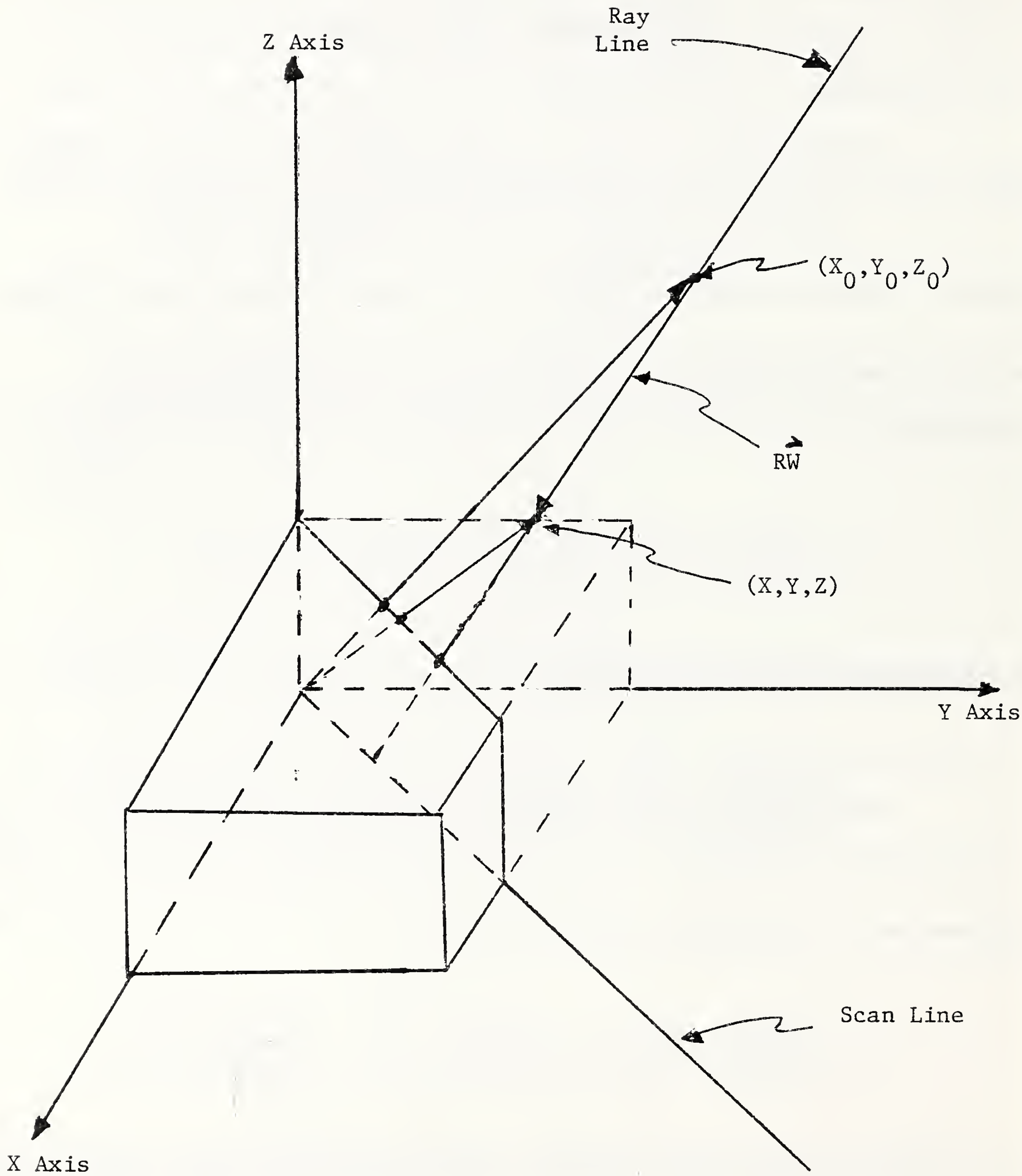


Figure 17

Indexing Along Rays

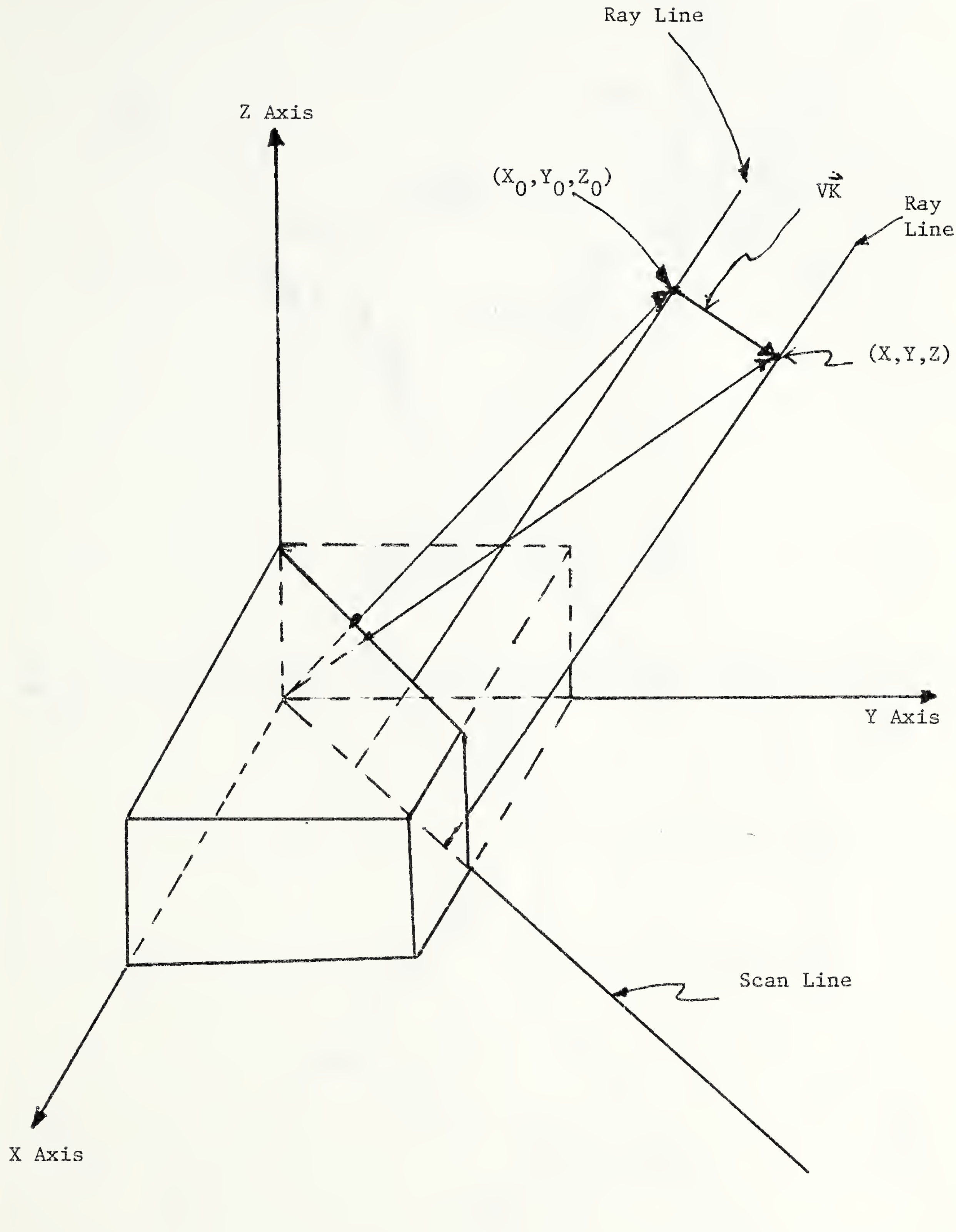


Figure 18

Indexing Different
Rays

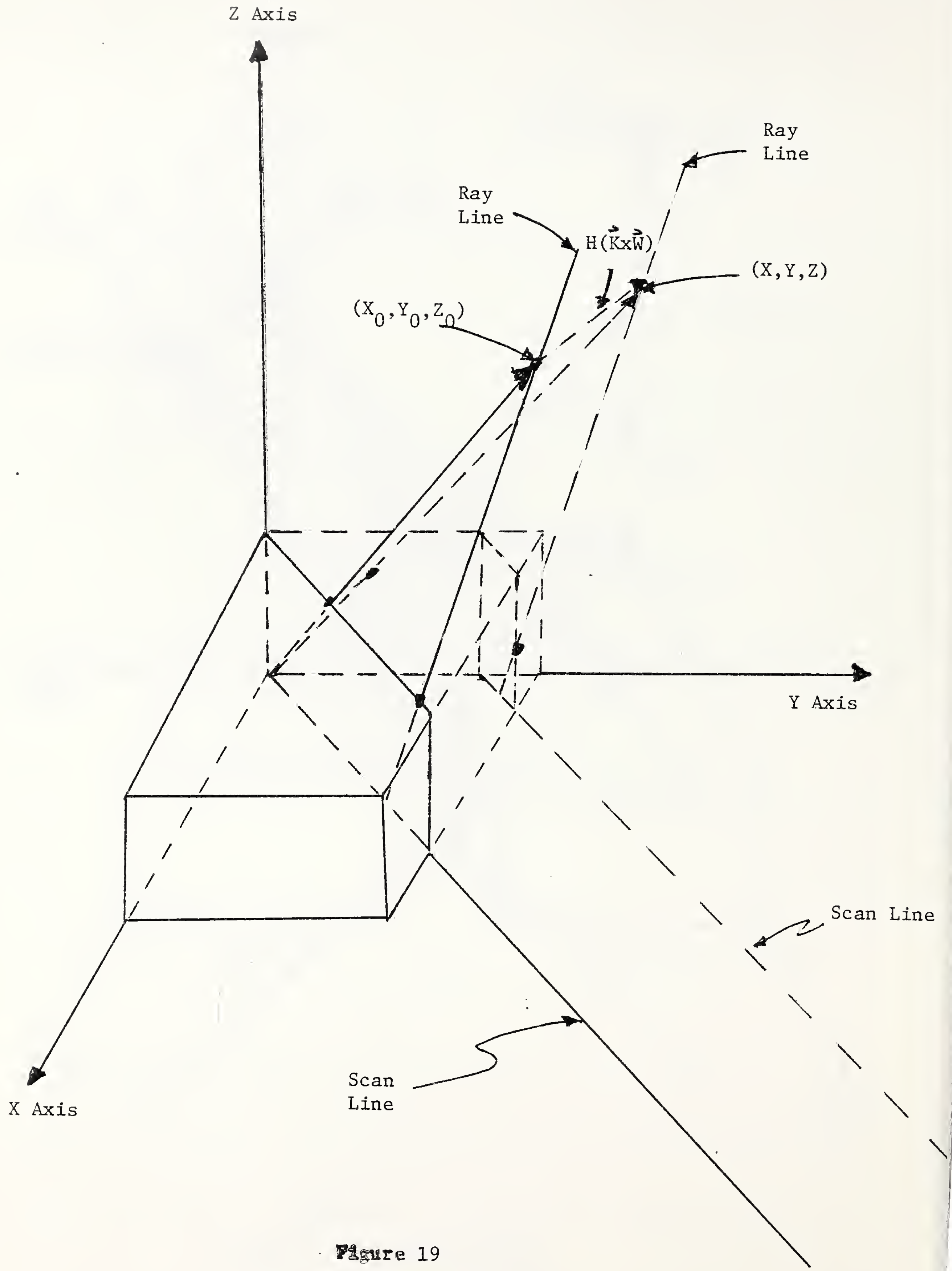


Figure 19

Indexing Different
Ray Planes

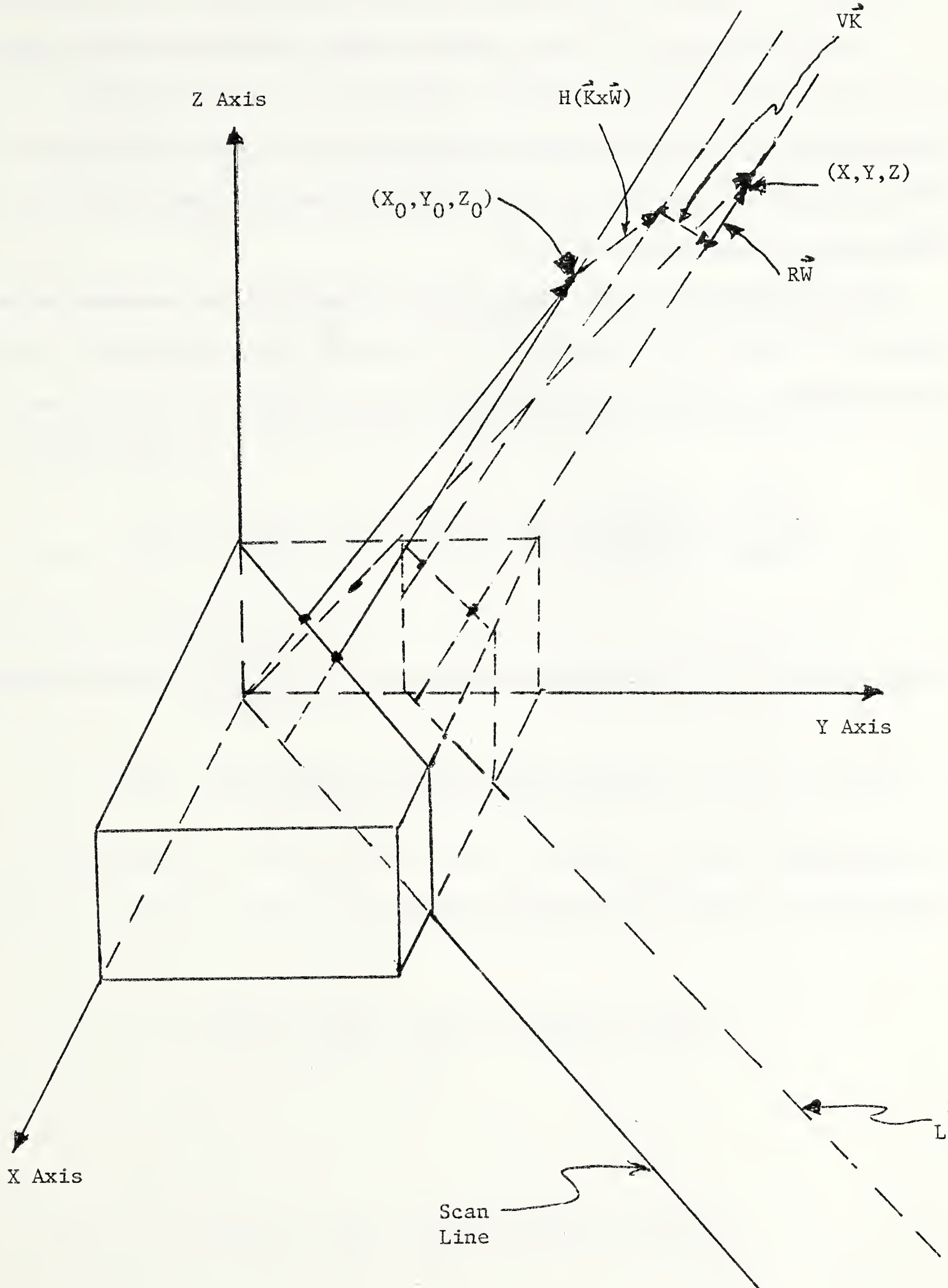


Figure 20

Indexing a Point from Another

3.7 Projection of World Points to the Viewing Window

The coordinates for the viewing space are handled the same as for the light casting space. Points on a viewplane are addressed by the coordinates V and H since the unit vectors $\vec{K}$, $\vec{K} \times \vec{W}$ generate a viewing surface. Points along a viewing ray are addressed by the coordinate R.

For purposes of simplifying, the viewplane is assumed to be placed so that given a point on the screen (V,H), then a corresponding value in world space, can be found by the formula

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_o \\ Y_o \\ Z_o \end{pmatrix} + (V - V_o)\vec{K} + (H - H_o)(\vec{K} \times \vec{W})$$

where V_o, H_o is the viewplane projection of $\begin{pmatrix} X_o \\ Y_o \\ Z_o \end{pmatrix}$. Conversely, if a point $\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$ in world coordinates is specified, then a corresponding row or column in the viewplane can be computed by noting that, since $\vec{K}$ is a unit vector,

$$(V - V_o)\ \vec{K} \cdot \vec{K} = \begin{pmatrix} X - X_o \\ Y - Y_o \\ Z - Z_o \end{pmatrix} \cdot \vec{K}$$

or

$$V = V_o + (X - X_o) \cdot K(1) + (Y - Y_o) \cdot K(2) + (Z - Z_o) \cdot K(3).$$

This formula is implemented in the subroutine GETROW. A similar argument gets the column as

$$H = H_o + (X - X_o) \cdot ((K \times W)(1)) + (Y - Y_o) \cdot ((K \times W)(2)) + (Z - Z_o) \cdot ((K \times W)(3)) .$$

This formula is not needed in the program but is given here for the sake of completeness.

3.8 Conversion from World Coordinates to Light or Viewing Coordinates

Given a point $\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$ in the world coordinates, then it can be uniquely represented by R, V and H in the ray coordinates since

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = R\vec{W} + V\vec{K} + H(\vec{K} \times \vec{W})$$

implies, by taking inner products, that

$$R = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \cdot \vec{W} = XW(1) + YW(2) + ZW(3)$$

$$V = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \cdot \vec{K} = XK(1) + YK(2) + ZK(3)$$

$$H = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \cdot (\vec{K} \times \vec{W}) = X((K \times W)(1)) + Y((K \times W)(2)) .$$

These formulas have been implemented in the subroutines GETR, GETV, GETH, respectively.

3.9 Conversion of a Viewplane Point to a World Coordinate Point

Given a point (V, H) on the viewplane, then

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_o \\ Y_o \\ Z_o \end{pmatrix} + (V - V_o)\vec{K} + (H - H_o)(\vec{K} \times \vec{W})$$

associates the $\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$ value with that point. This formula is implemented in subroutine GETXYZ.

3.10 Computing the Height Along a Ray

Any point has an equivalent representation in the two coordinate systems $\hat{X}$, $\hat{Y}$, $\hat{Z}$ and $\vec{W}$, $\vec{K}$, $\vec{K}\times\vec{W}$. This equivalence can be represented by

$$X \cdot \hat{X} + Y \cdot \hat{Y} + Z \cdot \hat{Z} = R \cdot \vec{W} + V \cdot \vec{K} + H \cdot (\vec{K} \times \vec{W}) .$$

Then, given X, Y and a ray index V , one can compute

$$Z(\hat{Z} \cdot \vec{K}) = V - X(\hat{X} \cdot \vec{K}) - Y(\hat{Y} \cdot \vec{K})$$

$$Z \cdot K(3) = V - X \cdot K(1) - Y \cdot K(2)$$

and finally

$$Z = (1/K(3)) \cdot (V - X \cdot K(1) - Y \cdot K(2)) .$$

See Table 3.1.

Table 3.1

VECTOR REPRESENTATIONS

$$W(1) = \hat{X} \cdot \vec{W} = -CE*CA$$

$$K(1) = \hat{X} \cdot \vec{K} = CA*SE$$

$$(K\times W)(1) = \hat{X} \cdot (\vec{K} \times \vec{W}) = -SA$$

$$W(2) = \hat{Y} \cdot \vec{W} = -CE*SA$$

$$K(2) = \hat{Y} \cdot \vec{K} = SA*SE$$

$$(K\times W)(2) = \hat{Y} \cdot (\vec{K} \times \vec{W}) = CA$$

$$W(3) = \hat{Z} \cdot \vec{W} = -SE$$

$$K(3) = \hat{Z} \cdot \vec{K} = -CE$$

$$(K\times W)(3) = \hat{Z} \cdot (\vec{K} \times \vec{W}) = 0$$

4.0 Discussion of Algorithms

This section covers the broad details of the major algorithms used in this program. The two main algorithms are the shadow graph generation algorithm and the solid projection algorithm. These are supported by two subsidiary algorithms. The first is the entry point selection algorithm which has three components: (1) A case selection look up table, (2) extreme point selection table, and (3) the entry point selection algorithm itself. The second major subsidiary algorithm is the line drawing algorithm. This last algorithm is sometimes referred to in the graphics literature as a scan conversion algorithm.

4.1 Shadowgraph Algorithm

This section describes in step form the major tasks performed by the shadowgraph algorithm as it is implemented in the program.

Step 1: Transfer the data image file from the disk to the first refresh memory of the image processor.

Step 2: Initialize refresh memory 2 of the image processor by blanking it so that the shadowgraph can be created there. This leaves the monitor image all black.

Step 3: Interactively read in the azimuth and elevation angles for the light source.

Step 4: Compute the orthonormal coordinate system $\vec{W}$, $\vec{K}$, $\vec{K} \times \vec{W}$ for the light rays.

Step 5: From the signs of the $\vec{W}$-vector components, look up the current case number.

Step 6: Identify the entire image for shadowing. This is done by specifying the picture vertices as the refresh memory limits.

Step 7: For the current case number given in Step 5, determine the extreme points of the image. See Figure 21 for some examples.

Step 8: Set the first extreme point as the first point on the picture plane that a projection of the light rays onto the plane contacts.

Step 9: Since this point is not in shadow, transfer its pixel value from refresh memory 1 to refresh memory 2 of the image processor.

Step 10: For the current case number get the next boundary or entry point of the image in refresh memory 1 at which a projected $\vec{W}$-vector enters the picture. Set this point as (X,Y). If (X,Y) is the second extreme point, go to step 20.

Step 11: There is no shadowing at this boundary point, since the light ray encounters this point. Transfer the picture intensity value from refresh memory 1 of the image processor to refresh memory 2.

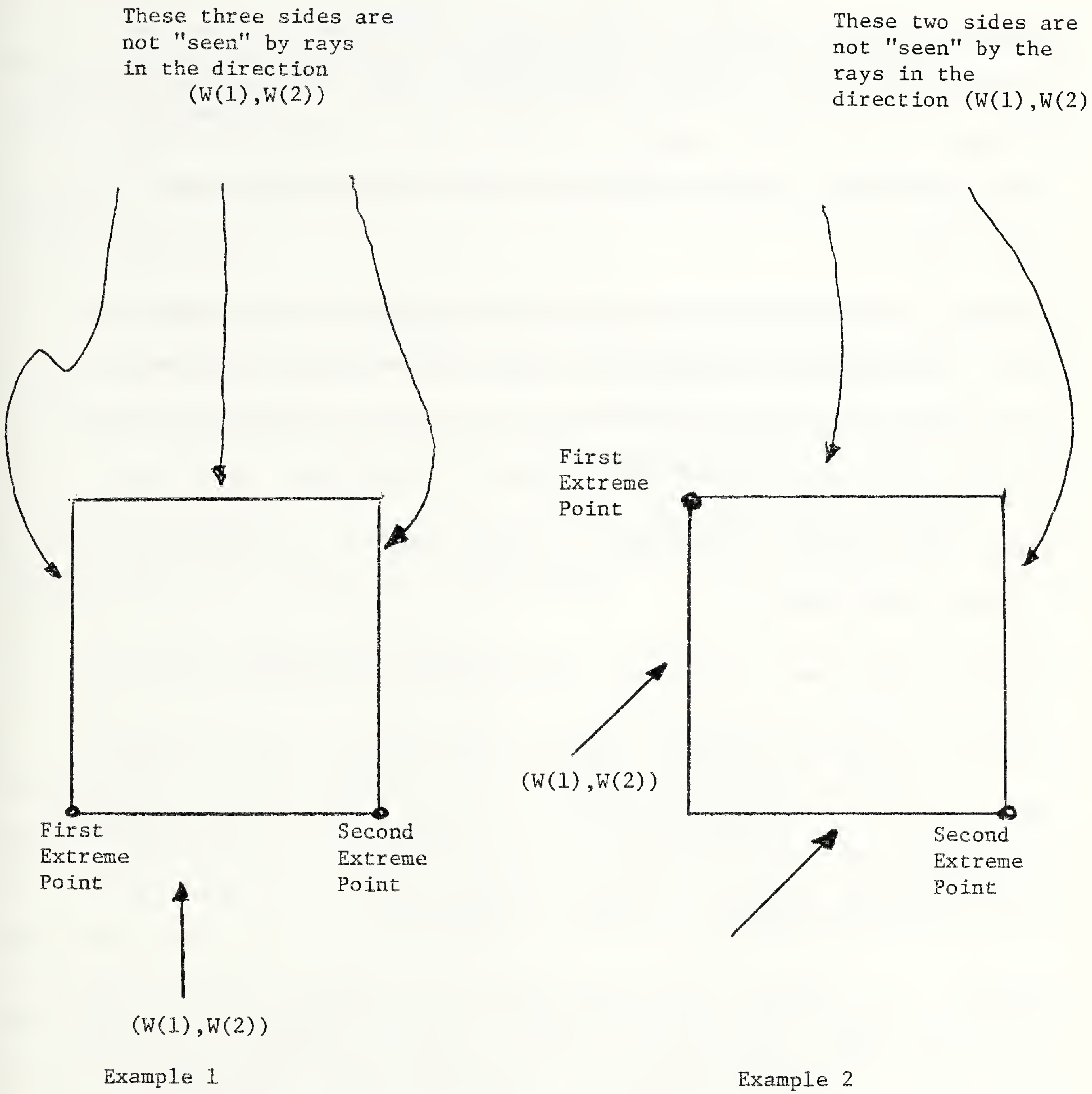


Figure 21

Extreme Point Selection

__Step 12__: Let PICV be the picture value at this boundary point, i.e., the current (X,Y).

__Step 13__: Since there is a unique plane orthogonal to the XY-plane of the image in which the $\vec{W}$-vector lies, compute the unique multiple, V, of the $\vec{K}$ vector, in that plane, which identifies a light ray lying in that plane and passing through the point (X,Y, PICV).

__Step 14__: With the line drawing algorithm generate the next (X,Y) pixel index along the projection of the $\vec{W}$-vector on the image plane. If this point is outside of the picture rectangle, then get the next ray plane by going back to step 10.

__Step 15__: Compute the height of the current ray, indexed by V, and call this value ZT.

__Step 16__: Get the image pixel value, PICV, at the point (X,Y).

__Step 17__: If ZT is greater that the image value, PICV, at the point (X,Y), then the pixel is not visible to this ray. Do not write anything at this pixel in refresh memory 2. Leave the black background there. Go back to step 14.

__Step 18__: If ZT equals the pixel value at the point (X,Y), then write the image value PICV at (X,Y) in refresh memory 1 to the point (X,Y) in refresh memory 2. Since the light ray model assumes that the ray skims the top of a pixel, return to step 14 to generate the next (X,Y).

Step 19: If ZT is less than the pixel value PICV at the current point (X,Y), then the pixel is seen by the ray. Write the pixel value PICV from refresh memory 1 to refresh memory 2 at (X,Y). Get the new index V of the ray that goes through (X, Y, PICV). Return to step 14.

Step 20: Save the shadowgraph as an indexed file of 512 records of 512 bytes each.

An example of a shadowgraph was given previously in Figure 5. From the viewer's perspective, both the image and shadowgraph appear as if one were looking vertically downwards at the scene. The orthogonal projection of a world coordinate in the (X,Y) plane translates to the same point on the screen, but in screen coordinates the Y and X are interchanged.

4.2 The Solid Projection Algorithm

Before beginning this algorithm, the image file must be loaded into refresh memory 1 of the image processor and the shadowgraph must also be loaded into refresh memory 2. Furthermore, the contents of refresh memory 2 must be visible on the display monitor. The program steps are as follows:

Step 1: Initialize the cursor and turn it on in order to interactively specify pixel points in refresh memory 2.

Step 2: Use the trackball cursor to identify two diametrically opposite points of a rectangle of interest in the shadowgraph. This rectangle will be the area converted to a three-dimensional image.

__Step 3__: Set up the corner vertices so that the upper left is indexed by (1,1). The indexing proceeds counterclockwise from (1,1) to (2,1) to (2,2) to (1,2). See Figure 22.

__Step 4__: Identify the center of the rectangle of interest as (XO,YO) and let ZO = 128, which is the midpoint of the intensity levels that run from 0 to 255.

__Step 5__: Draw lines around the rectangle of interest and place a mark at the center. If the viewer does not like this region, return to Step 1, otherwise continue.

__Step 6__: Turn off the cursor and initialize a third refresh memory of the image processor for solid projection image.

__Step 7__: Interactively get the azimuth and elevation angles for the viewing plane and the percent reduction for shadowing.

__Step 8__: Compute the orthonormal vectors for the viewing rays $\vec{W}$, $\vec{K}$ and $\vec{K}x\vec{W}$.

__Step 9__: Get the case number for $\vec{W}$.

__Step 10__: Get the extreme points of the shadowgraph rectangle.

__Step 11__: Set up the first extreme point at the first entry point of the projection of the viewing ray $\vec{W}$-vector onto the XY-plane.

__Step 12__: Compute the H multiples of the $\vec{K}x\vec{W}$ unit vector that yield the vertical ray planes passing through the two extreme points. These planes form the left and right bounds for the viewing window. Designate the first as HMIN and the second as HMAX. The

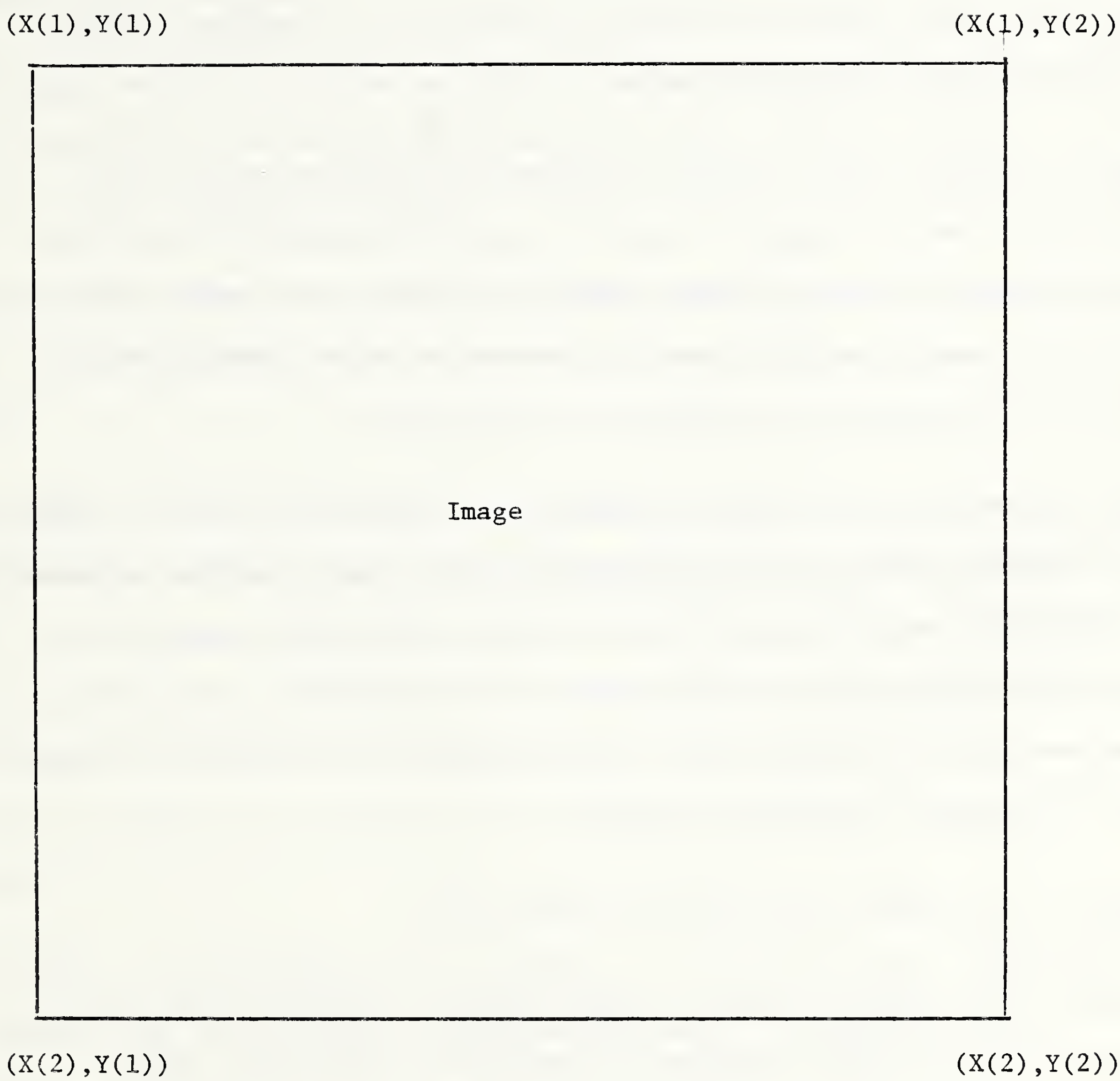


Figure 22

Extreme Point Indexing

solid is generated by vertical scans on the viewing window moving from left to right after each vertical scan.

Step 13: Set up the viewport, or monitor screen center, as the projection of the center of the rectangle of interest.

Step 14: Modify HMIN and HMAX to conform to a viewport that is the smallest to bound the solid.

Step 15: Since the algorithm proceeds by selecting each ray plane from left to right, tracing rays through each pixel from bottom to top in a ray plane, begin by setting H = HMIN. Get the starting viewport column for this H and set the starting row as 511 which represents the bottom of the screen.

Step 16: On the first pass through this step, the first extreme point is designated as the beginning entry point to the rectangle of interest, but the algorithm picks the next entry point to begin. If the second extreme point is encountered, stop the algorithm and go to Step 27. Set the entry point as the current world coordinate point of interest.

Step 17: Increment the column counter by 1.

Step 18: Get the row index on the viewing window of the entry point (X,Y,0) in the XY-plane of the world coordinate space.

Step 19: Once the row and column indices have been selected on the viewing plane, specify this as the current screen point.

Step 20: Get the corresponding world coordinate point for the current screen point, i.e. row and column, on the viewplane. Note that this is not the same world coordinate point as (X,Y,0).

Step 21: Get the ray index V of the ray through this viewplane world coordinate point.

Step 22: Get the Z value on the ray indexed by V at the current (X,Y) point on the world coordinate XY-plane. For an entry point, this Z value will be 0.

Step 23: Get the pixel value from the image in refresh memory 1 and the shadowgraph value from refresh memory 2 for the (X,Y) point.

Step 24: If the ray height ZT is greater than the image value PICV at the current (X,Y) point on the plane, then the ray does not see the pixel. Generate the next (X,Y) point along the ray projection on the XY-plane. Test whether it remains within the rectangle of interest. If it does, go back to Step 22, otherwise go back to Step 16 to move to the next ray plane or screen column.

Step 25: If the height ZT is equal to the pixel value, the pixel is seen. The current ray is not continued. A new ray is generated through the next screen point above it and tracing continues. This is done by first writing the pixel value from the original image in refresh memory 1 to the viewport refresh memory, i.e. refresh memory 3, at the projected (V,H) coordinate. Reduce the pixel value by the percent required for shadowing if the intensity value on the shadowgraph in refresh memory 2 is 0 at that pixel. Move up one pixel in the viewport column and get the associated world coordinate (X,Y,Z) for this point on the viewplane. Get the V index for the ray through this point. Generate the next (X,Y) point along the projected $\vec{W}$ vector line on the (X,Y) plane. Go back to Step 22.

Step 26: If ZT is less than the pixel value at the current (X,Y), then the pixel is seen by the ray. If the intensity of the associated pixel in refresh memory 2 is 0, then this indicates that the point is in shadow. Write out to refresh memory 3 the height ZT reduced, if necessary, by the percent specified if shadowing is indicated. Decrement the row index to move up one row. Get the (X,Y,Z) world coordinate that is equivalent to the new viewplane point. Get the V index for the ray through this point. Go back to Step 22.

Step 27: Write out the solid image to the disk if desired.

4.3 Case Selection Table

Table 4.1 specifies a case index that can be referenced by other subroutines in the program. It distinguishes each possible case combination of the first two components of the $\vec{W}$-vector that points along the rays from either the light source or viewer towards the origin.

4.4 Extreme Point Selection Table

Depending on the direction vector (W(1), W(2)) along the base plane of the solid, this table specifies the first and the

Table 4.1

$\vec{W}$-VECTOR CASES

Case	Index
W(1) = 0, W(2) = 0	1
W(1) = 0, W(2) > 0	2
W(1) = 0, W(2) < 0	3
W(1) > 0, W(2) = 0	4
W(1) > 0, W(2) > 0	5
W(1) > 0, W(2) < 0	6
W(1) < 0, W(2) = 0	7
W(1) < 0, W(2) > 0	8
W(1) < 0, W(2) < 0	9

Table 4.2

EXTREME POINT TABLE

Case	Extreme Point 1	Extreme Point 2
1. W(1) = 0, W(2) = 0	Flag Returned	
2. W(1) = 0, W(2) > 0	(X(1), Y(1))	(X(2), Y(1))
3. W(1) = 0, W(2) < 0	(X(2), Y(2))	(X(1), Y(2))
4. W(1) > 0, W(2) = 0	(X(1), Y(2))	(X(1), Y(1))
5. W(1) > 0, W(2) > 0	(X(1), Y(2))	(X(2), Y(1))
6. W(1) > 0, W(2) < 0	(X(2), Y(2))	(X(1), Y(1))
7. W(1) < 0, W(2) = 0	(X(2), Y(1))	(X(2), Y(2))
8. W(1) < 0, W(2) > 0	(X(1), Y(1))	(X(2), Y(2))
9. W(1) < 0, W(2) < 0	(X(2), Y(1))	(X(1), Y(2))

last base point of the boundary rectangle encountered by the rays. See Figure 21 for an illustration.

Let the four vertices be labeled with X and Y components as shown in Figure 22: (X(1), Y(1)) is the upper left corner, (X(2), Y(1)) is the lower left corner, (X(1), Y(2)) is the upper right corner and (X(2), Y(2)) is the lower right corner. For each case there are two extreme points. These are detailed in Table 4.2.

4.5 Entry Point Algorithm

By an entry point is meant a point on the boundary of a base rectangle through which the projection of a ray in space onto the XY-plane passes as it traverses across the base rectangle. See Figure 4 for an illustration. This algorithm begins with the assumption that there is a current entry point. The algorithm returns the next entry point or a flag if an extreme point is encountered. Let IXIN, IYIN be the current entry point. The algorithm is a case-by-case analysis.

<u>Case 1</u>: W(1) = W(2) = 0. Return a flag.

<u>Case 2</u>: W(1) = 0, W(2) > 0. Beginning with extreme point (X(1),Y(1)), set IXIN = X(1), IYIN = Y(1). The new entry point is then defined by IXIN = IXIN+1, IYIN = IYIN. This case terminates when IXIN-X(2) = 0 and IYIN-Y(1) = 0.

<u>Case 3</u>: W(1) = 0, W(2) < 0. Beginning with the first extreme point IXIN = X(2), IYIN = Y(2), set the next entry point as IXIN = IXIN-1, IYIN = IYIN and stop when IXIN-X(1) = IYIN-Y(2) = 0.

Case 4: W(1) > 0, W(2) = 0. Begin with IXIN = X(1), IYIN = Y(2). Set the next entry point as IXIN = IXIN, IYIN = IYIN-1. Stop when IXIN-X(1) = IYIN-Y(1) = 0.

Case 5: W(1) > 0, W(2) > 0. Begin with IXIN = X(1), IYIN = Y(2). Set the next entry point as IXIN = IXIN, IYIN = IYIN-1 until IXIN-X(1) = IYIN-Y(1) = 0. Then set the next entry point as IXIN = IXIN+1, IYIN = IYIN. Stop when IXIN-X(2) = IYIN-Y(1) = 0.

Case 6: W(1) > 0, W(2) < 0. Begin with IXIN = X(2), IYIN = Y(2). Set the next entry point to IXIN = IXIN-1, IYIN = IYIN until IXIN-X(1) = IYIN-Y(2) = 0. Then set the next entry point to IXIN = IXIN, IYIN = IYIN-1. Stop when IXIN-X(1) = IYIN-Y(1) = 0.

Case 7: W(1) < 0, W(2) = 0. Begin with IXIN = X(2), IYIN = Y(1). Set the next entry point to IXIN - IXIN, IYIN - IYIN+1. Stop when IXIN-X(2) = IYIN-Y(2) = 0.

Case 8: W(1) < 0, W(2) > 0. Begin with IXIN = X(1), IYIN = Y(1). Set the next entry point IXIN = IXIN+1, IYIN - IYIN until IXIN-X(2) = IYIN-Y(1) = 0. Then set the next entry point to IXIN = IXIN, IYIN = IYIN+1. Stop when IXIN-X(2) = IYIN-Y(2) = 0.

Case 9: W(1) < 0, W(2) < 0. Begin with IXIN = X(2), IYIN = Y(1). Set the next entry point to IXIN = IXIN, IYIN - IYIN+1 until IXIN-X(2) = IYIN-Y(2) = 0. Then set the next entry point to IXIN = IXIN-1, IYIN = IYIN. Stop when IXIN-X(1) = IYIN-Y(2) = 0.

4.6 Line Drawing Algorithm

The task of a line drawing algorithm is to compute the coordinates of the pixels that lie near a line on a two-dimensional raster grid in such a manner that when the pixels are strung together, they approximate the straight line (see Figure 22). There are several such algorithms in the literature and they are sometimes referred to as scan-conversion algorithms. Ordinarily, the algorithms are applied to the problem in which two endpoints of the line are specified. In the present case, an algorithm will be presented in which the starting value and the direction vector of the line are given. The problem then is to start from a point on the line and generate the next pixel along the line. The pixels chosen are based on integer truncation rather than rounding.

Assume that a point (X,Y) is given and let (IX,IY) be the point composed of the integer truncated values of X and Y. This point will be referred to as the current pixel. Furthermore, suppose that a direction vector in the XY-plane has been given by (W(1),W(2)).

<u>Case 1</u>: W(1) = W(2) = 0

RETURN Error flag.

<u>Case 2</u>: W(1) = 0, W(2) > 0

```
IF  Y > = 0,  RETURN  (IX,IY+1).
IF  Y < 0
  AND IF  IY > Y,  RETURN  (IX,IY);
  OTHERWISE IF  IY = Y,  RETURN  (IX,IY+1).
```

Case 3: W(1) = 0, W(2) < 0

```
IF  Y >= 0
  AND IF  IY > Y,  RETURN  (IX,IY);
  OTHERWISE IF  IY = Y,  RETURN  (IX,IY-1).
IF  Y < 0,  RETURN  (IX,IY-1).
```

Case 4: W(1) > 0, W(2) = 0

```
IF  X >= 0,  RETURN  (IX+1,IY).
IF  X < 0
  AND IF  IX > X,  RETURN  (IX,IY);
  OTHERWISE IF  IX = X,  RETURN  (IX+1,IY).
```

Case 5: W(1) > 0, W(2) > 0

```
LET  SLOPE = W(2)/W(1).
IF  X >= 0,  LET  XT = IX+1.
IF  X < 0
  AND IF  IX > X,  LET  XT = IX;
  OTHERWISE IF  IX = X,  LET  XT = IX+1.
LET  YT = SLOPE * (XT-X) + Y.
IF  Y >= 0  AND  IY =< YT =< IY+1,
  RETURN  (IXT,IYT)
  WHERE  IXT = INTEGER TRUNCATED XT
         IYT = INTEGER TRUNCATED YT.
IF  Y >= 0  AND  IY+1 < YT,
  LET  YT = IY+1,
  LET  XT = (1./SLOPE) * (YT-Y) + X,
  TRUNCATE  XT  TO  IXT,
  TRUNCATE  YT  TO  IYT,
  RETURN  (IXT,IYT).
```

```
IF  Y < 0  AND  IY > Y  AND
  IF  IY >= YT >= IY-1,  RETURN  (IXT,IYT).
  IF NOT,
       LET  YT = IY,
       LET  XT = (1./SLOPE) * (YT-Y) + X,
       TRUNCATE  XT  TO  IXT,
       TRUNCATE  YT  TO  IYT,
       RETURN  (IXT,IYT).
IF  Y < 0  AND  IY = Y  AND
  IF  IY =< YT =< IY+1,  RETURN  (IXT,IYT).
  IF NOT,
       LET  YT = IY+1,
       LET  XT = (1./SLOPE) * (YT-Y) + X,
       TRUNCATE  XT  TO  IXT,
       TRUNCATE  YT  TO  IYT,
       RETURN  (IXT,IYT).
```

<u>Case 6</u>: W(1) > 0, W(2) < 0

```
LET  SLOPE = W(2)/W(1).
IF  X >= 0,  LET  XT = IX+1.
IF  X < 0
  AND IF  IX > X,  LET  XT = IX;
  OTHERWISE IF  IX = X,  LET  XT = IX+1.
LET  YT = SLOPE * (XT-X) + Y.
IF  Y > 0  AND  Y > IY  AND
  IF  IY =< YT =< IY+1,  RETURN  (IXT,IYT).
IF  Y > 0  AND  YT < IY  AND
  IF  IY < Y,  LET  YT = IY,
```

```
        LET  XT = (1./SLOPE) * (YT-Y) + X,
        RETURN  (IXT,IYT).
    IF  IY = Y,  LET  YT = IY-1,
        LET  XT = (1./SLOPE) * (YT-Y) + X,
        RETURN  (IXT,IYT).
  IF  Y < 0  AND
    IF  IY >= YT >= IY-1,  RETURN  (IXT,IYT).
    IF NOT,
        LET  YT = IY-1,
        LET  XT = (1./SLOPE) * (YT-Y) + X,
        RETURN  (IXT,IYT).
```

<u>Case 7</u>: W(1) < 0, W(2) = 0

```
  LET  SLOPE = W(2)/W(1) = 0.
  IF  X >= 0  AND
    IF  X > IX,  LET  XT - IX;
    IF  X = IX,  LET  XT = IX-1.
  IF  X < 0,  LET  XT = IX-1.
    LET  YT = SLOPE * (XT-X) +  Y,
    RETURN  (IXT,IYT).
```

<u>Case 8</u>: W(1) < 0, W(2) > 0

```
  LET  SLOPE = W(2)/W(1).
  IF  X >= 0  AND
    IF  X > IX,  LET  XT = IX,
    IF  X = IX,  LET  XT = IX-1.
  IF  X < 0,  LET  XT = IX-1.
    LET  YT = SLOPE * (XT-X) + Y.
```

```
        IF  Y >= 0  AND
          IF  IY = < YT =< IY+1,
          RETURN  (IXT,IYT);
          OTHERWISE IF  IY < Y,  LET  YT = IY.
          LET  XT = (1./SLOPE) * (YT-Y) + X,
          RETURN  (IXT,IYT).
        IF  Y < 0,  LET  YT = IY-1.
          LET  XT = (1./SLOPE) * (YT-Y) + X,
          RETURN  (IXT,IYT).
```

Case 9: W(1) < 0, W(2) < 0

```
        LET SLOPE = W(2)/W(1).
        IF  X >= 0  AND
          IF  X > IX,  LET  XT = IX.
          IF  X = IX,  LET  XT = IX-1.
        IF  X < 0,  LET  XT = IX-1;
          LET  YT = SLOPE * (XT-X) + Y.
        IF  Y >= 0  AND
          IF  IY < Y  AND  IY <= YT <= IY+1,
          RETURN  (IXT,IYT).
          OTHERWISE LET  YT = IY,
          LET  XT = (1./SLOPE) * (YT-Y) + X,
          RETURN  (IXT,IYT).
        IF  IY = Y  AND  IY >= YT >= IY-1,
          RETURN  (IXT,IYT).
          OTHERWISE  LET  YT = IY-1,
          LET  XT = (1./SLOPE) * (YT-Y) + X,
          RETURN  (IXT,IYT).
```

```
IF  Y < 0  AND

  IF  IY >= YT >= IY-1,

  RETURN  (IXT,IYT).

  OTHERWISE LET  YT = IY-1,

  LET  XT = (1./SLOPE) * (YT-Y) + X,

  RETURN  (IXT,IYT).
```

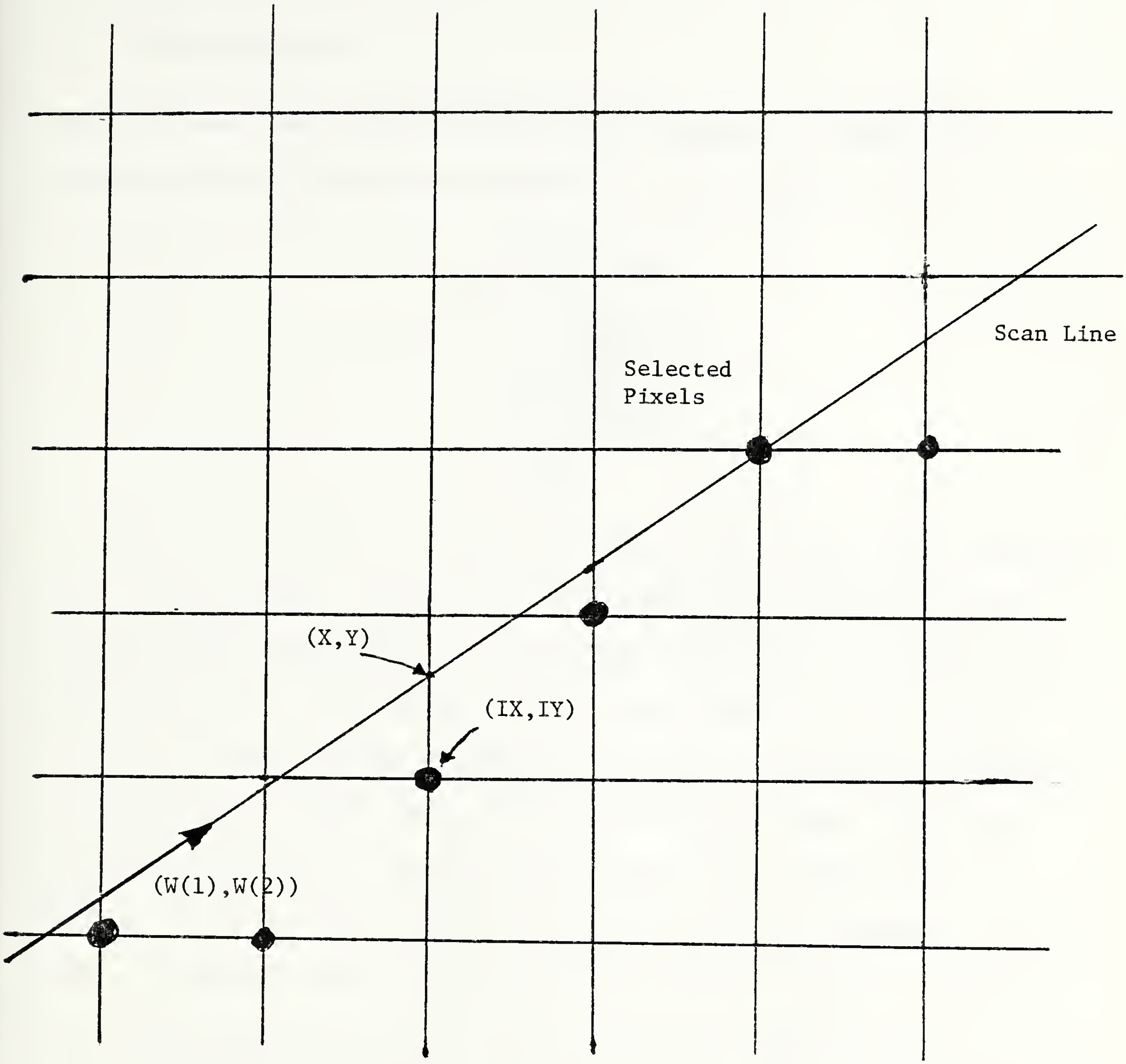


Figure 23

Scan Line Conversion to Pixels

5.0 Program Implementation

5.1 System Commands

When the user types SOLID on the console to begin the program, the host system transfers to the following file:

```
*
*SOLID.CSS
*
L .BG,SOLID
T . BG
AS 3,12S:
AS 5,C:
AS 6,NULL:
ST
$EXIT
```

The first three lines are comments identifying this command file as SOLID.CSS. The fourth line loads the linked task with the name SOLID and gives it the system designated name .BG for a background job if the multiterminal environment is not active. If the multiterminal environment is active, the system identifies the job with the user name entered at sign-on time. The next line identifies any following assignments with the task just loaded. The next three lines assign logical unit number 3 to the image processor, known to the operating system by the mnemonic I2S:, logical unit 5 to the user's terminal and logical unit 6 to a null device. This means that the logical unit 6 is assigned to the task, but any input/output through it will be ignored. This is inserted so that the user could assign logical unit 6 to an input/output unit for program error analysis at a later time, if necessary. The next to the last line starts the designated task and the final line exits to the user console at program termination.

5.2 Main Program

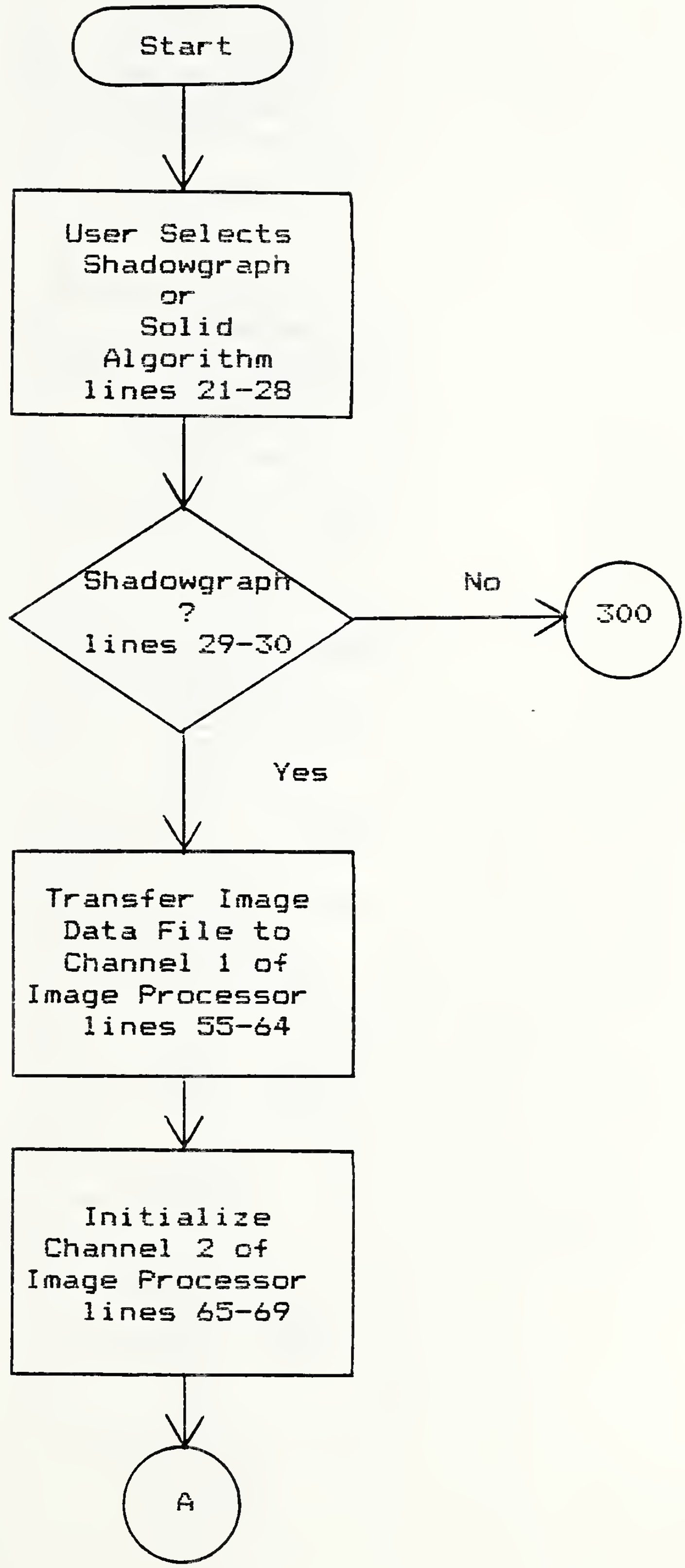
5.2.1 Summary

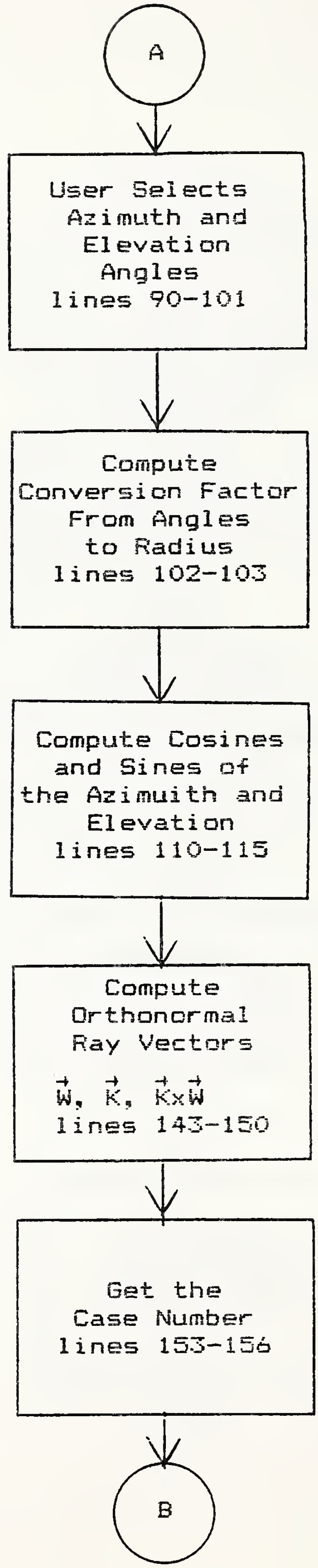
This subsection contains the flow chart and listing of the main program. It implements both the shadowgraph algorithm and solid generation algorithm. The user selects which algorithm to use interactively.

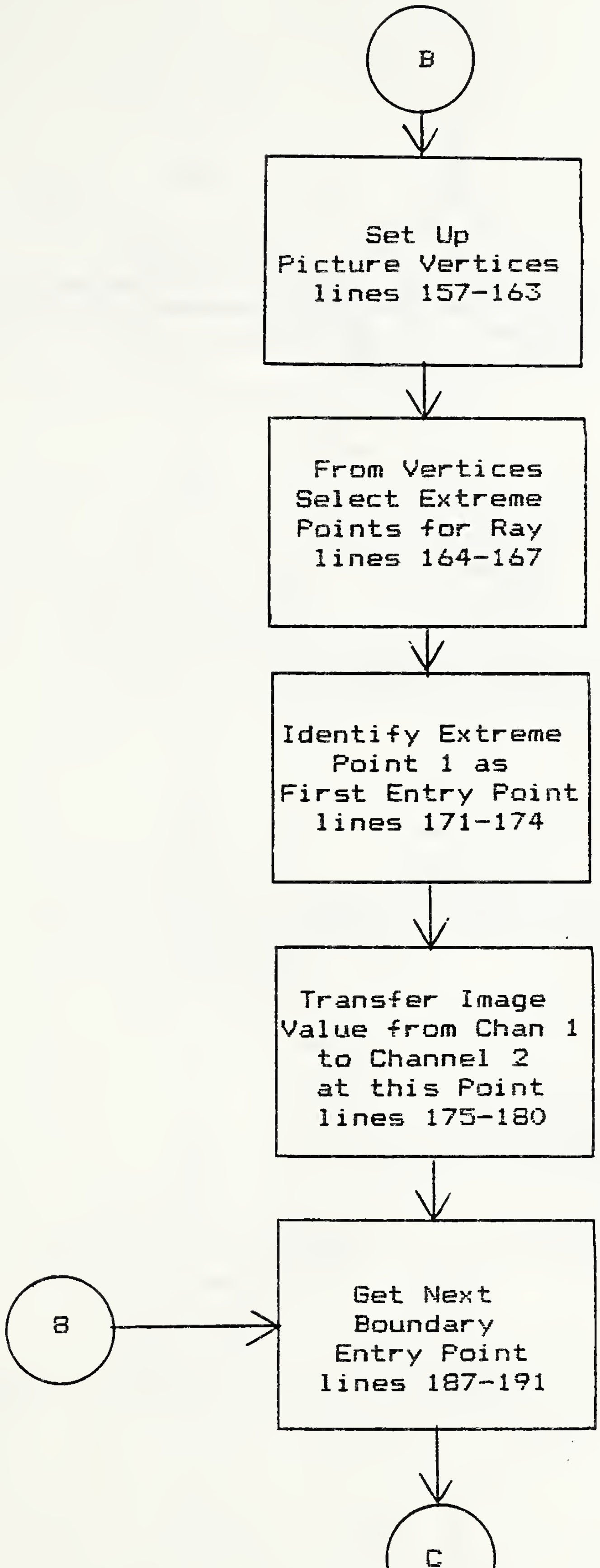
In the shadowgraph algorithm, the program traces individual rays from the light source to the surface. The light source is located at a specified azimuth and elevation angle, selected interactively by the user. As each ray is traced to the surface, the height along the ray is either greater than, equal to, or less than a pixel height representing a topographic amplitude. If the ray height is greater than the pixel value, that pixel is not seen by the ray and falls into shadow relative to the ray. If the ray height is equal to the pixel height, then the pixel is seen and the ray is continued as well. If the ray has a height less than the pixel, the pixel is seen and a new ray is selected that touches the tip of the pixel. Tracing then continues along the new ray.

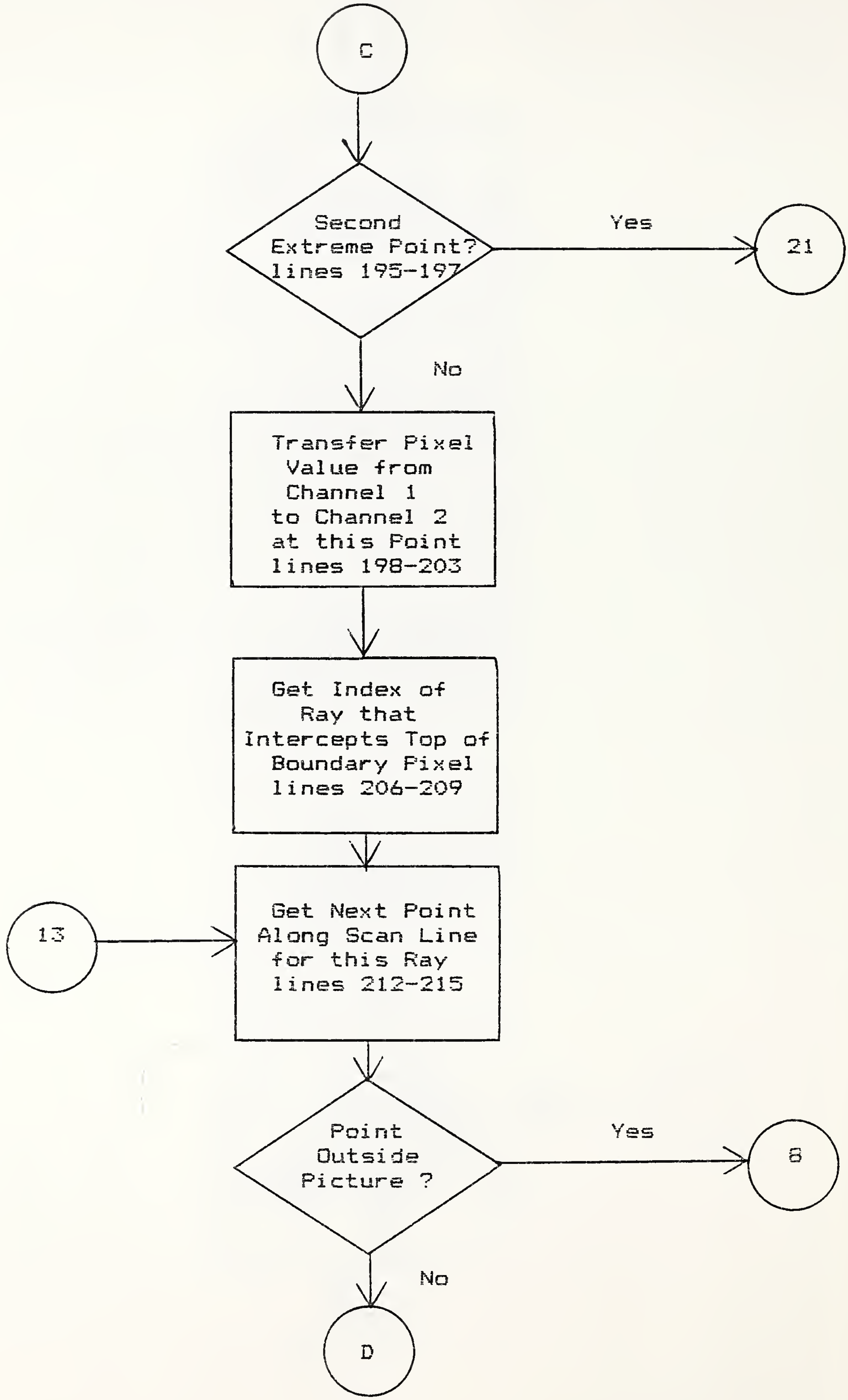
In the solid generation algorithm, rays are traced from the viewing plane to the surface. If a pixel is seen, then the height of the ray at contact is projected back to the viewing plane, modified by an intensity reduction factor if the pixel lies in shadow. If a pixel is not seen, then the ray is continued.

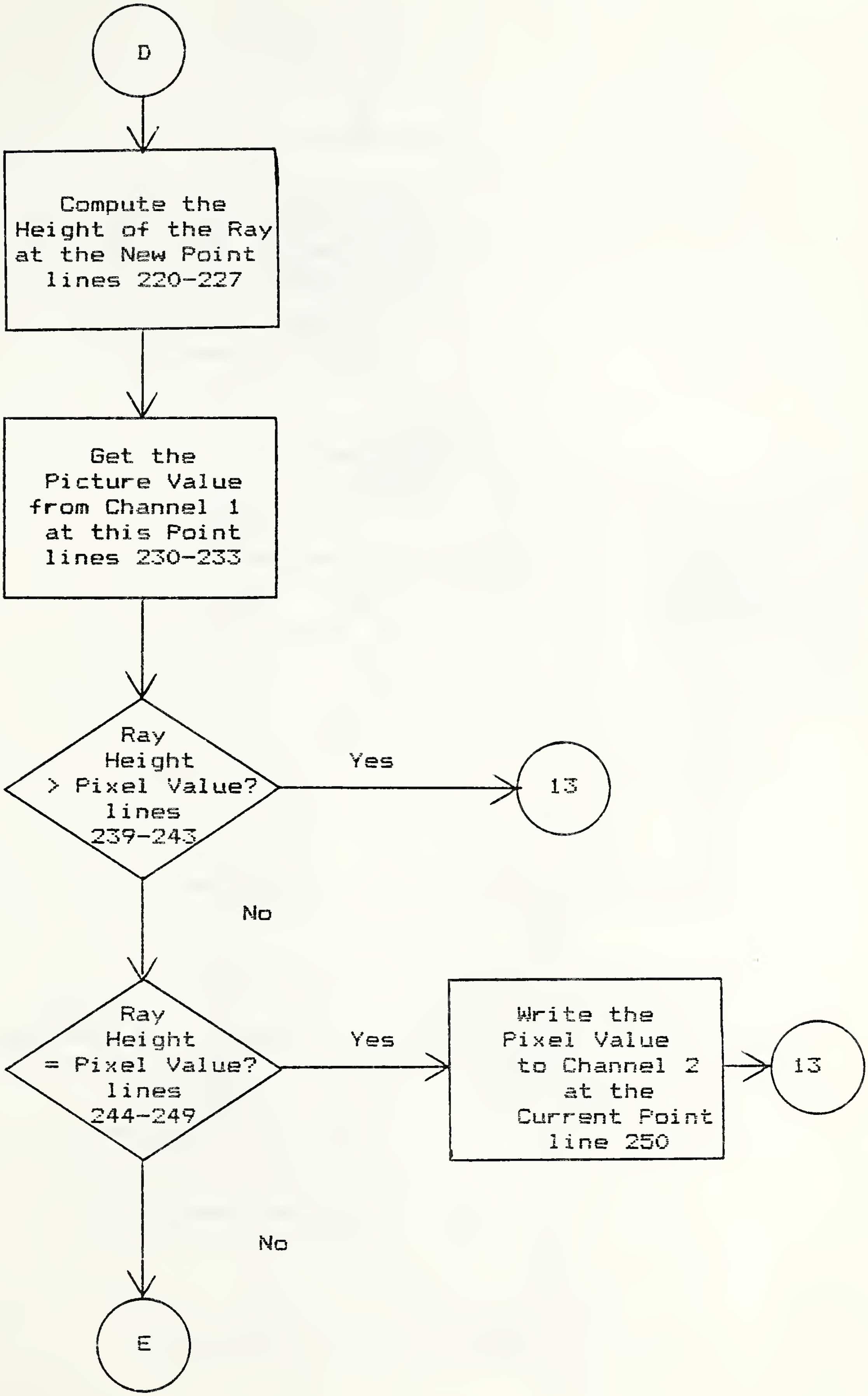
5.2.2 Flow Chart

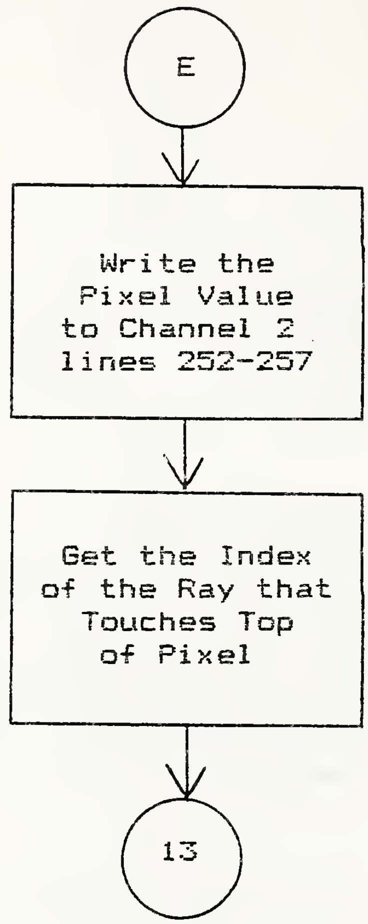


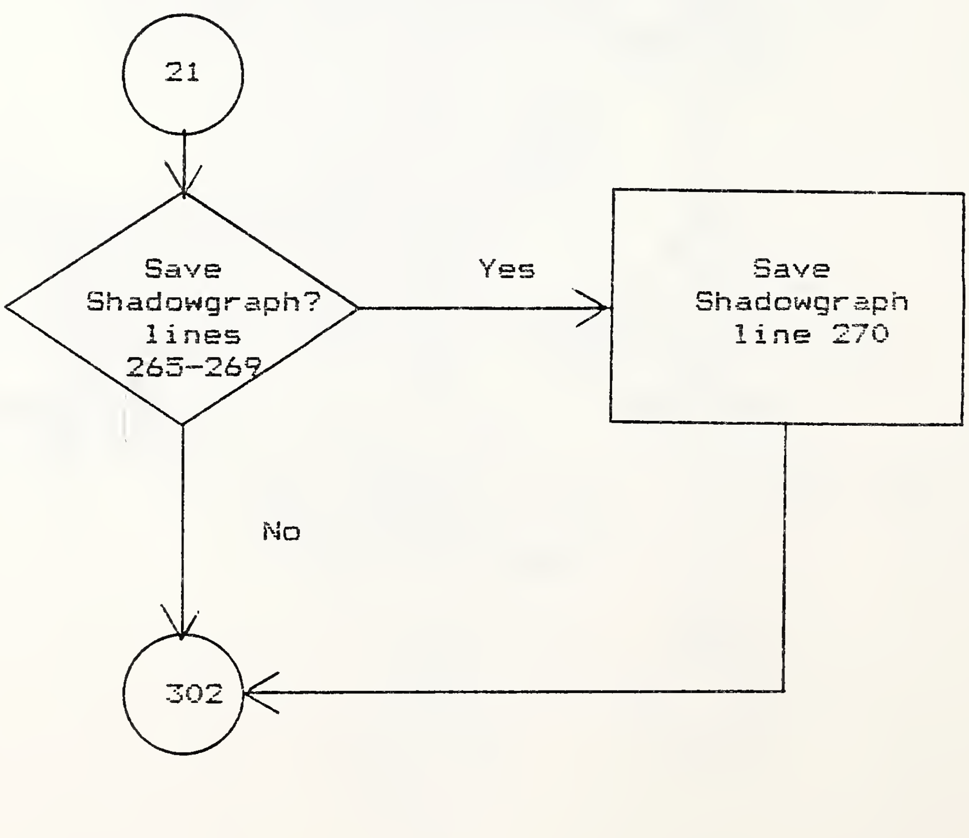

A
User Selects Azimuth and Elevation Angles lines 90-101
Compute Conversion Factor From Angles to Radius lines 102-103
Compute Cosines and Sines of the Azimuith and Elevation lines 110-115
Compute Orthonormal Ray Vectors
$\vec{W}$, $\vec{K}$, $\vec{K}x\vec{W}$
lines 143-150
Get the Case Number lines 153-156
B

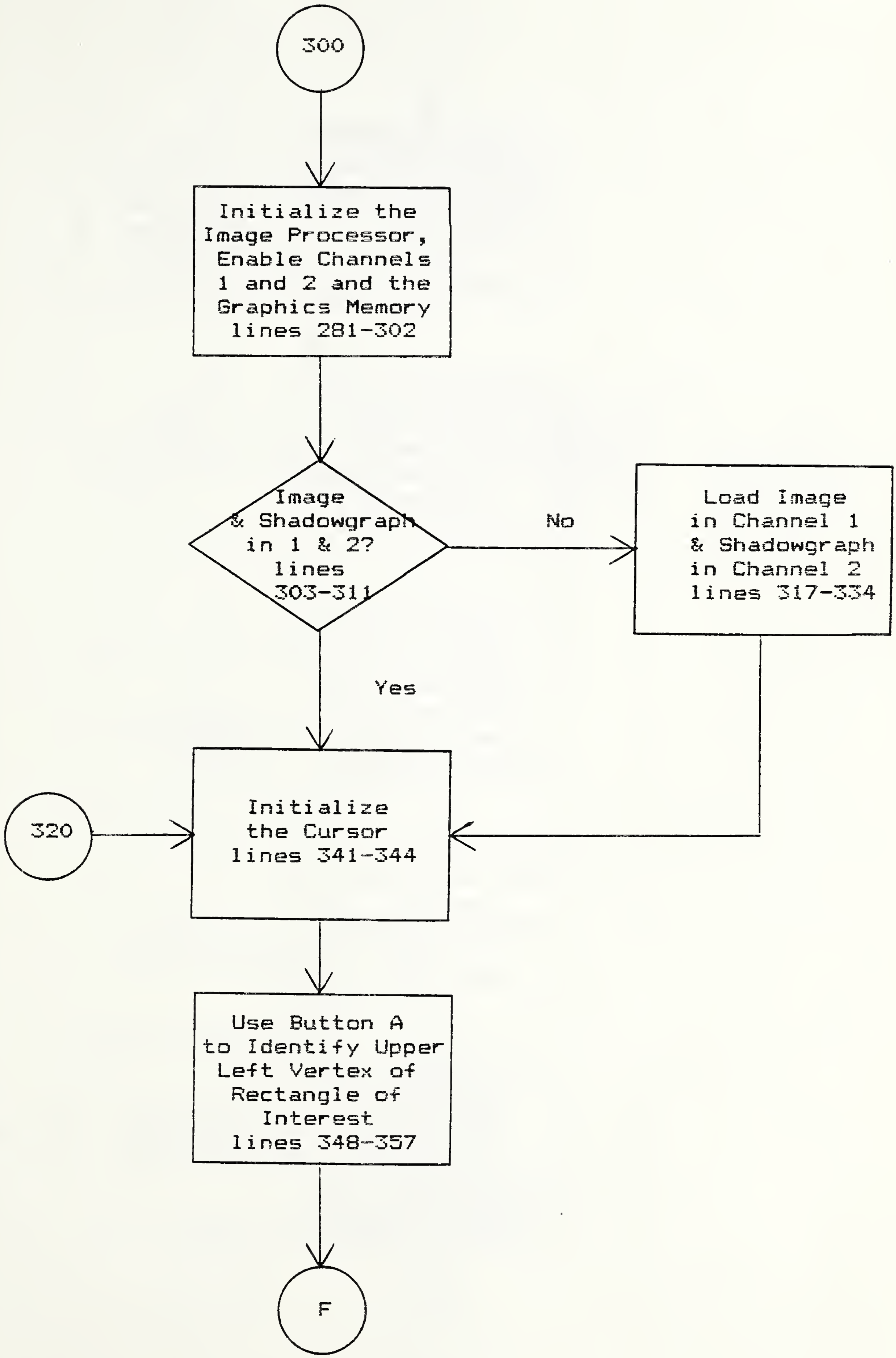

B
Set Up
Picture Vertices
lines 157-163
From Vertices
Select Extreme
Points for Ray
lines 164-167
Identify Extreme
Point 1 as
First Entry Point
lines 171-174
Transfer Image
Value from Chan 1
to Channel 2
at this Point
lines 175-180
8
Get Next
Boundary
Entry Point
lines 187-191
C

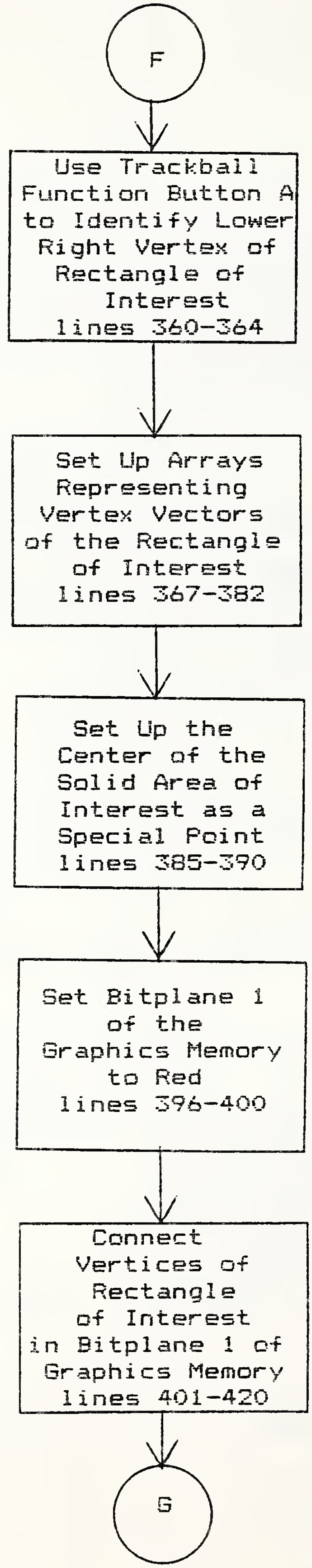

C
Second Extreme Point? lines 195-197
Yes
21
No
Transfer Pixel Value from Channel 1 to Channel 2 at this Point lines 198-203
Get Index of Ray that Intercepts Top of Boundary Pixel lines 206-209
13
Get Next Point Along Scan Line for this Ray lines 212-215
Point Outside Picture ?
Yes
8
No
D

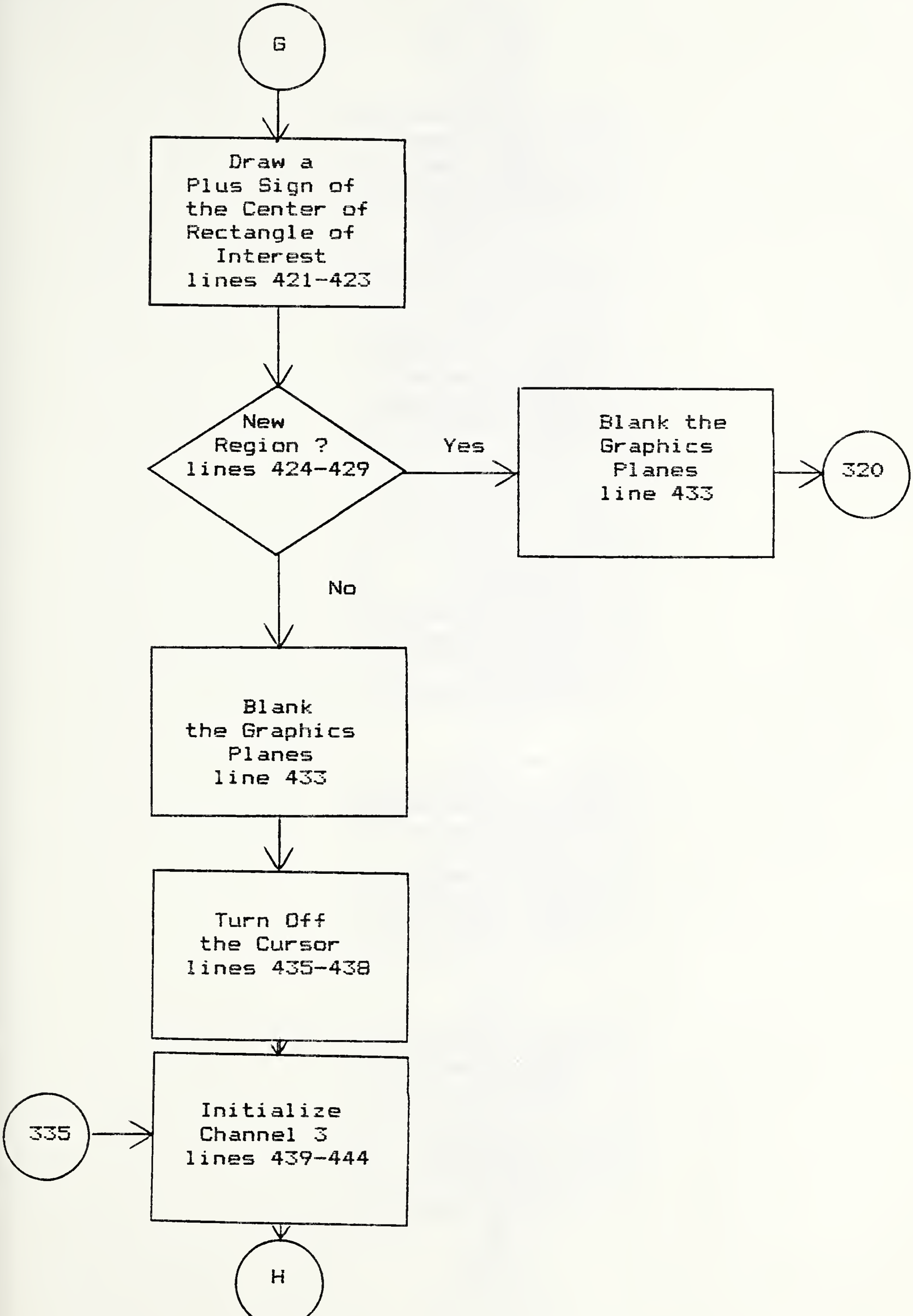

D
Compute the Height of the Ray at the New Point lines 220-227
Get the Picture Value from Channel 1 at this Point lines 230-233
Ray Height > Pixel Value? lines 239-243
Yes
13
No
Ray Height = Pixel Value? lines 244-249
Yes
Write the Pixel Value to Channel 2 at the Current Point line 250
13
No
E

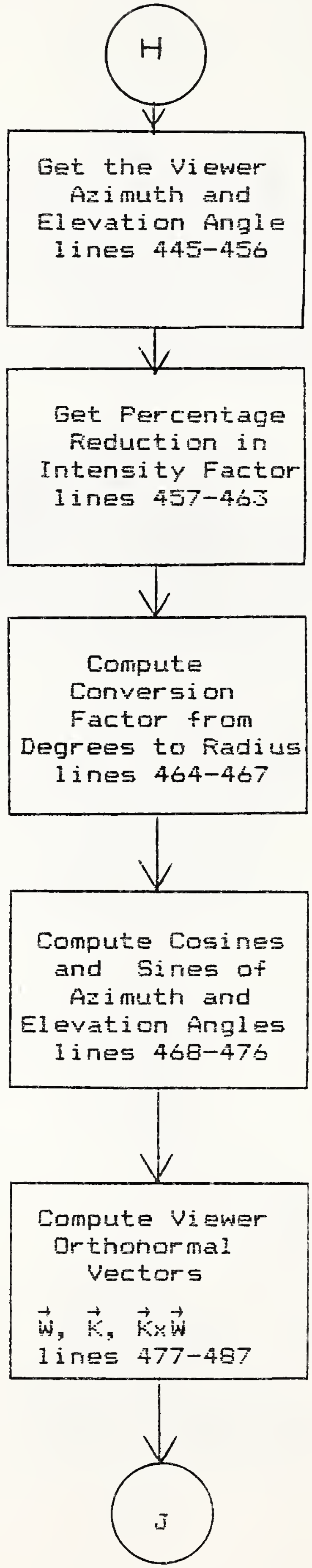

E
Write the Pixel Value to Channel 2 lines 252-257
Get the Index of the Ray that Touches Top of Pixel
13

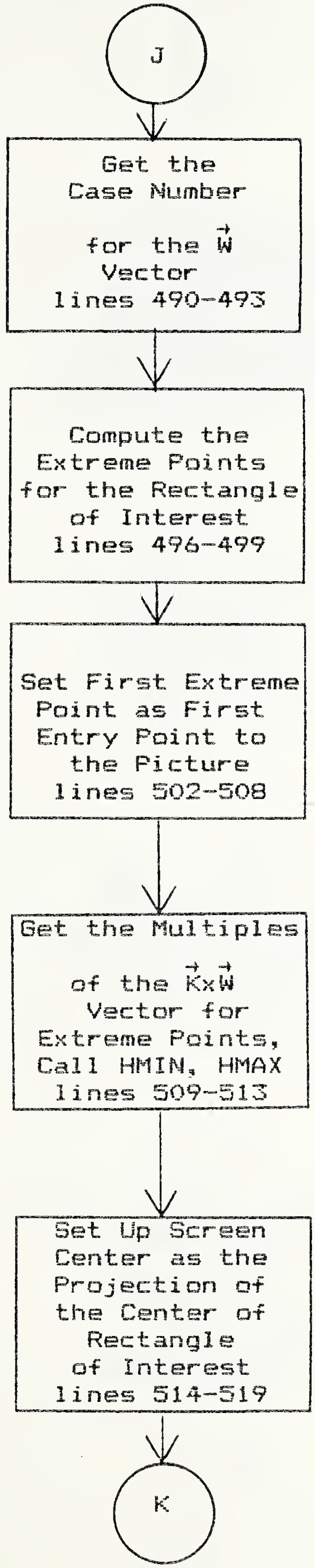

21
Save Shadowgraph? lines 265-269
Yes
Save Shadowgraph line 270
No
302

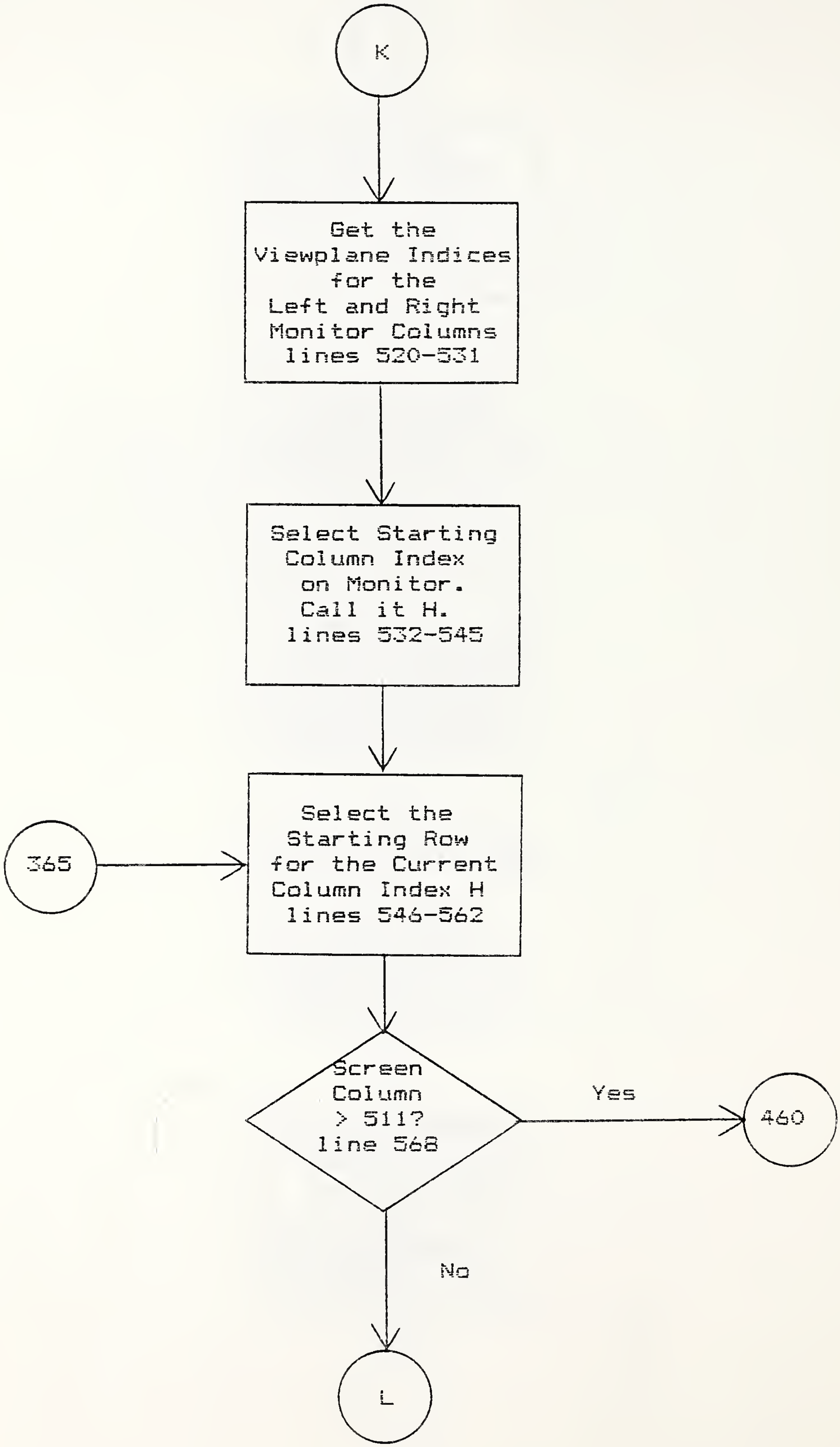

300
Initialize the Image Processor, Enable Channels 1 and 2 and the Graphics Memory lines 281-302
Image & Shadowgraph in 1 & 2? lines 303-311
No
Load Image in Channel 1 & Shadowgraph in Channel 2 lines 317-334
Yes
320
Initialize the Cursor lines 341-344
Use Button A to Identify Upper Left Vertex of Rectangle of Interest lines 348-357
F

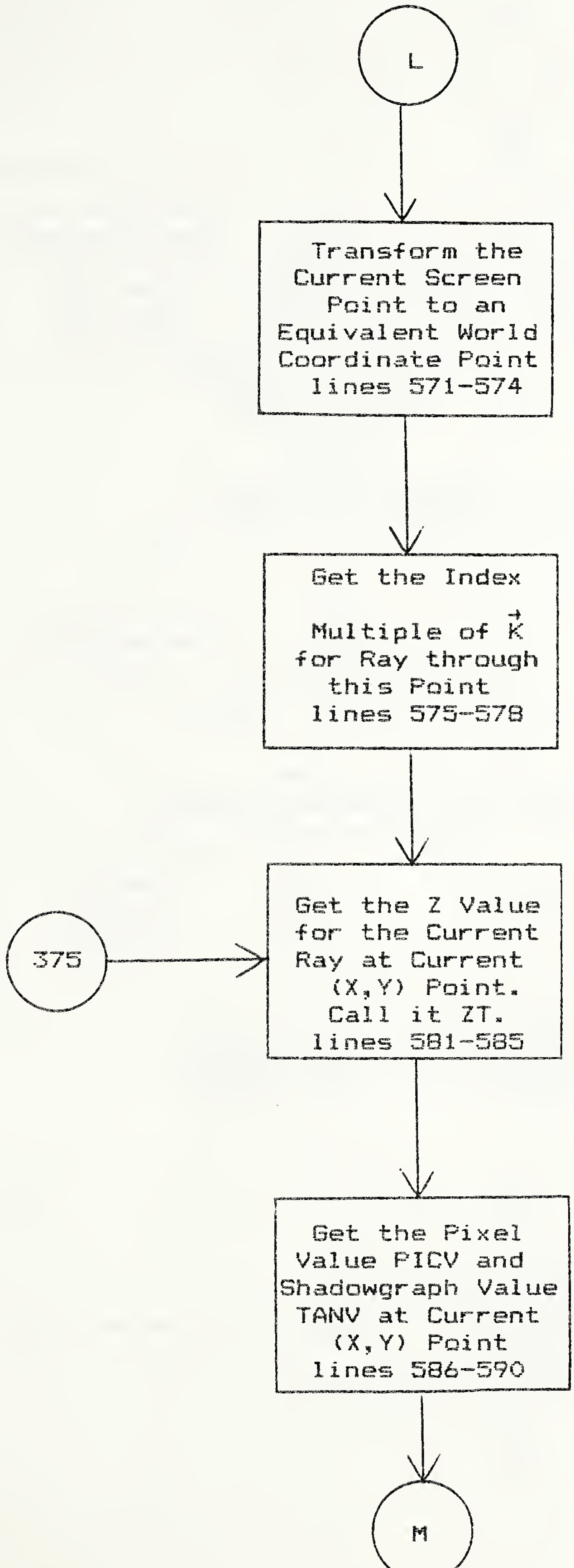

F
Use Trackball Function Button A to Identify Lower Right Vertex of Rectangle of Interest lines 360-364
Set Up Arrays Representing Vertex Vectors of the Rectangle of Interest lines 367-382
Set Up the Center of the Solid Area of Interest as a Special Point lines 385-390
Set Bitplane 1 of the Graphics Memory to Red lines 396-400
Connect Vertices of Rectangle of Interest in Bitplane 1 of Graphics Memory lines 401-420
G

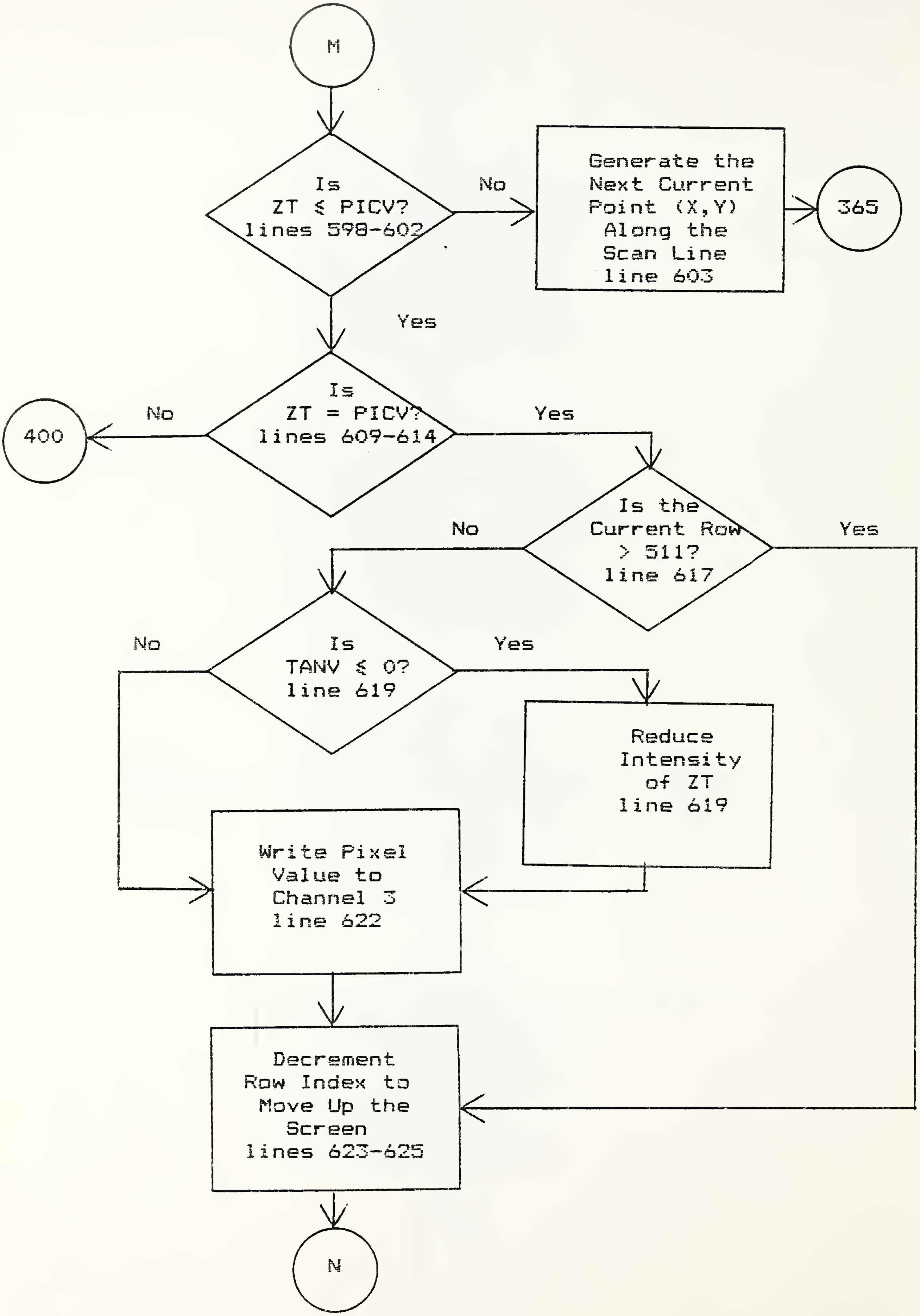

G
Draw a
Plus Sign of
the Center of
Rectangle of
Interest
lines 421-423
New
Region ?
lines 424-429
Yes
Blank the
Graphics
Planes
line 433
320
No
Blank
the Graphics
Planes
line 433
Turn Off
the Cursor
lines 435-438
335
Initialize
Channel 3
lines 439-444
H

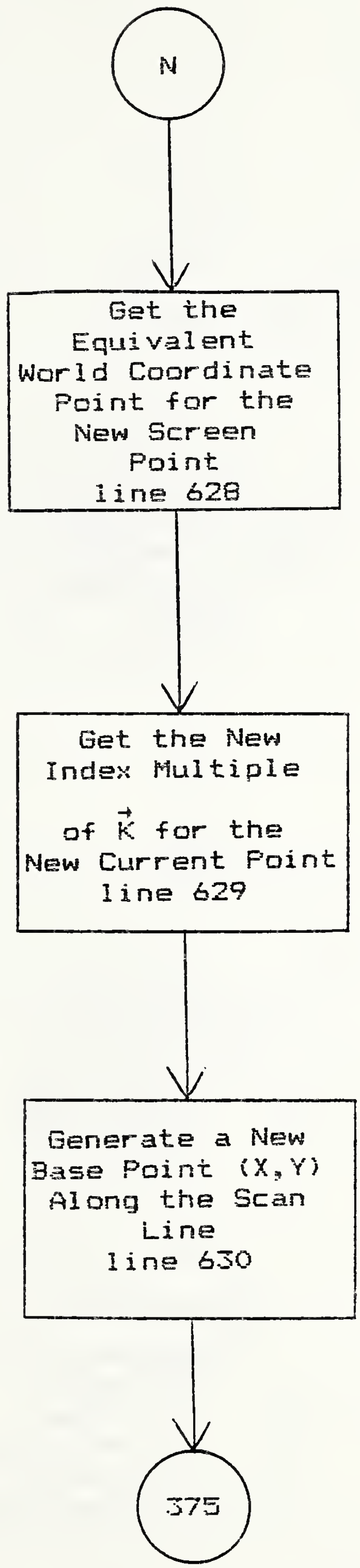

H
Get the Viewer Azimuth and Elevation Angle lines 445-456
Get Percentage Reduction in Intensity Factor lines 457-463
Compute Conversion Factor from Degrees to Radius lines 464-467
Compute Cosines and Sines of Azimuth and Elevation Angles lines 468-476
Compute Viewer Orthonormal Vectors
$\vec{W}$, $\vec{K}$, $\vec{K}\times\vec{W}$
lines 477-487
J

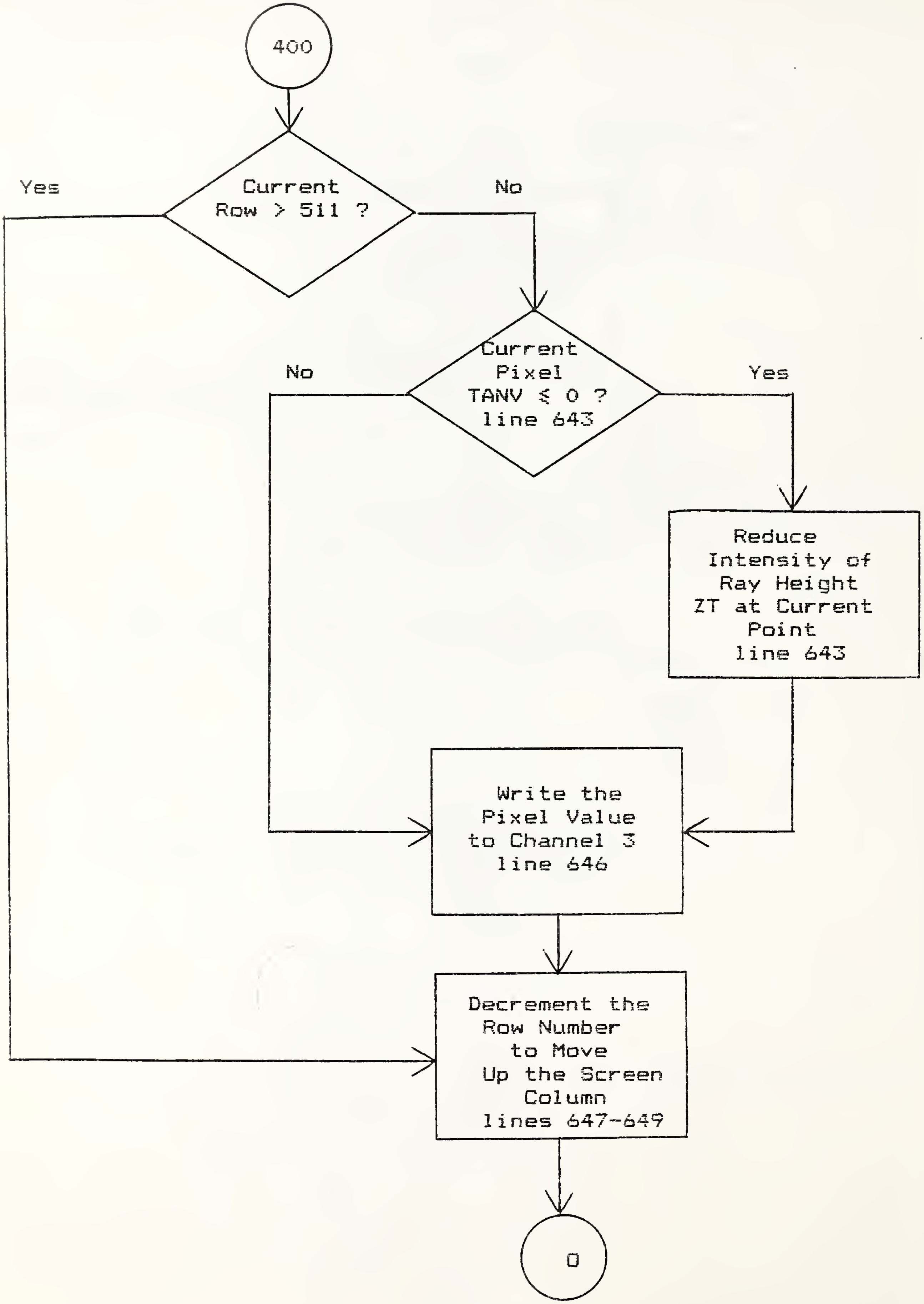

J
Get the Case Number for the $\vec{W}$ Vector lines 490-493
Compute the Extreme Points for the Rectangle of Interest lines 496-499
Set First Extreme Point as First Entry Point to the Picture lines 502-508
Get the Multiples of the $\vec{K}x\vec{W}$ Vector for Extreme Points, Call HMIN, HMAX lines 509-513
Set Up Screen Center as the Projection of the Center of Rectangle of Interest lines 514-519
K

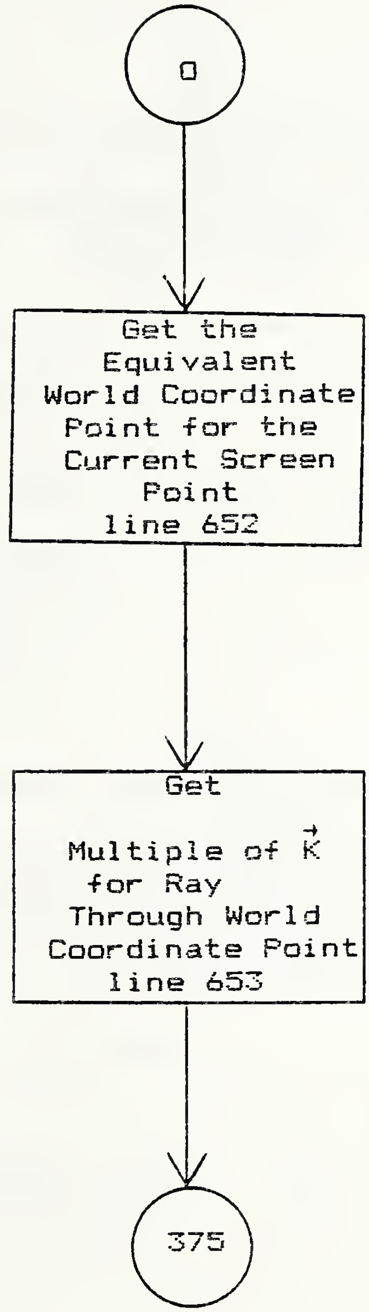

K
Get the Viewplane Indices for the Left and Right Monitor Columns lines 520-531
Select Starting Column Index on Monitor. Call it H. lines 532-545
365
Select the Starting Row for the Current Column Index H lines 546-562
Screen Column > 511? line 568
Yes
460
No
L


L
Transform the Current Screen Point to an Equivalent World Coordinate Point lines 571-574
Get the Index Multiple of $\vec{K}$ for Ray through this Point lines 575-578
375
Get the Z Value for the Current Ray at Current (X,Y) Point. Call it ZT. lines 581-585
Get the Pixel Value PICV and Shadowgraph Value TANV at Current (X,Y) Point lines 586-590
M

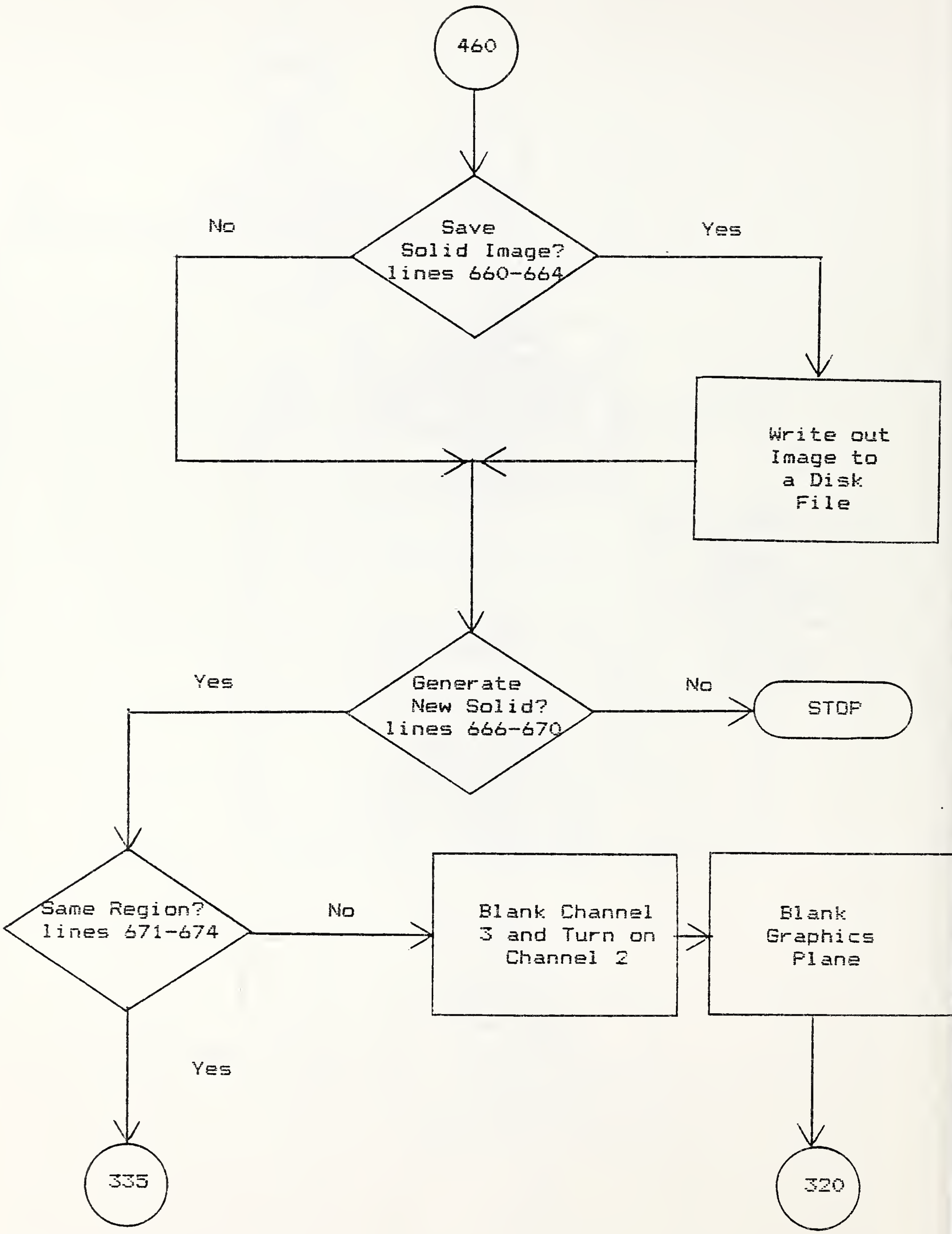

M
Is
ZT ≤ PICV?
lines 598-602
No
Generate the
Next Current
Point (X,Y)
Along the
Scan Line
line 603
365
Yes
Is
ZT = PICV?
lines 609-614
No
400
Yes
Is the
Current Row
> 511?
line 617
No
Yes
Is
TANV ≤ 0?
line 619
No
Yes
Reduce
Intensity
of ZT
line 619
Write Pixel
Value to
Channel 3
line 622
Decrement
Row Index to
Move Up the
Screen
lines 623-625
N


N
Get the Equivalent World Coordinate Point for the New Screen Point line 628
Get the New Index Multiple of $\vec{K}$ for the New Current Point line 629
Generate a New Base Point (X,Y) Along the Scan Line line 630
375


400
Current Row > 511 ?
Yes
No
Current Pixel TANV ≤ 0 ? line 643
No
Yes
Reduce Intensity of Ray Height ZT at Current Point line 643
Write the Pixel Value to Channel 3 line 646
Decrement the Row Number to Move Up the Screen Column lines 647-649
0



O
Get the Equivalent World Coordinate Point for the Current Screen Point
line 652
Get
Multiple of $\vec{K}$ for Ray Through World Coordinate Point
line 653
375


460
No
Save
Solid Image?
lines 660-664
Yes
Write out
Image to
a Disk
File
Yes
Generate
New Solid?
lines 666-670
No
STOP
Same Region?
lines 671-674
No
Blank Channel
3 and Turn on
Channel 2
Blank
Graphics
Plane
Yes
335
320

5.2.3 Listing

```
C
C*********************************************************************
C
C                            MAIN PROGRAM TO CREATE
C                                      A
C                                 PSEUDOSOLID
C
C*********************************************************************
      PROGRAM SOLID
      INTEGER*2 FCB(2048), BUFFER(2048), CHAN1(16), TAB1(16)
      INTEGER*2 CHAN2(16),TAB2(16)
      INTEGER*2 CHAN3(16),TAB3(16)
      INTEGER*2 PICV, TANV
      INTEGER*2 PSDO
      INTEGER*2 INBUF(2048)
      REAL W(3), K(3), KXW(3)
      REAL VRX(2), VRY(2)
      REAL EX(2), EY(2)
      INTEGER FILE(7),FILT(7)
      INTEGER FILP(7)
C
C BRANCH TO CREATE A SHADOW GRAPH OR PSEUDOSOLID ON USER REQUEST
C
      WRITE(5,5)
5     FORMAT(' IF YOU WISH TO SHADOW A PICTURE TYPE 0.'/
     1       ' IF YOU WISH TO CREATE A PSEUDO-SOLID TYPE 1.'/
     2       ' *** NOTE: TO PSEUDO-SOLID AN IMAGE A SHADOWGRAPH'/
     3       '              MUST HAVE PREVIOUSLY BEEN CREATED.')
      READ(5,*) IGO
      IF (IGO .NE. 0) GO TO 300
C
C---------------------------------------------------------------------
C
C INITIALIZATION SECTION
C
C---------------------------------------------------------------------
C
C
C REMARKS TO THE USER
C
      WRITE(5,1)
1     FORMAT('                   ***************************'/
     1       '                   *     PSEUDO-SOLID        *'/
     2       '                   *        PHASE 1          *'/
     3       '                   *     SHADOW GRAPH        *'/
     4       '                   ***************************'/
     5       ' '/
```

```
     6          '      THIS PORTION OF THE PSEUDO-SOLID GENERATION'/
     7          ' SIMULATES THE EFFECT OF A DISTANT SOURCE OF LIGHT'/
     8          ' SHINING ON THE SURFACE.  THOSE AREAS OF THE SURFACE'
     9          ' THAT WOULD BE SHADED ARE DARKENED.  NO THREE-'/
     1          ' DIMENSIONAL EFFECT IS CREATED IN THIS PART. '/
     2          '      THE MAIN PICTURE IS WRITTEN TO CHANNEL 1'/
     3          ' AND THE SHADOW GRAPH IS GENERATED ON CHANNEL 2. ')
C
C GET THE PICTURE FILE
C
        TAB1(1) = 1
15      CALL GETFIL(FCB,BUFFER,TAB1,CHAN1)
        WRITE(5,20)
20      FORMAT(' IF THE PICTURE HAS BEEN PROPERLY GENERATED,'/
     1          ' TYPE 1, OTHERWISE 0 TO GET ANOTHER PICTURE.')
        READ(5,*) IGO
        IF(IGO .EQ. 0) GO TO 15
C
C SET UP CHANNEL 2 OF I2S FOR SHADOW GRAPH
C
        TAB2(1) = 2
        CALL GETCHN(FCB,BUFFER,TAB2,CHAN2)
C
C SET UP FILE SPECIFICATIONS FOR:
C
C PICTURE FILE -
C
        FILE(3) = CHAN1(1)
C
C SHADOWGRAPH -
C
        FILT(3) = CHAN2(1)
C
C-----------------------------------------------------------------------
C
C END INITIALIZATION SECTION
C
C-----------------------------------------------------------------------
C
C BEGIN GEOMETRIC SPECIFICATION SECTION
C
C-----------------------------------------------------------------------
C
C GET AZIMUTH AND ELEVATION FOR THE LIGHT SOURCE
C
210     WRITE(5,121)
121     FORMAT(' ENTER AZIMUTH ANGLE AND ELEVATION ANGLE'/
     1          ' IN DEGREES FOR THE LIGHT SOURCE. '/
     2          ' AZIMUTH ANGEL LIMITS ARE 0 TO 360'/
     3          ' ELEVATION ANGLE LIMITS ARE 0 TO 90'/
     4          ' ENTER AS AZ , EL. ')
```

```
      READ(5,*) AZ,EL
      IF ( AZ .LT. 0. .OR. AZ .GT. 360 .OR. EL .LT. 0. .OR.
     1      EL .GE. 90. ) GO TO 210
C
C SET UP CONVERSION FACTOR FROM DEGREES TO RADIANS
C
      CONV = 3.14159/180.
C
C COMPUTE THE DIRECTION SINES AND COSINES FOR THE RAYS FROM
C THE LIGHT SOURCE TO THE SURFACE
C
      AZ = CONV*AZ
      EL = CONV*EL
      CE = COS(EL)
      CA = COS(AZ)
      SE = SIN(EL)
      SA = SIN(AZ)
C
C SET UP TWO UNIT VECTORS:
C             W  -  THIS UNIT VECTOR POINTS ALONG THE LIGHT
C                   RAYS TOWARDS THE ORIGIN
C             K  -  THIS UNIT VECTOR IS ORTHOGONAL TO W AND
C                   POINTS ACROSS A PLANE MADE OF LIGHT RAYS
C
C THESE COMMENTS ARE A NOTE ON THE UNDERLYING GEOMETRY.
C GIVEN AN AZIMUTH AND AN ELEVATION FOR THE LIGHT SOURCE ONE
C CAN THINK OF A PLANE FORMED BY ROTATING THE X-Z PLANE
C BY THE AZIMUTH ANGLE. NOW FILL UP THIS PLANE WITH LIGHT
C RAYS POINTING IN THE DIRECTION OF THE W-VECTOR. NOW
C CONSIDER A UNIT VECTOR IN THIS PLANE, CALLED K, THAT IS
C ORTHOGONAL TO W. THIS VECTOR IS THEN ORTHOGONAL TO ALL OF THE
C LIGHT RAYS POINTING IN THE DIRECTION W. EACH LIGHT RAY
C CAN BE INDEXED FROM A FIXED POINT ON THE PLANE BY ADDING
C SOME MULTIPLE OF THE K-VECTOR. FURTHERMORE FROM THAT SAME
C FIXED POINT ON THE PLANE ONE CAN ACCESS ANY POINT ON ANY
C LIGHT RAY IN THE PLANE BY ADDING A MULTIPLE OF K AND THE
C ADDING A MULTIPLE OF W. FINALLY, ALL LIGHT RAYS IN THE
C DIRECTION W FALL ON SOME PLANE PARALLEL TO THE ROTATED PLANE
C ABOVE. IF WE TAKE THE CROSS PRODUCT OF K AND W WE GET A
C VECTOR THAT CAN BE USED TO ACCESS ANY PLANE PARALLEL TO THE
C ROTATED PLANE. IN THIS PROGRAM THE MULTIPLES OF K ARE
C THE V-VARIABLES, THE MULTIPLES OF W ARE R-VARIABLES
C AND THE MULTIPLES OF THE CROSS PRODUCT ARE THE H'S.
C
      W(1)=-CE*CA
      W(2)=-CE*SA
      W(3)=-SE
      K(1)=CA*SE
      K(2)=SA*SE
      K(3)=-CE
      KXW(1) = -SA
      KXW(2) = CA
```

```
C      WRITE(6,900) W(1),W(2),W(3),K(1),K(2),K(3),KXW(1),KXW(2)
C900   FORMAT(' W=',3G15.7,' K=',3G15.7,' KXW=',2G15.7)
C
C GET THE W-VECTOR CASE INDEX
C
        CALL WCASE(W,IWCASE)
C
C SET UP THE ENTIRE PICTURE FOR SHADOWING
C
        VRX(1) = 0.
        VRY(1) = 0.
        VRX(2) = 511.
        VRY(2) = 511.
C
C GET THE EXTREME POINTS
C
        CALL EXTREM(IWCASE,VRX,VRY,EX,EY,IFLG)
C      WRITE(6,910)EX(1),EY(1),EX(2),EY(2)
C910   FORMAT(' EXTREME PTS 1=',2G15.7,' 2=',2G15.7)
C
        IXIN = EX(1)
        IYIN = EY(1)
        XIN = IXIN
        YIN = IYIN
C
C TRANSFER THE PIXEL VALUE TO THE SHADOWGRAPH SINCE IT CANNOT BE
C SHADOWED
C
        CALL RDPIC(FCB,FILE,PICV,IXIN,IYIN,1,IERR)
        CALL WRPIC(FCB,FILT,PICV,IXIN,IYIN,1,IERR)
C      WRITE(6,925) IXIN,IYIN,PICV
C925   FORMAT(' FIRST EXT. PT.=',3I10)
C
C GET THE BOUNDARY POINT OF THE PICTURE WHERE THE PROJECTED W
C RAY ENTERS
C
8       CALL XYIN(IWCASE,EX,EY,IXIN,IYIN,XIN,YIN,IFLG)
        IX = IXIN
        IY = IYIN
        X = XIN
        Y = YIN
C       WRITE(6,940)X,Y,IWCASE,EX(1),EY(1),EX(2),EY(2)
C940   FORMAT(' BOUNDARY PT =',2G15.7,' CASE=',I5,' EX1,EY1,EX2,EY2='
C    1         4G15.7)
        IF (IFLG .EQ. 1) STOP 'W(1)=W(2)=0 IN XYIN DURING SHADOW'
        IF (IFLG .EQ. 0) GO TO 10
        IF (IFLG .EQ. 2) GO TO 21
C
C TRANSFER THE PIXEL VALUE OF THE ENTRY POINT TO THE SHADOWGRAPH
C
10      CALL RDPIC(FCB,FILE,PICV,IX,IY,1,IERR)
        CALL WRPIC(FCB,FILT,PICV,IX,IY,1,IERR)
        Z = PICV
```

```
C       WRITE(6,950) IX,IY,PICV
C950    FORMAT(' BNDRY PT=',3I10)
C
C GET THE INDEX OF THE RAY THAT INTERCEPTS THE POINT (X,Y,Z)
C
        CALL GETV(X,Y,Z,K,V)
C       WRITE(6,960) V
C960    FORMAT(' BDRY RAY INDEX=',G15.7)
C
C GET THE NEXT POINT ALONG THE SCAN LINE
C
13      CALL GNXY(X,Y,IX,IY,W,IFLG)
C       WRITE(6,970) X,Y,IX,IY
C970    FORMAT(' NEXT POINT =',2G15.7,2I10)
        IF (IX .LT. 0 .OR. IX .GT. 511) GO TO 8
        IF (IY .LT. 0 .OR. IY .GT. 511) GO TO 8
C
C THE MODEL USED HERE ASSUMES THAT THE RAY SCIMS THE TOP OF A
C PIXEL THAT IT SEES, SINCE PIXELS ARE ASSUMED TO BE POINTS
C
C NOW COMPUTE THE HEIGHT ON THE CURRENT RAY INDEXED BY V AT THE
C POINT (X,Y)
C
        CALL GETZ(X,Y,V,K,ZT)
C       WRITE(6,980)ZT
C980    FORMAT(' ZT=',G15.7)
C
C GET THE PICTURE VALUE AT THE CURRENT POINT (IX,IY)
C
        CALL RDPIC(FCB,FILE,PICV,IX,IY,1,IERR)
C
C COMPARE THIS VALUE AGAINST THE RAY HEIGHT, ZT, AT THIS POINT
C TO DETERMINE WHETHER THE RAY SEES THE POINT
C
        P = PICV
C
C CASE 1: IF ZT > PICV THEN THE PIXEL IS NOT VISIBLE TO THIS RAY
C         CONTINUE TRACING THIS RAY.
C
        IF ( ZT .GT. P+1.E-5) GO TO 13
C
C CASE 2: IF ZT .EQ. PICV THEN THE POINT IS VISIBLE, WRITE THE
C         PIXEL OUT TO THE SHADOWGRAPH BUT CONTINUE TRACING
C         THE SAME RAY    .
C
        IF (ZT .LT. P-1.E-5) GO TO 19
        CALL WRPIC(FCB,FILT,PICV,IX,IY,1,IERR)
        GO TO 13
C
C CASE 3: IF ZT < PICV THEN THE PIXEL VALUE IS SEEN BY THE RAY
C         WRITE IT OUT AND GET THE FIRST RAY THAT SATISFIES ZT =
C         PICV
C
```

```
19      CALL WRPIC(FCB,FILT,PICV,IX,IY,1,IERR)
        Z = P + 1.E-5
        CALL GETV(X,Y,Z,K,V)
        GO TO 13
C
C WRITE SHADOWGRAPH OUT
C
21      CONTINUE
        WRITE(5,25)
25      FORMAT(' IF YOU WISH TO SAVE THIS SHADOWGRAPH TYPE 1,'/
     1         ' OTHERWISE 0')
        READ(5,*) IGO
        IF (IGO .NE. 1) GO TO 302
        CALL PUTFIL(FCB,BUFFER,TAB2,CHAN2)
        GO TO 302
C
C-----------------------------------------------------------------------------
C
C                                 ENTER THE PSEUDOSOLID
C                                       SECTION
C                                        BELOW
C
C-----------------------------------------------------------------------------
300     CONTINUE
C
C
C INITIALIZATION SECTION
C THIS SECTION IS ENTERED WHEN USING PSEUDOSOLID WITHOUT
C FIRST ENTERING THE SHADOWGRAPH SECTION
C
       CALL ZBUFF(FCB,16)
       CALL INFCB(FCB,2000,3)
       CALL MSTCL(FCB)
       TAB1(1) = 1
       TAB2(1) = 2
       CALL GETCHN(FCB,BUFFER,TAB1,CHAN1)
       CALL GETCHN(FCB,BUFFER,TAB2,CHAN2)
C
C ENABLE GRAPHICS
C
302    ICH = -32768
       CALL GRAFE(FCB,0,0,0,0,0,0,0,0)
C
C SET UP CHANNEL FOR PSEUDOSOLID
C
        TAB3(1) = 3
C
C DETERMINE WHETHER THE CHANNELS HAVE BEEN SETUP FOR PSEUDO
C ORIGINAL PICTURE MUST BE IN CHANNEL 1 AND THE SHADOWGRAPH
C MUST BE IN CHANNEL 2
C
```

```
      WRITE(5,305)
305   FORMAT(' IF THE ORIGINAL PICTURE IS IN CHANNEL 1 AND'/
     1       ' ITS SHADOWGRAPH IS IN CHANNEL 2 THEN TYPE 1'/
     2       ' OTHERWISE TYPE 0 TO TRANSFER THE PICTURES.')
      READ(5,*) IGO
      IF (IGO .NE. 0) GO TO 320
C
C LOAD PICTURE AND SHADOWGRAPH
C
      WRITE(5,307)
307   FORMAT(' ******** LOADING ORIGINAL IMAGE ********')
      TAB1(1) = 1
308   CALL GETFIL(FCB,BUFFER,TAB1,CHAN1)
      WRITE(5,309)
309   FORMAT(' IF THE PICTURE HAS BEEN PROPERLY GENERATED,'/
     1       ' TYPE 1, OTHERWISE 0 TO GET ANOTHER PICTURE.')
      READ(5,*) IGO
      IF (IGO .EQ. 0) GO TO 308
      WRITE(5,310)
310   FORMAT(' ******** LOADING THE SHADOWGRAPH *********')
      TAB2(1) = 2
311   CALL GETFIL(FCB,BUFFER,TAB2,CHAN2)
      WRITE(5,312)
312   FORMAT(' IF THE SHADOWGRAPH HAS BEEN PROPERLY GENERATED,'/
     1       ' TYPE 1, OTHERWISE 0 TO GET ANOTHER SHADOWGRAPH.')
      READ(5,*) IGO
      IF (IGO .EQ. 0) GO TO 311
C
C GET FILE SPECS
C
320   FILE(3) = CHAN1(1)
      FILT(3) = CHAN2(1)
      FILP(3) = CHAN3(1)
C
C SETUP THE CURSOR
C
      CALL GTCURS(FCB,BUFFER)
C
C GET THE REGION OF THE SHADOWGRAPH FOR PSEUDOSOLID ENHANCEMENT
C
      WRITE(5,321)
321   FORMAT(' ******** IDENTIFY THE REGION FOR PSEUDO-SOLID'/
     1       ' ******** ENHANCEMENT BY USING THE TRACKBALL.'/)
      WRITE(5,322)
322   FORMAT(' THE USER MUST IDENTIFY TWO DIAMETRICALLY OPPOSITE'/
     1       ' CORNERS OF A RECTANGLE USING THE TRACKBALL BUTTONS.'/
     2       ' MOVE THE CURSOR WITH THE TRACKBALL TO THE FIRST'/
     3       ' CORNER OF THE RECTANGLE OF INTEREST. PUSH BUTTON A.')
      CALL RBUTN(FCB,IB,IY1,IX1)
      CALL WAITB(FCB,10,IB,IY1,IX1)
C     WRITE(6,3221) IX1,IY1
C3221 FORMAT(' IX1,IY1 =',2I10)
      WRITE(5,323)
```

```
323     FORMAT(' NOW MOVE THE CURSOR TO THE DIAMETRICALLY OPPOSITE'/
     1        ' CORNER OF THE RECTANGLE OF INTEREST.  PUSH BUTTON A. ')
        CALL RBUTN(FCB, IB, IY2, IX2)
        CALL WAITB(FCB, 10, IB, IY2, IX2)
C       WRITE(6,3231) IX2, IY2
C3231   FORMAT(' IX2, IY2 =', 2I10)
C
C SETUP THE CORNER ARRAYS VRX, VRY
C
        IF (IX1 .LE. IX2) GO TO 325
        VRX(1) = IX2
        VRX(2) = IX1
        GO TO 326
325     VRX(1) = IX1
        VRX(2) = IX2
326     IF (IY1 .LE. IY2) GO TO 327
        VRY(1) = IY2
        VRY(2) = IY1
        GO TO 328
327     VRY(1) = IY1
        VRY(2) = IY2
328     CONTINUE
C       WRITE(6,3271) VRX(1),VRX(2),VRY(1),VRY(2)
C3271   FORMAT(' VRX, VRY =', 4G15.7)
C
C COMPUTE THE CENTER OF THE RECTANGLE OF INTEREST
C
        XO = (VRX(1) + VRX(2))/2.0
        YO = (VRY(1) + VRY(2))/2.0
        ZO = 128.
C       WRITE(6,3272) VRX(1),VRY(1),VRX(2),VRY(2),XO,YO
C3272   FORMAT(' VRX1,VRY1,VRX2,VRY2=', 4G15.7, ' XO,YO=', 2G15.7)
C
C OUTLINE THE AREA AND PUT A PLUS AT THE CENTER
C
        DO 329 I = 1,2048
        INBUF(I) = -1
329     CONTINUE
       CALL STCOL(FCB, BUFFER, 0, 1.0, 0.0, 0.0, 1)
       CALL XCOLR(FCB, BUFFER, 0, 1)
        IX1 = VRX(1)
        IX2 = VRX(2)
        IY1 = VRY(1)
        IY2 = VRY(1)
        CALL DVECT(FCB, IY1, IX1, IY2, IX2, ICH, 1, INBUF)
        IX1 = IX2
        IX2 = VRX(2)
        IY1 = IY2
        IY2 = VRY(2)
        CALL DVECT(FCB, IY1, IX1, IY2, IX2, ICH, 1, INBUF)
        IX1 = IX2
        IX2 = VRX(1)
        IY1 = IY2
        IY2 = VRY(2)
```

```
      CALL DVECT(FCB, IY1, IX1, IY2, IX2, ICH, 1, INBUF)
      IX1 = IX2
      IX2 = VRX(1)
      IY1 = IY2
      IY2 = VRY(1)
      CALL DVECT(FCB, IY1, IX1, IY2, IX2, ICH, 1, INBUF)
      IXO = XO
      IYO = YO
      CALL DPLUS(FCB, INBUF, ICH, 1, IYO, IXO, 32)
      WRITE(5, 330)
330   FORMAT(' IF YOU WISH TO CHANGE YOUR MIND ON THE '/
     1       ' RECTANGLE OF INTEREST PUSH BUTTON B, OTHERWISE'/
     2       ' PUSH BUTTON A')
      CALL RBUTN(FCB, BUFFER, IY, IX)
      CALL WAITB(FCB, 10, IB, IY, IX)
C
C BLANK THE GRAPHICS PLANES
C
      CALL BCHAN(FCB, BUFFER, -32768, -1)
      IF (IB .GE. 2) GO TO 320
C
C TURN OFF THE CURSOR
C
      CALL CRCTL(FCB, 0, 0, 0, 0, 0, 0, 0, 0, 0)
C
C INITIALIZE CHANNEL 3 FOR PSEUDOSOLID GENERATION
C
335   TAB3(1) = 3
      CALL GETCHN(FCB, BUFFER, TAB3, CHAN3)
      FILP(3) = CHAN3(1)
C
C GET THE AZIMUTH AND ELEVATION OF THE VIEWING ANGLE
C
340   WRITE(5, 341)
341   FORMAT(' ENTER AZIMUTH ANGLE AND ELEVATION ANGLE'/
     1       ' IN DEGREES FOR THE VIEWER.  AZIMUTHE ANGLE'/
     2       ' LIMITS ARE 0 TO 360.  ELEVATION ANGLE LIMITS'/
     3       ' ARE 0 TO 90.'/
     4       ' ENTER AS AZ, EL.')
      READ(5,*) AZ, EL
      IF (AZ .LT. 0. .OR. AZ .GT. 360. .OR. EL .LT. 0. .OR.
     1    EL .GE. 90.) GO TO 340
C
C GET THE PERCENTAGE REDUCTION FOR SHADOWING
C
      WRITE(5, 342)
342   FORMAT(' ENTER THE PERCENT REDUCTION IN INTENSITY DESIRED'/
     1       ' FOR SHADOWING.  ENTER FROM 0.  TO 100.')
      READ(5,*) PRCNT
C
C CONVERSION FACTOR: DEGREES TO RADIANS
C
      CONV = 3.14159/180.
C
```

```
C GET SINES AND COSINES
C
      AZ = CONV*AZ
      EL = CONV*EL
      CE = COS(EL)
      CA = COS(AZ)
      SE = SIN(EL)
      SA = SIN(AZ)
C
C SET UP THE VIEWER FRAME OF REFERENCE
C
      W(1) = -CE*CA
      W(2) = -CE*SA
      W(3) = -SE
      K(1) = CA*SE
      K(2) = SA*SE
      K(3) = -CE
      KXW(1) = -SA
      KXW(2) = CA
C     WRITE(6,345)W(1),W(2),W(3),K(1),K(2),K(3),KXW(1),KXW(2)
C345  FORMAT(' W1,W2,W3,K1,K2,K3,KXW1,KXW2=',8G15.7)
C
C GET THE W-VECTOR CASE INDEX
C
      CALL WCASE(W,IWCASE)
C     WRITE(6,346) IWCASE
C346  FORMAT(' IWCASE =',I4)
C
C GET THE EXTREME POINTS FOR THE PSEUDOSOLID RECTANGLE
C
      CALL EXTREM(IWCASE,VRX,VRY,EX,EY,IFLG)
C     WRITE(6,347) EX(1),EY(1),EX(2),EY(2)
C347  FORMAT(' EX1,EY1,EX2,EY2=',4G15.7)
C
C SET UP THE FIRST ENTRY POINT TO THE PICTURE
C
      IXIN = EX(1)
      IYIN = EY(1)
      XIN = IXIN
      YIN = IYIN
C
C GET THE MULTIPLES OF THE KXW VECTOR FOR THE EXTREME POINTS
C
      CALL GETH(EX(1),EY(1),0.,KXW,HMIN)
      CALL GETH(EX(2),EY(2),0.,KXW,HMAX)
C
C SET UP THE SCREEN CENTER FOR THE PROJECTION OF THE RECTANGLE
C OF INTEREST CENTER. NOTE THESE ARE IN SCREEN COORDINATES. X AND Y ARE REVERSE
C
      SX0 = 256.
      SY0 = 256.
C
C GET THE LIGHT PLANE INDEX FOR THE LEFT HAND COLUMN OF THE
C MONITOR
C
```

```
      SY = 511.
      SX = 0.
      CALL GETXYZ(XO, YO, ZO, SYO, SXO, SY, SX, K, KXW, X, Y, Z)
      CALL GETH(X, Y, Z, KXW, H1)
      SY = 511.
      SX = 511.
      CALL GETXYZ(XO, YO, ZO, SYO, SXO, SY, SX, K, KXW, X, Y, Z)
      CALL GETH(X, Y, Z, KXW, H2)
C
C GET THE STARTING VIEWPORT COLUMN
C
      IF (H1 .LE. HMIN) GO TO 350
      H = H1
      ISXO = H1
      IF (H1 .GT. HMAX) ISXO = HMAX
      GO TO 360
350   H = HMIN
      ISXO = HMIN - H1
      IF (ISXO .GT. 511) ISXO = 511
360   CONTINUE
      ISY = 511
      ISX = ISXO - 1
C
C GET THE STARTING VIEWPORT ROW
C FOR THE CURRENT H
C
365   CALL XYIN(IWCASE, EX, EY, IXIN, IYIN, XIN, YIN, IFLG)
C     WRITE(6,366) IXIN, IYIN, XIN, YIN
C366  FORMAT(' AT COL. ENTRY IXIN, IYIN, XIN, YIN=', 2I4, 2G15.7)
      IF (IFLG .EQ. 0 ) GO TO 370
      IF (IFLG .EQ. 1) STOP 'W(1)=W(2)=0 IN PSEUDO'
      IF (IFLG .EQ. 2) GO TO 460
370   IX = IXIN
      IY = IYIN
      X = XIN
      Y = YIN
      ISX = ISX + 1
      CALL GETROW(XO, YO, ZO, SYO, K, X, Y, 0. , SY)
      ISY = SY
C
C BEGIN MOVING UP THE COLUMN ON THE MONITOR
C
      SX = ISX
      SY = ISY
      IF (ISX .GT. 511 ) GO TO 460
C     WRITE(6,371) ISY, ISX
C371  FORMAT(' AT COL. ENTRY ISY, ISX=', 2I4)
C
C GET THE X, Y, Z VALUE ASSOCIATED WITH THE SCREEN POINT ISY, ISX
C
      CALL GETXYZ(XO, YO, ZO, SYO, SXO, SY, SX, K, KXW, XT, YT, ZT)
C
C GET THE INDEX OF THE RAY THROUGH THIS POINT
C
```

```
      CALL GETV(XT,YT,ZT,K,V)
C     WRITE (6,372)XT,YT,ZT,V
C372  FORMAT(' WRLD. COORD. FOR ENTRY XT,YT,ZT=',3G15.7,'V=',G15.7)
C
C GET THE Z VALUE ON THE RAY INDEXED BY V AT THE CURRENT POINT
C X,Y
C
375   CALL GETZ(X,Y,V,K,ZT)
C
C GET THE PIXEL AND SHADOWGRAPH VALUE AT THE CURRENT POINT
C
      CALL RDPIC(FCB,FILE,PICV,IX,IY,1,IERR)
      CALL RDPIC(FCB,FILT,TANV,IX,IY,1,IERR)
C     WRITE(6,376)X,Y,IX,IY,ZT,PICV,TANV
C376  FORMAT(' AT CURR.PT.X,Y,IX,IY,ZT,PICV,TANV=',2G15.7,2I4,
C    1       G15.7,2I4)
C
C COMPARE THE PICTURE VALUES AGAINST THE RAY HEIGHT
C
      P = PICV
C
C CASE 1:  IF ZT > PICV THEN THE CURRENT RAY DOES NOT SEE
C          THE POINT.  CONTINUE TRACING THE RAY
C
      IF (ZT .LE. P + 1.E-5) GO TO 380
      CALL GNXY(X,Y,IX,IY,W,IFLG)
C     WRITE(6,377)X,Y,IX,IY
C377  FORMAT(' ZT>PICV :  X,Y,IX,IY=',2G15.7,2I4)
      IF (VRX(1) .LE. X .AND. X .LE. VRX(2) .AND. VRY(1) .LE.
     1    Y .AND. Y .LE. VRY(2)) GO TO 375
      GO TO 365
C
C CASE 2:  IF ZT = PICV THE POINT IS SEEN BY THE RAY.  DO NOT
C          CONTINUE THE RAY.  GET A NEW RAY AND THEN CONTINUE
C          TRACING
C
380       IF (ZT .LT. P - 1.E-5) GO TO 400
C         WRITE(6,3801)
C3801     FORMAT(' ZT=PICV')
          IF (ISY .GT. 511) GO TO 385
      IZT = ZT
      IF (TANV .LE. 0) IZT = (100.-PRCNT)*ZT/100.
      PSDO = IZT
      IF (IZT .LT. 0) PSDO = 0
      CALL WRPIC(FCB,FILP,PSDO,ISY,ISX,1,IERR)
385   ISY = ISY - 1
      SY = ISY
      SX = ISX
C     WRITE(6,387)ISY,ISX
C387  FORMAT(' NEW SCREEN PT.=',2I4)
      CALL GETXYZ(XO,YO,ZO,SYO,SXO,SY,SX,K,KXW,XT,YT,ZT)
      CALL GETV(XT,YT,ZT,K,V)
      CALL GNXY(X,Y,IX,IY,W,IFLG)
```

```
C       WRITE(6,386)XT,YT,ZT,V,X,Y,IX,IY
C386    FORMAT(' WRLD.COORD.  FOR CURR.PT.X,Y,Z=',3G15.7,'V=',G15.7,
C     1           ' NEW PT.=X,Y,IX,IY=',2G15.7,2I4)
        GO TO 375
C
C CASE 3:  IF ZT < PICV THE PIXEL IS SEEN BY THE RAY BUT DO NOT
C             CONTINUE THE RAY.
C
  400   IF (ISY .GT.  511) GO TO 450
C       WRITE(6,401)
C401    FORMAT(' ZT<PICV')
        IZT = ZT
        IF (TANV .LE.  0) IZT = (100.-PRCNT)*ZT/100.
        PSDO = IZT
        IF (IZT .LT.  0) PSDO = 0
        CALL WRPIC(FCB,FILP,PSDO,ISY,ISX,1,IERR)
  450   ISY = ISY - 1
        SY = ISY
        SX = ISX
C       WRITE(6,451)ISY,ISX
C451    FORMAT(' NEW SCREEN POINT=',2I4)
        CALL GETXYZ(XO,YO,ZO,SYO,SXO,SY,SX,K,KXW,XT,YT,ZT)
        CALL GETV(XT,YT,ZT,K,V)
C       WRITE(6,452)XT,YT,ZT,V
C452    FORMAT(' WRLD.COORD.  XT,YT,ZT=',3G15.7,' V=',G15.7)
        GO TO 375
C
C WRITE OUT PSEUDOSOLID PICTURE
C
  460   WRITE(5,470)
  470   FORMAT(' IF YOU WISH TO SAVE THE PSEUDOSOLID IMAGE TYPE'/
      1           ' 1, OTHERWISE 0')
        READ(5,*) IGO
        IF (IGO .NE.  1) GO TO 475
        CALL PUTFIL(FCB,BUFFER,TAB3,CHAN3)
  475   WRITE(5,480)
  480   FORMAT(' IF YOU WISH TO GENERATE ANOTHER SOLID TYPE 1,'/
      1          ' OTHERWISE 0')
        READ(5,*) IGO
        IF ( IGO .EQ.  0) STOP
        WRITE(5,485)
  485   FORMAT(' IF YOU WANT THE SAME REGION-OF-INTEREST TYPE 1,'/
      1          ' OTHERWISE 0')
        READ(5,*) IM
        IF (IM .EQ.  0) GO TO 320
        GO TO 335
        STOP
        END
C
```

5.3 Subroutine GTCURS

5.3.1 Summary

This subroutine intializes the programmable cursor at the center point of the screen. The calling sequence is:

CALL GTCURS (FCB, BUFFER).

The parameters passed are:

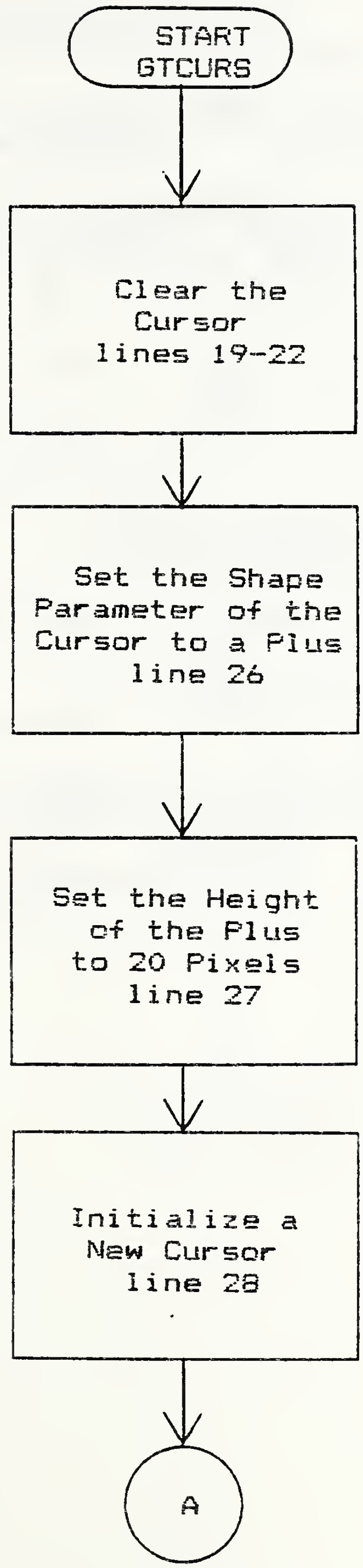
FCB	-	System Function Control Block for the image processor. INTEGER*2 Array
BUFFER	-	System buffer. INTEGER*2 Array

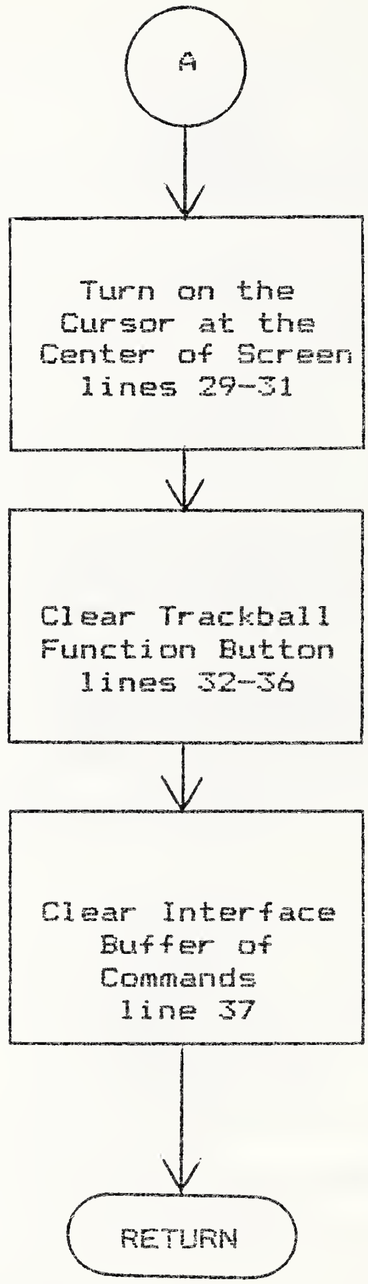
GTCURS calls the following subroutines:

DCURS
DEXEC
ONCUR
RBUTN .

The calling sequences for the system supplied subroutines or functions required by each of the major user subroutines are given in Appendix B. These are unique to the host and image processor systems used and are not transportable. In order to implement this code on another system, these system calls must be emulated or the entire code converted to any new system calls.

5.3.2 Flow Chart




A
Turn on the
Cursor at the
Center of Screen
lines 29-31
Clear Trackball
Function Button
lines 32-36
Clear Interface
Buffer of
Commands
line 37
RETURN

5.3.3 Listing

```
      SUBROUTINE GTCURS(FCB,BUFR)
C**************************************************
C
C THIS SUBROUTINE INITIALIZES THE CURSOR AT THE
C CENTER POINT OF THE SCREEN
C
C**************************************************
      INTEGER*2 FCB(1),BUFR(1)
      INTEGER SHAPE
      REAL SIZE
C
C CLEAR CURSOR DEFINITION
C
      CALL DCURS(FCB,BUFR,5,0.0)
C
C CREATE A PLUS SHAPED CURSOR
C
      SHAPE = 3
      SIZE = 20.
      CALL DCURS(FCB,BUFR,SHAPE,SIZE)
      IX = 255
      IY = 255
      CALL ONCUR(FCB,BUFR,1.,0.,0.,IX,IY,0)
C
C CLEAR BUTTONS WITH A READ OF BUTTON WORD WHICH IS
C 0 FOR NO BUTTONS PUSHED
C
      CALL RBUTN(FCB,BUTTON,IX,IY)
      CALL DEXEC(FCB)
      RETURN
      END
```

5.4 Subroutine SETCOL

5.4.1 Summary

This subroutine sets the color specifications for the image processor graphics memory bitplanes. Its calling sequence is:

CALL SETCOL (FCB, BUFFER).

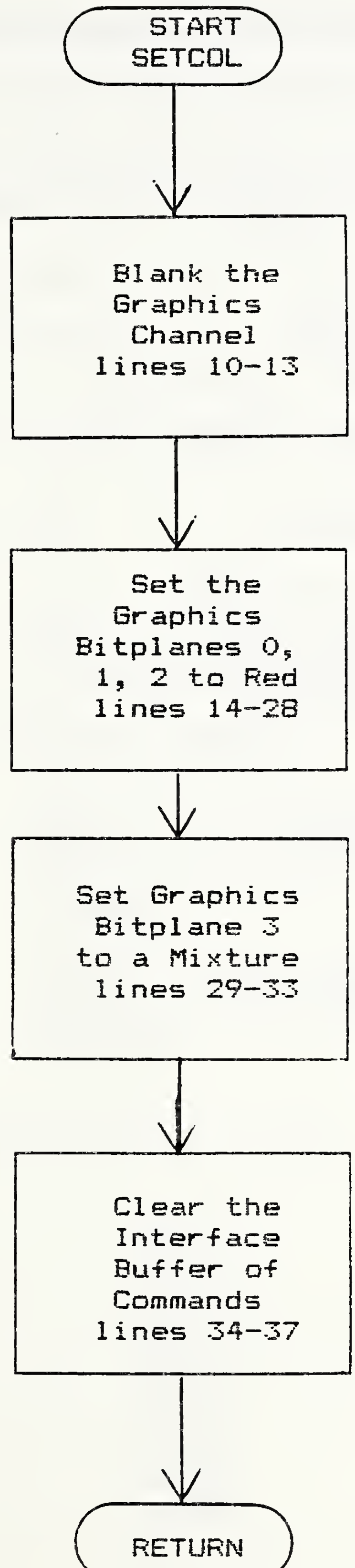
The parameters passed are:

FCB	–	System Function Control Block for the image processor. INTEGER*2 Array
BUFFER	–	System buffer array. INTEGER*2 Array

SETCOL calls the following subroutines:

BCHAN
DEXEC
STCOL
XCOLR .

5.4.2 Flow Chart



5.4.3 Listing

```
      SUBROUTINE SETCOL(FCB,BUFR)
C**************************************************
C
C THIS SUBROUTINE INITIALIZES COLOR IN THE
C GRAPHICS BITPLANES
C
C**************************************************
      INTEGER*2 FCB(1),BUFR(1)
      INTEGER BUTTON
C
C BLANK THE GRAPHICS CHANNEL
C
      CALL BCHAN(FCB,BUFR,-32768,127)
C
C SET GRAPHICS BITPLANE 0 TO RED
C
      CALL STCOL(FCB,BUFR,0,1.,0.,0.,1)
      CALL XCOLR(FCB,BUFR,0,1)
C
C SET GRAPHICS BITPLANE 1 TO RED
C
      CALL STCOL(FCB,BUFR,1,1.,0.,0.,1)
      CALL XCOLR(FCB,BUFR,1,1)
C
C SET GRAPHICS BITPLANE 2 TO RED
C
      CALL STCOL(FCB,BUFR,2,1.,0.,0.,1)
      CALL XCOLR(FCB,BUFR,2,1)
C
C SET GRAPHICS BITPLANE 3 TO A MIXTURE
C
      CALL STCOL(FCB,BUFR,3,.7,.7,.7,1)
      CALL XCOLR(FCB,BUFR,3,0)
C
C DO IT!
C
      CALL DEXEC(FCB)
      RETURN
      END
```

5.5 Subroutine GETFIL

5.5.1 Summary

This subroutine interactively inquires of the user the name of a desired picture file, opens the file, initializes a user selected refresh memory and writes the data file from the host computer to the selected refresh memory in the image processor. The program assumes that files are formatted as sequential files with 512 records of 512 bytes each. The calling sequence is:

CALL GETFIL (FCB, BUFFER, TABLE, CHANLS).

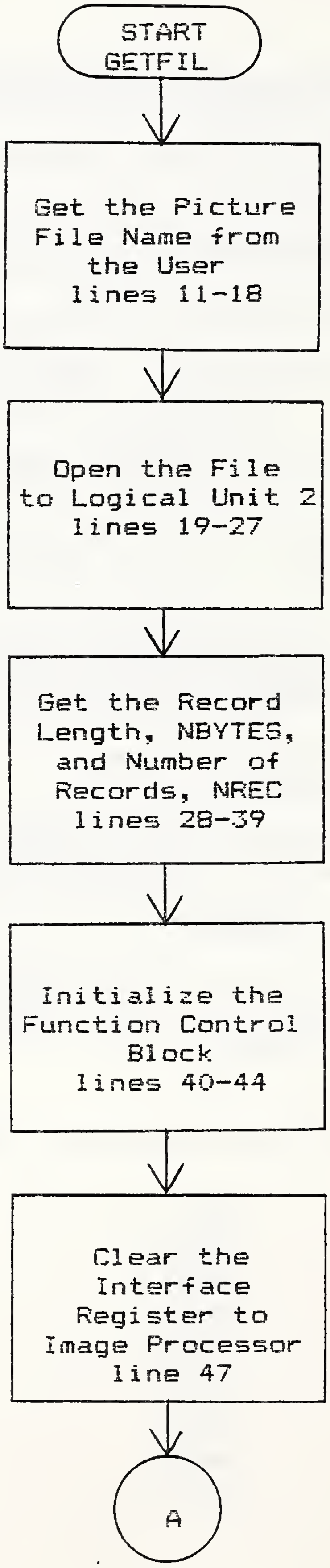
The parameters passed are:

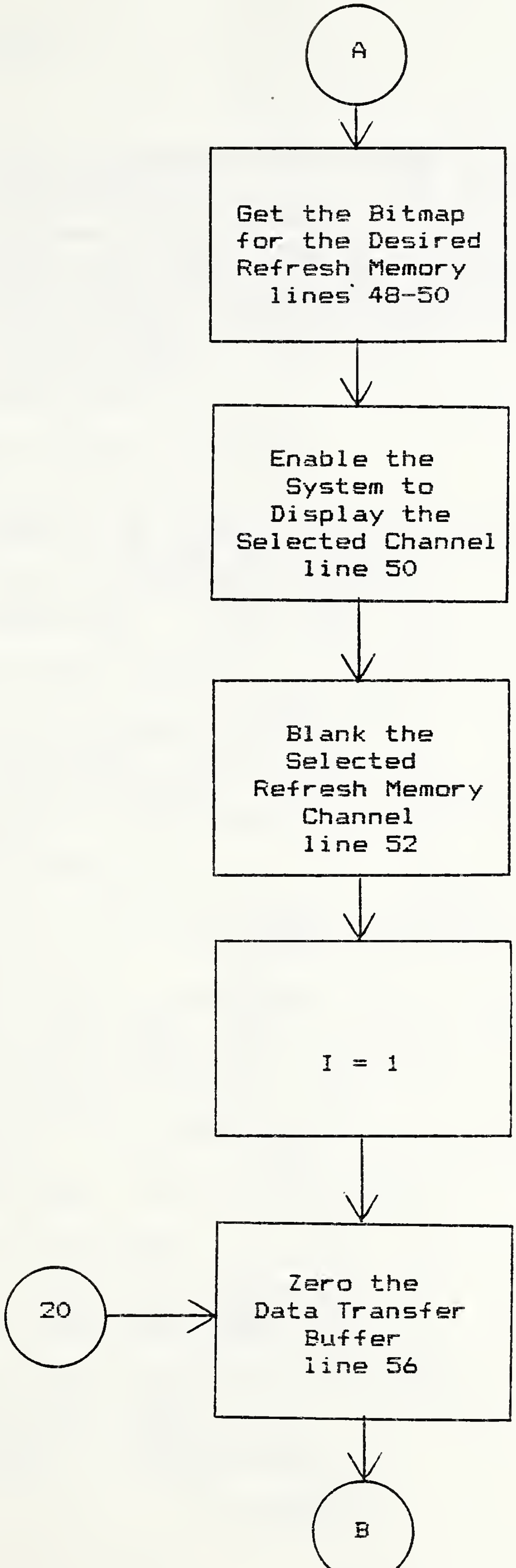
FCB	–	System Function Control Block for the image processor. INTEGER*2 Array
BUFFER	–	System buffer. INTEGER*2 Array
TABLE	–	Refresh memory number into which to write an image. Can be 1, 2, or 3. INTEGER*2
CHANLS	–	System channel mask for the selected refresh memory in TABLE. INTEGER*2

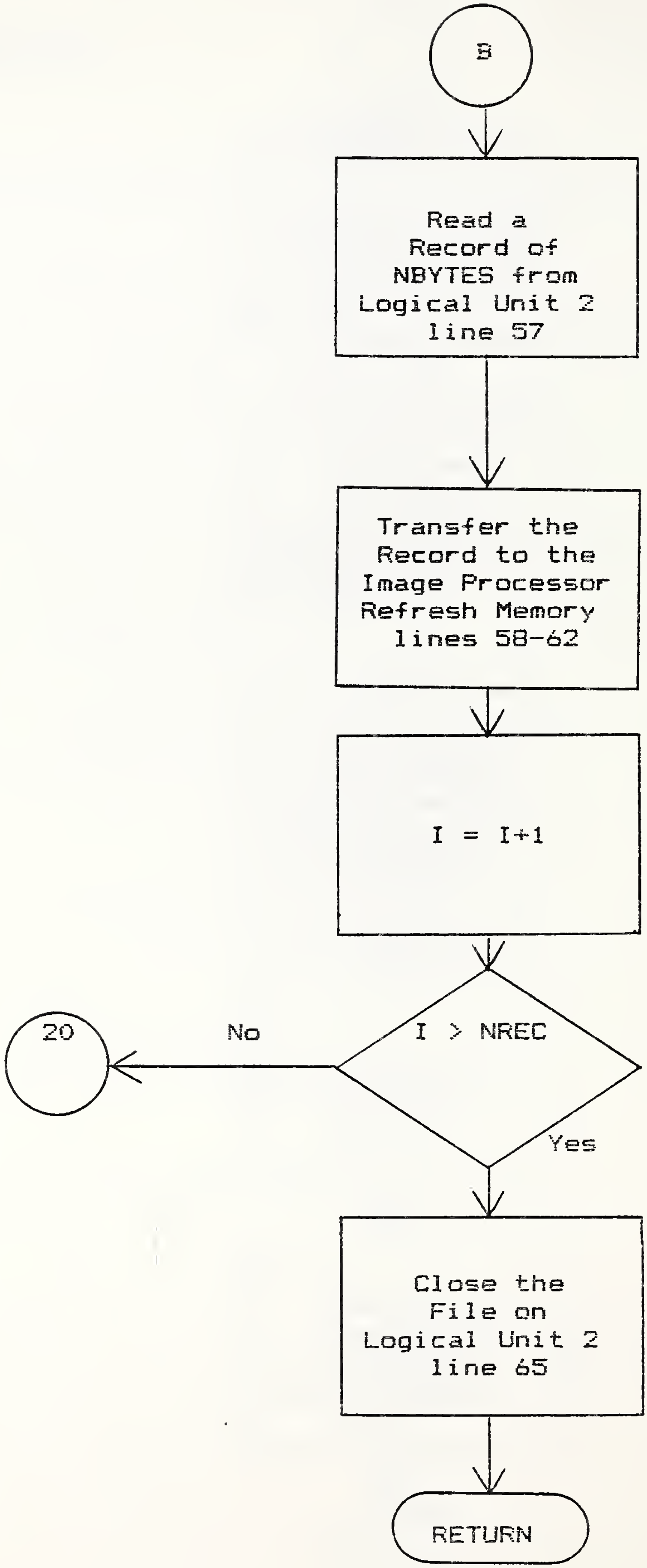
GETFIL calls the following subroutines or functions:

SVC7
ZBUFF
INFCB
MSTCL
DADRS
DUNIT
BCHAN
SYSIO
IMAGE
DMASK .

5.5.2 Flow Chart




A
Get the Bitmap for the Desired Refresh Memory lines 48-50
Enable the System to Display the Selected Channel line 50
Blank the Selected Refresh Memory Channel line 52
I = 1
20
Zero the Data Transfer Buffer line 56
B


B
Read a Record of NBYTES from Logical Unit 2 line 57
Transfer the Record to the Image Processor Refresh Memory lines 58-62
I = I+1
I > NREC
No
20
Yes
Close the File on Logical Unit 2 line 65
RETURN

5.5.3 Listing

```
C*********************************************************************
        SUBROUTINE GETFIL(FCB,BUFFER,TABLE,CHANLS)
C*********************************************************************
       INTEGER*2 FCB(2048), BUFFER(2048), CHANLS(16), TABLE(16)
       INTEGER BMIN,BMAX,GMIN,GMAX,RMIN,RMAX,BYPIFM,FMIN2,FMAX2
       INTEGER SS, SL, NL, NS, GRCODE, DMASK, CHCODE,CENTER
       INTEGER PACKED,EXT,ROTATE,DIRECT,BLANK
        INTEGER PBLK(8)
        INTEGER NP(8)
        CHARACTER*16 FD
C
C GET THE PICTURE FILE NAME
C
        WRITE(5,10)
10      FORMAT(' ENTER NAME OF IMAGE FILE YOU WISH TRANSFERRED',
     1          ' MAX OF 16 CHAR.')
        READ(5,11) FD
11      FORMAT(C16)
C
C OPEN THE FILE TO UNIT 2
C
        OPEN(2,FILE=FD,IOSTAT=IOS)
        IF(IOS .EQ. 0) GO TO 15
        WRITE(5,12) IOS
12      FORMAT(' IOSTAT ON OPENING FILE = ',I4)
        STOP
15      CONTINUE
C
C GET RECORD LENGTH IN BYTES AND NUMBER OF RECORDS
C
        NP(1) = 2
        INQUIRE(2,RECL=NBYTES,SIZE=NREC)
        CALL SVC7(NP)
        NBYTES= IAND(NP(2),Y'FFFF')
        IF(NBYTES .LE. 4096) GO TO 18
        WRITE(5,16)
16      FORMAT(' RECORD LENGTH OF FILE IS GREATER THAN 4096 BYTES')
        STOP
18      CONTINUE
C
C INITIALIZE THE I2S
C
       CALL ZBUFF (FCB, 16)
       CALL INFCB(FCB, 2000, 3)
C
C     CLEAR DEVICE TO READY FOR WRITING
       CALL MSTCL (FCB)
        GRCODE = DMASK(15)
        TABLE(1) = TABLE(1) - 1
```

```
      CALL DADRS (CHANLS, TABLE, CHCODE, 1)
      CALL DUNIT (FCB, BUFFER, TABLE, 1, 256)
       CALL BCHAN(FCB,BUFFER,CHCODE,-1)
      PACKED = 1
C
      DO 20 I = 1,NREC
         CALL ZBUFF (BUFFER, 2048)
          CALL SYSIO(PBLK,89,2,BUFFER,NBYTES,0)
         LGTREC = 512
         IF (NBYTES .LT. 512) LGTREC = NBYTES
         CALL IMAGE (FCB, BUFFER, 0, (I-1),
     1               LGTREC, DIRECT, CHCODE, -1, PACKED, 1,
     1               0, 0, 0, 0, 0)
 20    CONTINUE
C
       CLOSE(2)
      RETURN
      END
```

5.6 Subroutine GETCHN

5.6.1 Summary

This subroutine initializes a specified channel and enables the registers for that selected channel so that an image may be displayed. No image is actually transferred. The calling sequence for this subroutine is:

```
CALL GETCHN (FCB, BUFFER, TABLE, CHANLS).
```

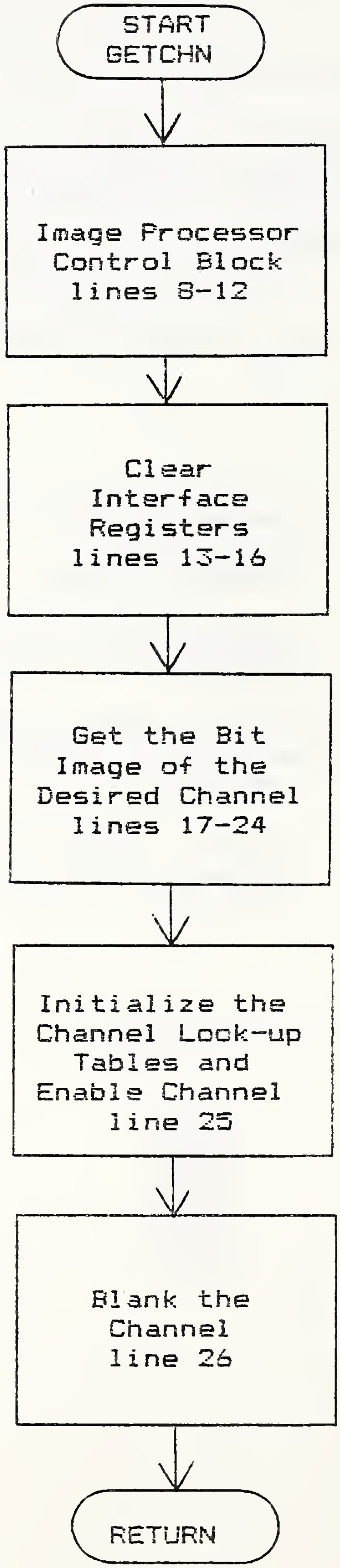
The parameters passed are:

FCB	–	System Function Control Block array. INTEGER*2 Array
BUFFER	–	System buffer array. INTEGER*2
TABLE	–	Refresh memory number. Set to 1, 2 or 3. INTEGER*2
CHANLS	–	Channel mask for the refresh memory in TABLE. INTEGER*2

GETCHN calls the following subroutines:

ZBUFF
INFCB
MSTCL
DADRS
DUNIT
BCHAN .

5.6.2 Flow Chart



5.6.3 Listing

```
C******************************************************************
      SUBROUTINE GETCHN(FCB,BUFFER,TABLE,CHANLS)
C******************************************************************
      INTEGER*2 FCB(2048),BUFFER(2048)
      INTEGER*2 CHANLS(16)
      INTEGER*2 TABLE(16)
      INTEGER CHCODE
C
C INITIALIZE THE I2S
C
      CALL ZBUFF(FCB,16)
      CALL INFCB(FCB,2000,3)
C
C CLEAR DEVICE TO READY FOR WRITING
C
      CALL MSTCL(FCB)
C
C MAKE CHANNEL 2 THE SHADOW GRAPH CHANNEL
C
      TABLE(1) = TABLE(1) - 1
C
C INITIALIZE REGISTERS AND LOOK-UP TABLES
C
      CALL DADRS(CHANLS,TABLE,CHCODE,1)
      CALL DUNIT(FCB,BUFFER,TABLE,1,256)
      CALL BCHAN(FCB,BUFFER,CHCODE,-1)
      RETURN
      END
```

5.7 Subroutine GNXY

5.7.1 Summary

Given a point in a unit square in the XY-plane and a direction vector (W(1),W(2)), this subroutine determines whether the point is interior to the square or on the boundary. If it is interior to the square, then the subroutine returns the exit boundary point of the directed line through the point with direction vector (W(1),W(2)). If it is a boundary point, then the direction vector (W(1),W(2)) either points inward or outward from the square. If inward, then the subroutine returns the exit point from the same square. If the direction vector points outward, then the subroutine returns the exit point of the neighboring square through which the directed line passes. The calling sequence for this subroutine is

CALL GNXY (X, Y, IX, IY, W, IFLG) .

GNXY passes the following parameters:

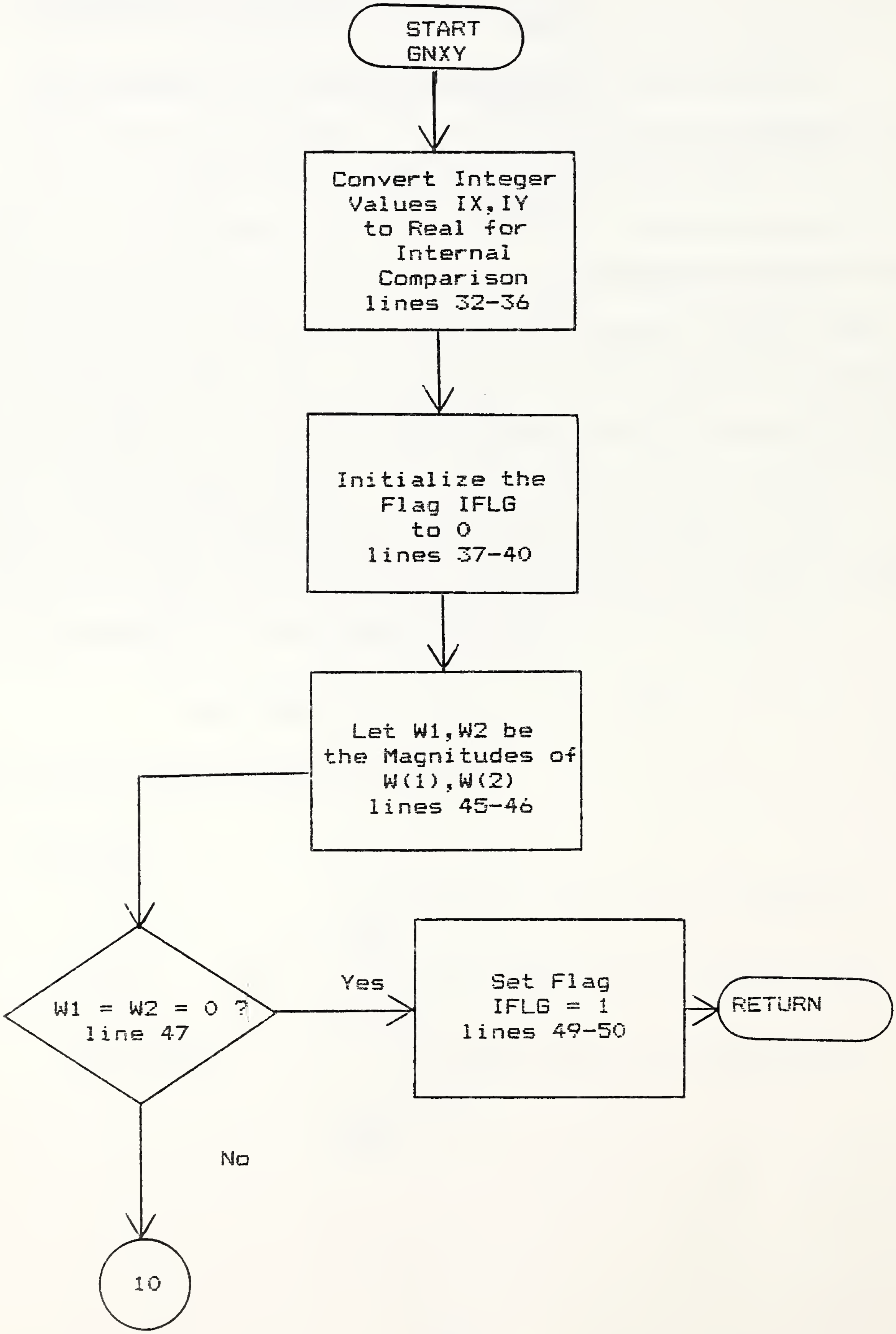
ON INPUT	–	
X,Y	–	Components of the point of interest. REAL
IX,IY	–	Truncated values of X,Y respectively. INTEGER
W(1),W(2)	–	X,Y components of the 3-D direction vector $\vec{W}$. REAL

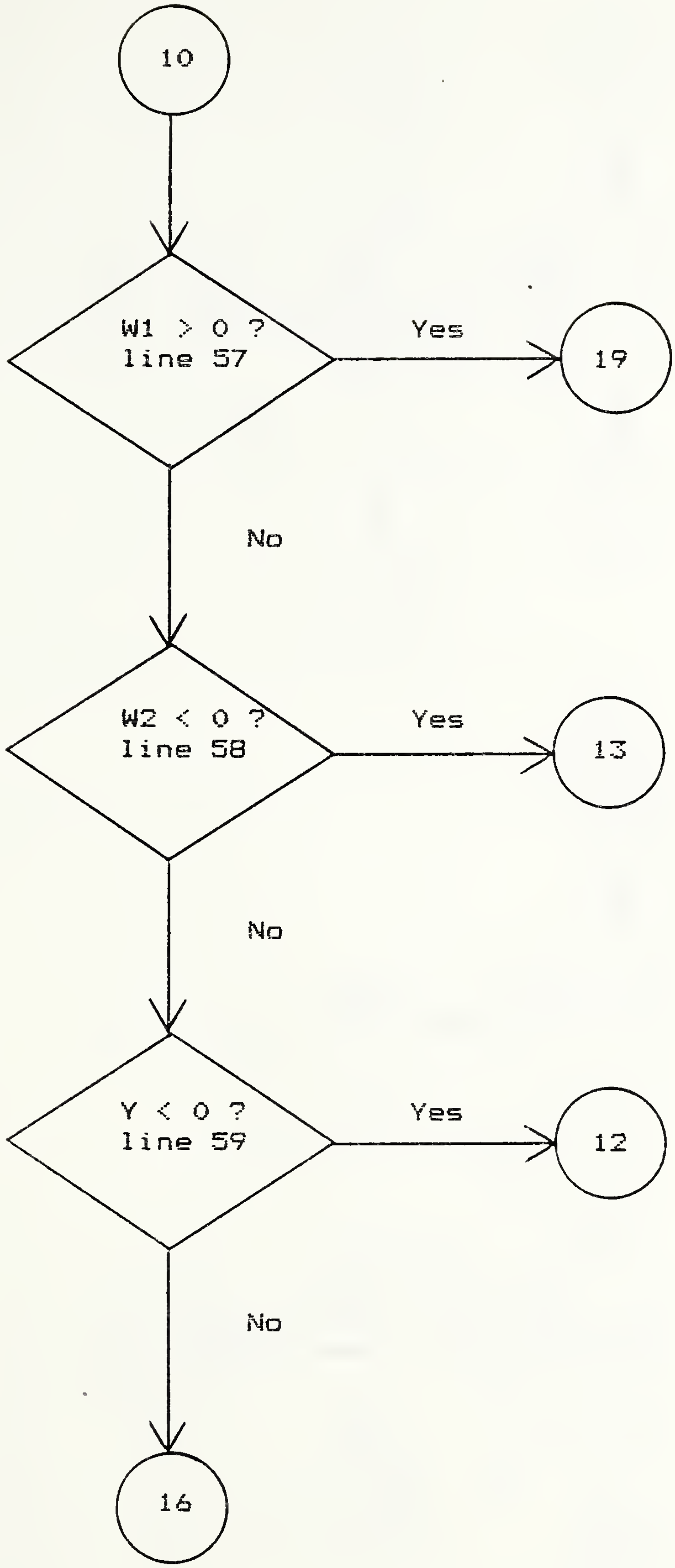
ON OUTPUT -

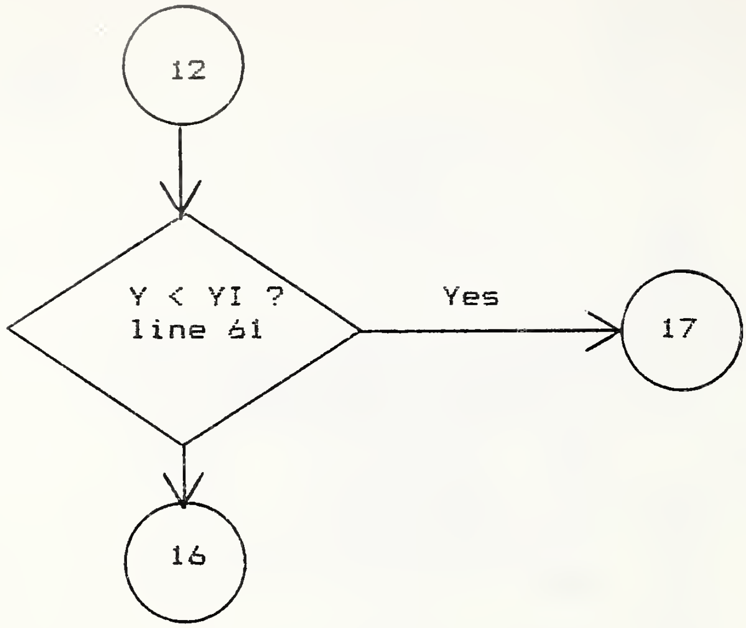
X,Y	–	X,Y components of exit point for the unit square of adjacent unit square. REAL
IX,IY	–	Truncated values of X,Y respectively. INTEGER
IFLG	–	= 1 if W(1) = W(2) = 0 = 0 otherwise . INTEGER

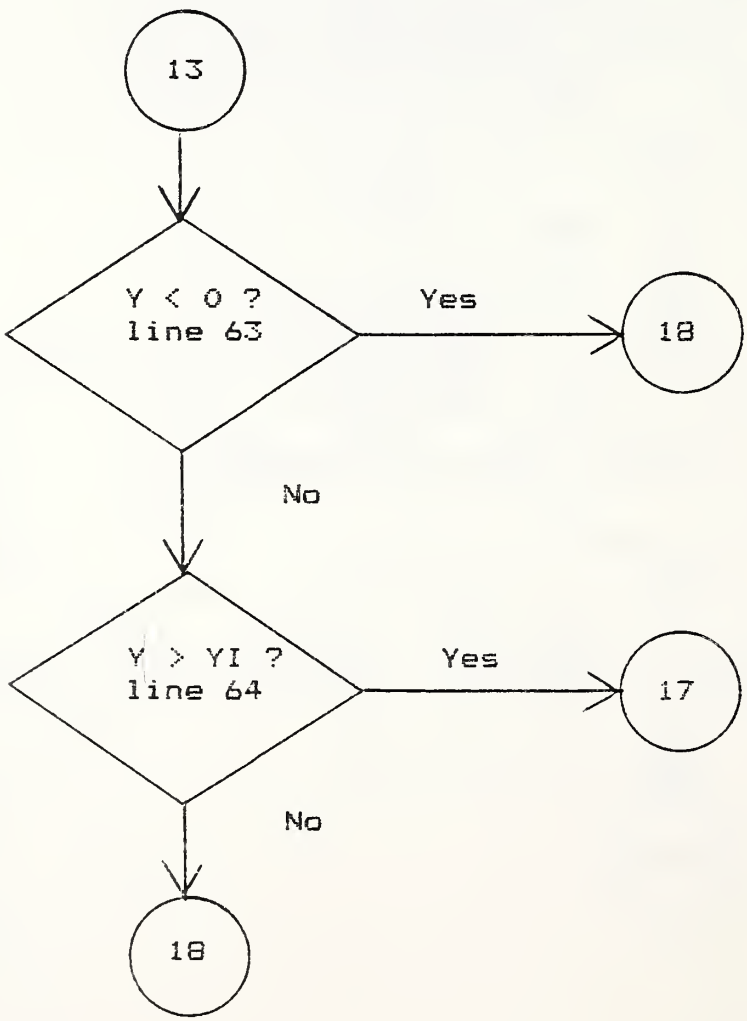
No subroutines are called.

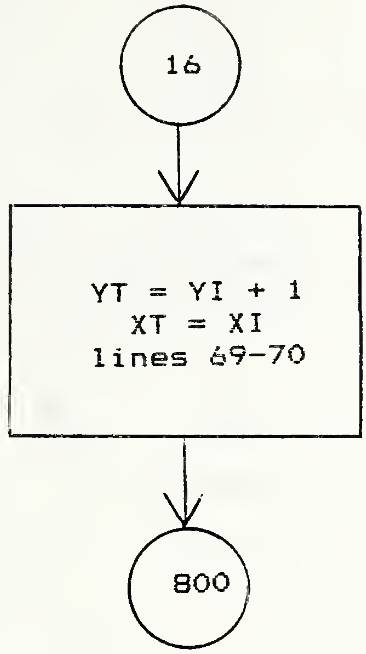
5.7.2 Flow Chart

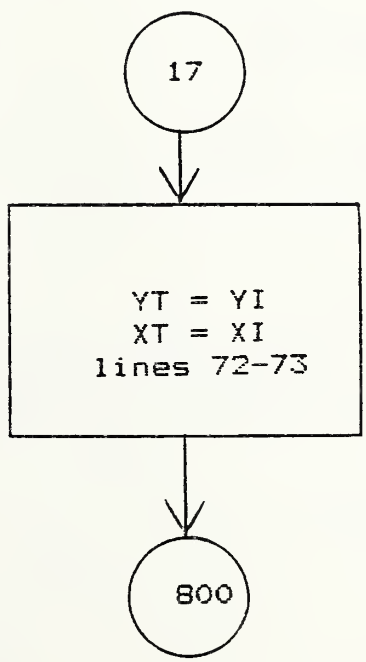


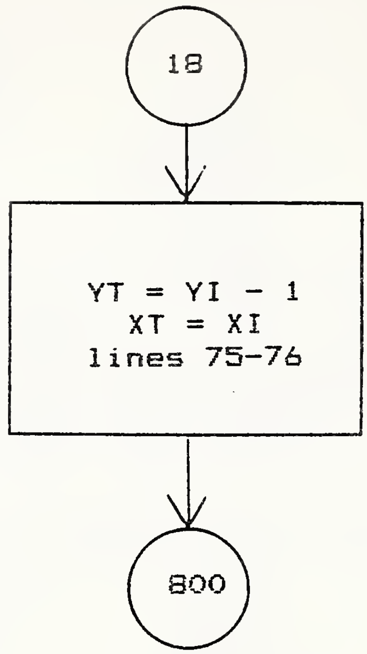

10
W1 > 0 ?
line 57
Yes
19
No
W2 < 0 ?
line 58
Yes
13
No
Y < 0 ?
line 59
Yes
12
No
16


12
Y < YI ?
line 61
Yes
17
16

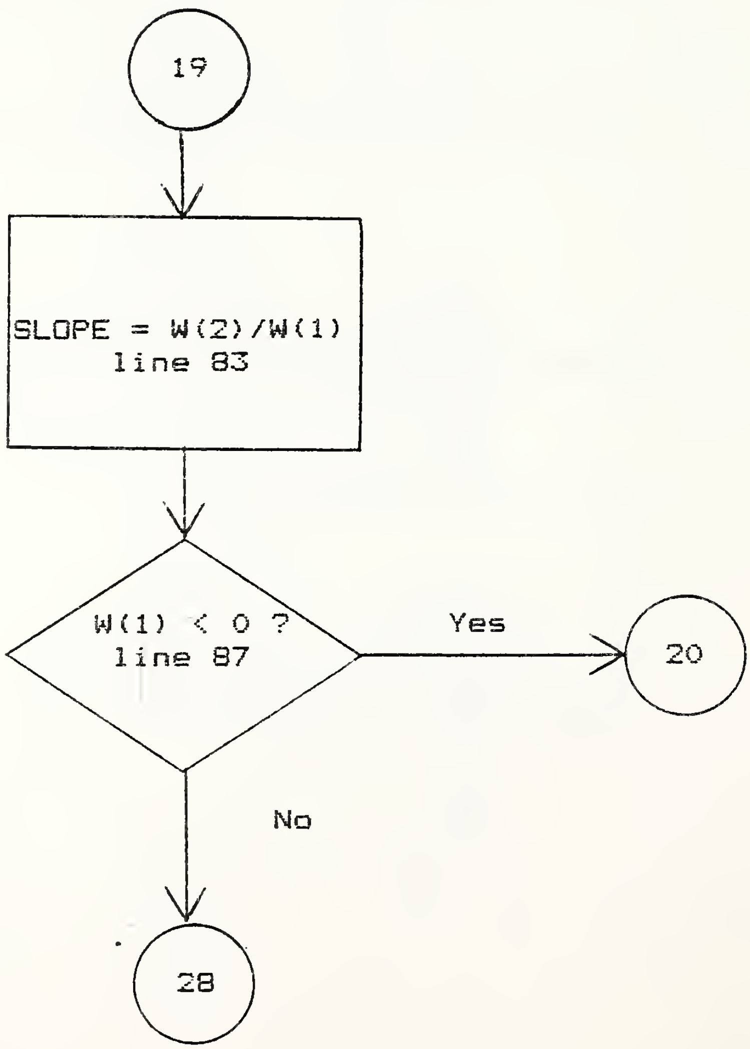

13
Y < 0 ?
line 63
Yes
18
No
Y > YI ?
line 64
Yes
17
No
18


16
YT = YI + 1
XT = XI
lines 69-70
800

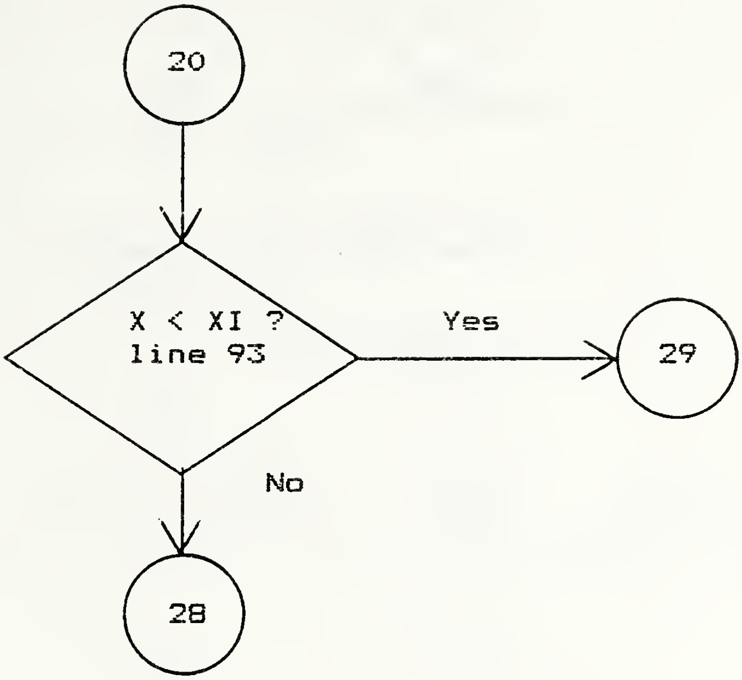

17
YT = YI
XT = XI
lines 72-73
800

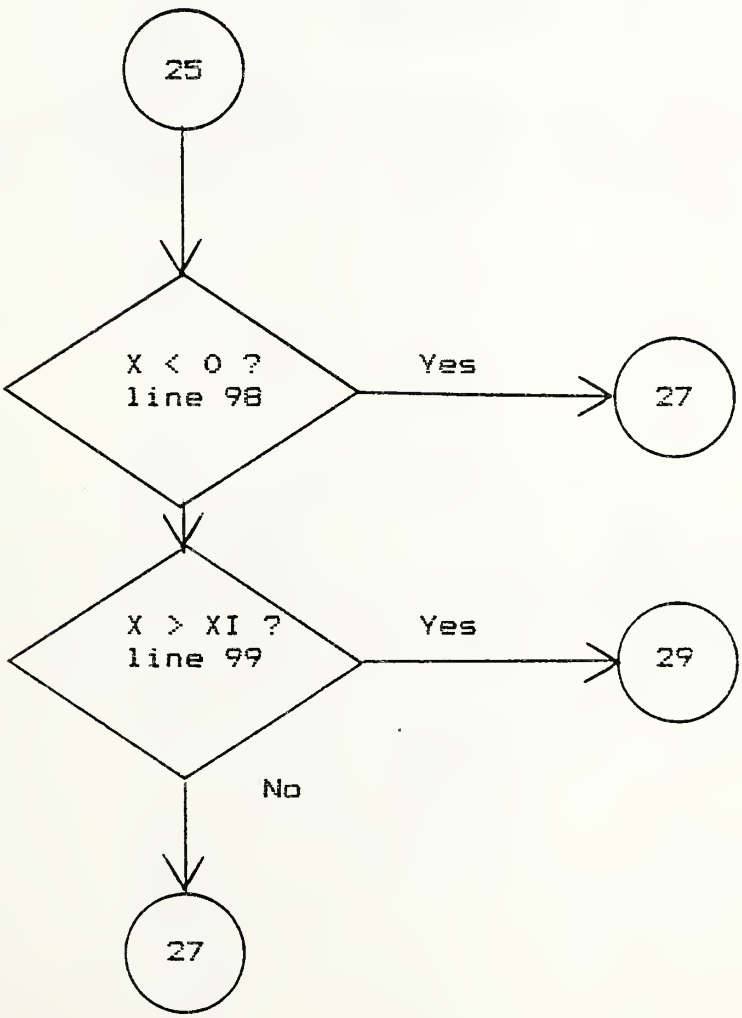


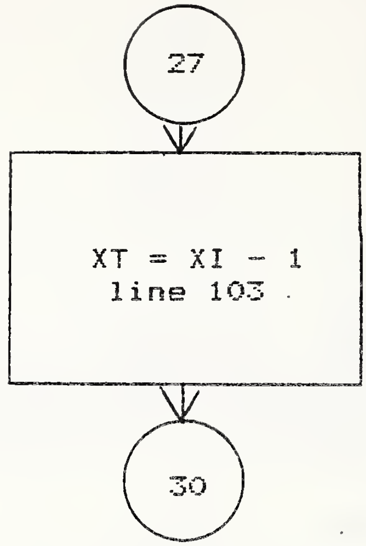
18
YT = YI - 1
XT = XI
lines 75-76
800

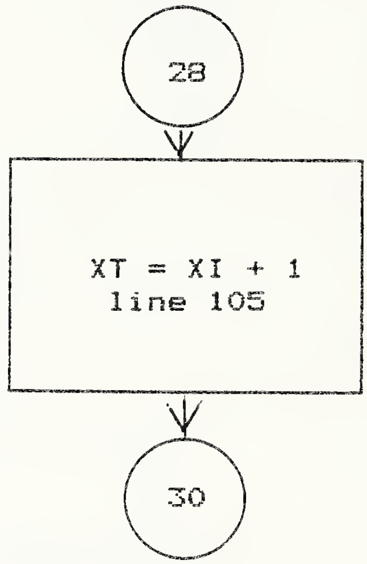


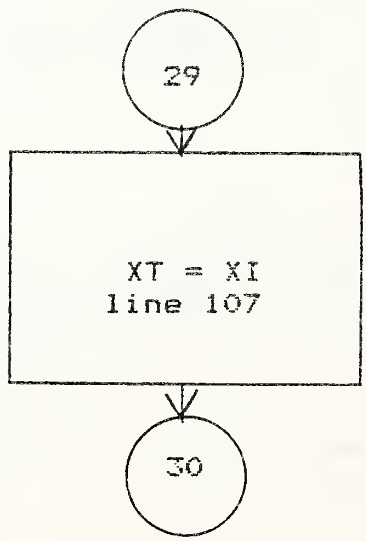
19
SLOPE = W(2)/W(1)
line 83
W(1) < 0 ?
line 87
Yes
20
No
28

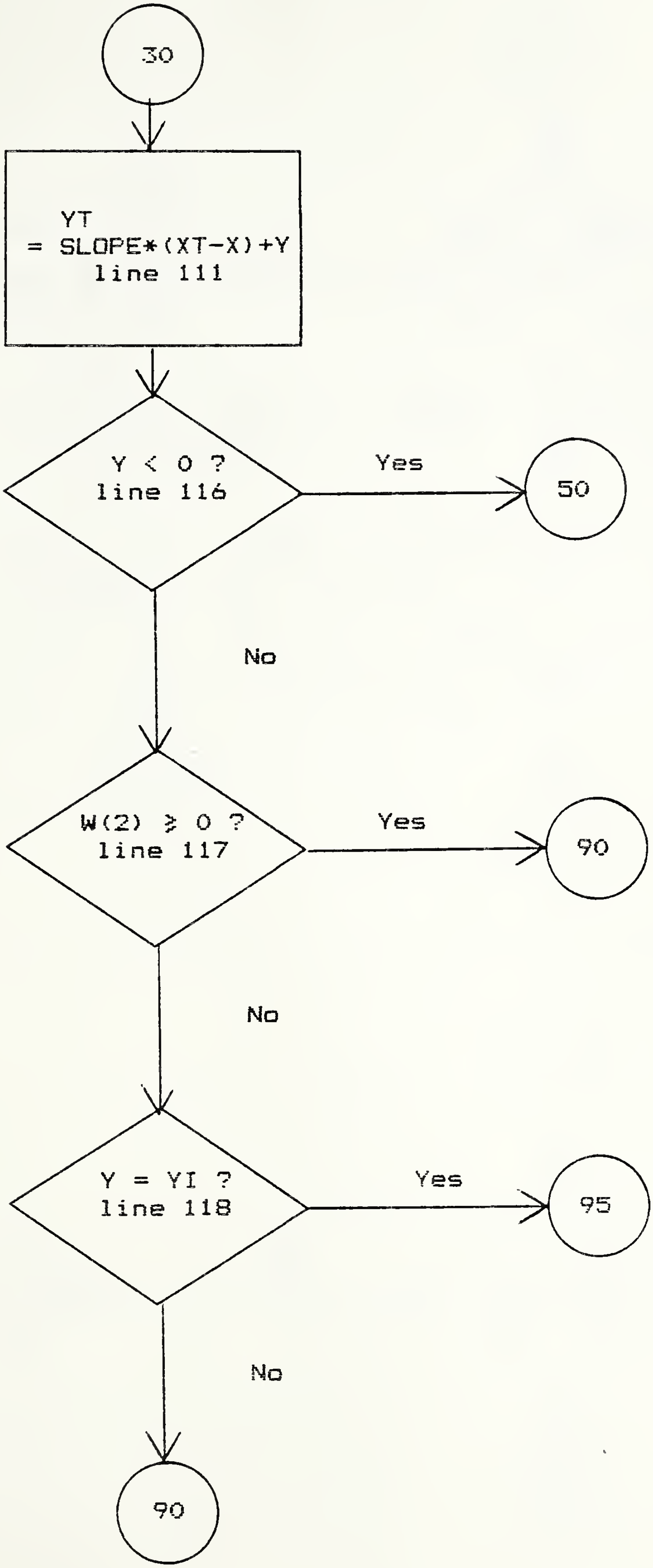

20
X < XI ?
line 93
Yes
29
No
28

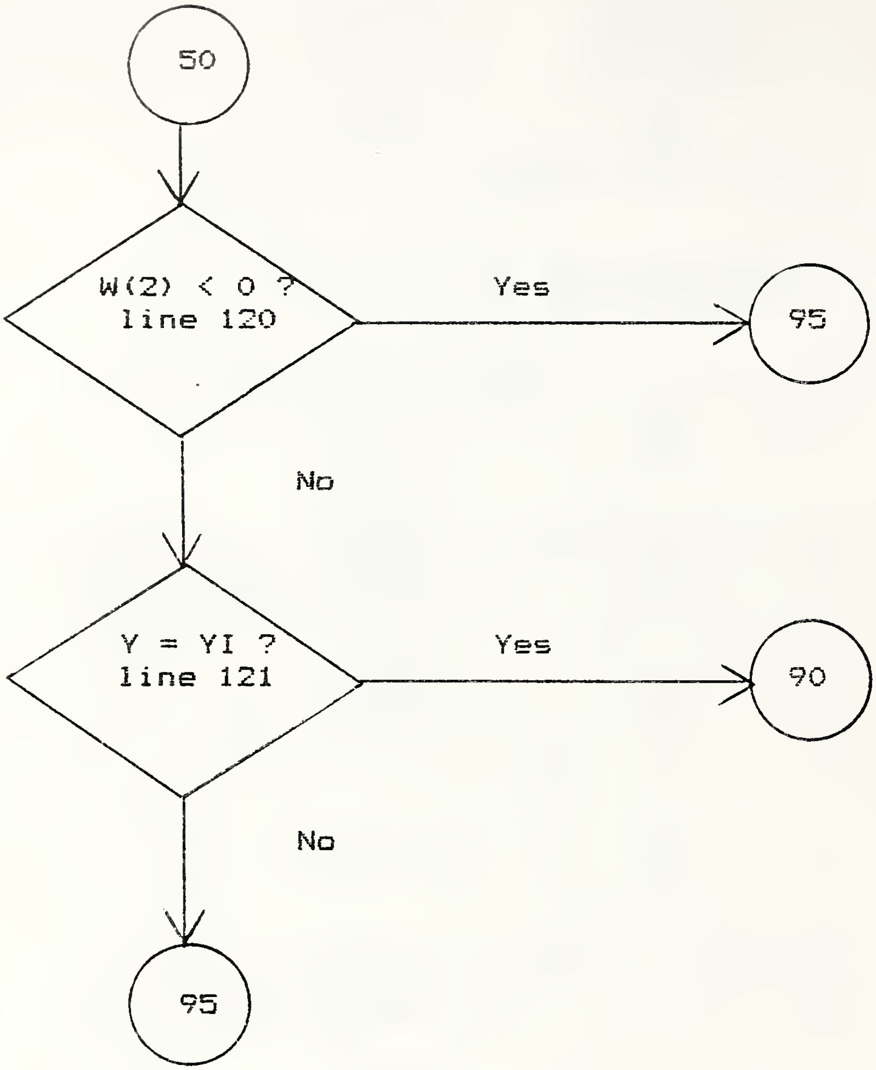

25
X < 0 ?
line 98
Yes
27
X > XI ?
line 99
Yes
29
No
27

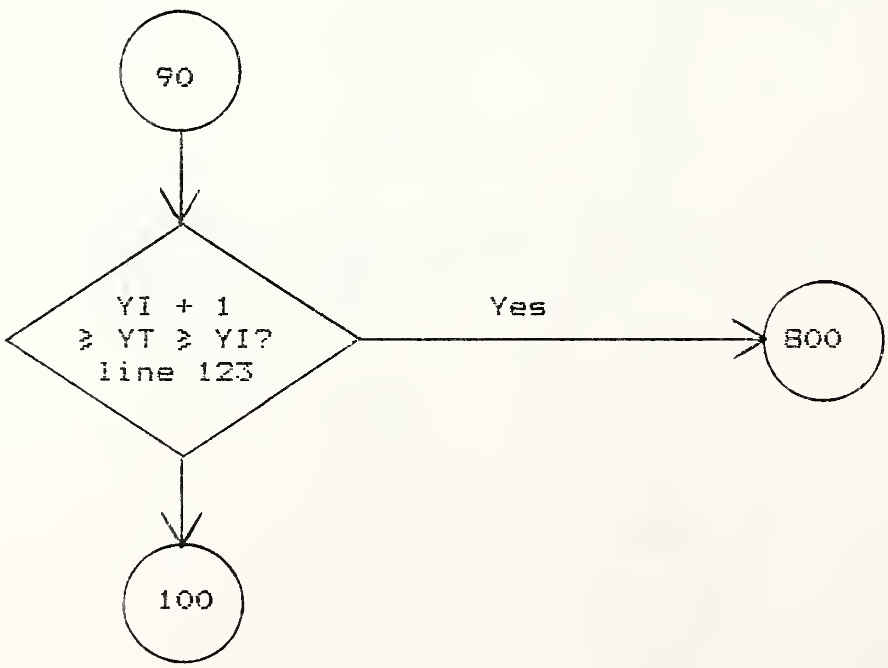

27
XT = XI - 1
line 103
30

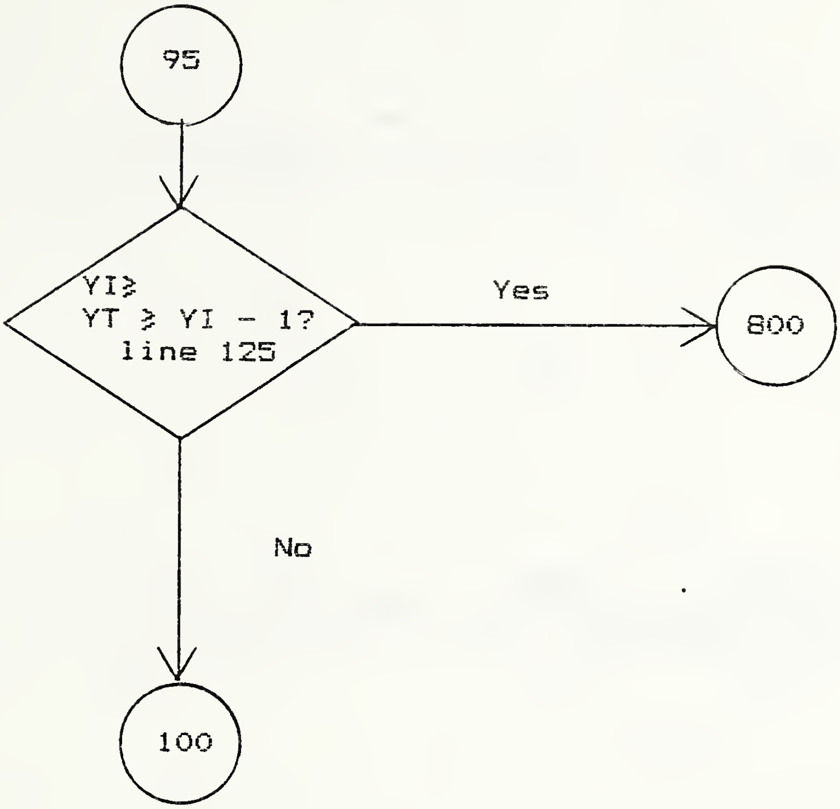

28
XT = XI + 1
line 105
30

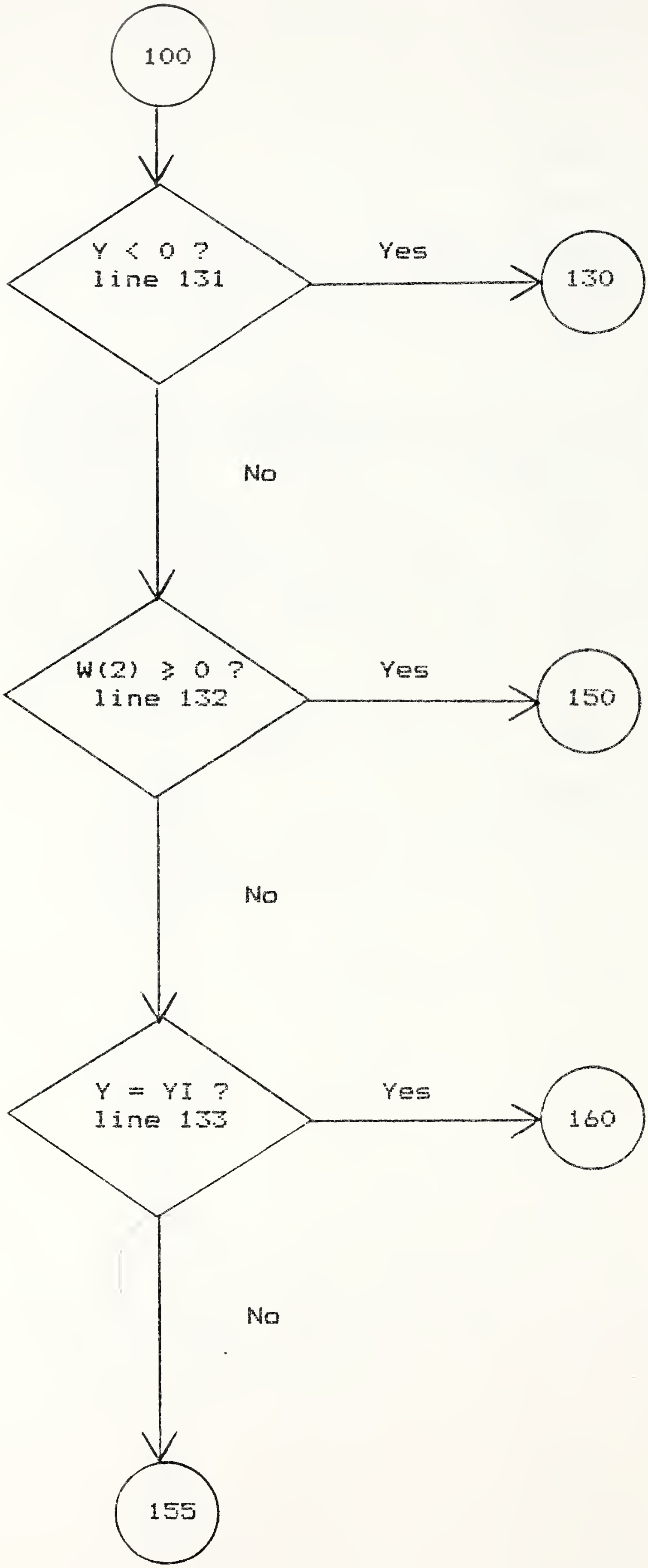

29
XT = XI
line 107
30

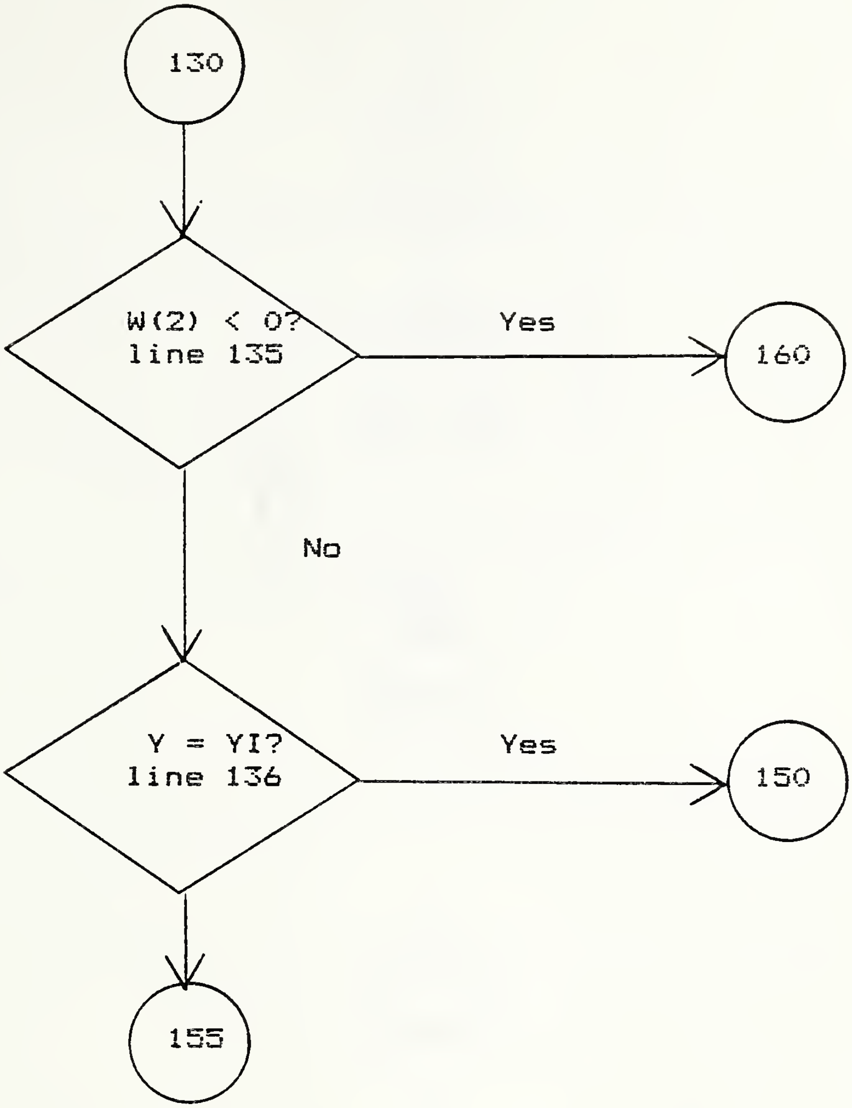

30
YT
= SLOPE*(XT-X)+Y
line 111
Y < 0 ?
line 116
Yes
50
No
W(2) ≥ 0 ?
line 117
Yes
90
No
Y = YI ?
line 118
Yes
95
No
90

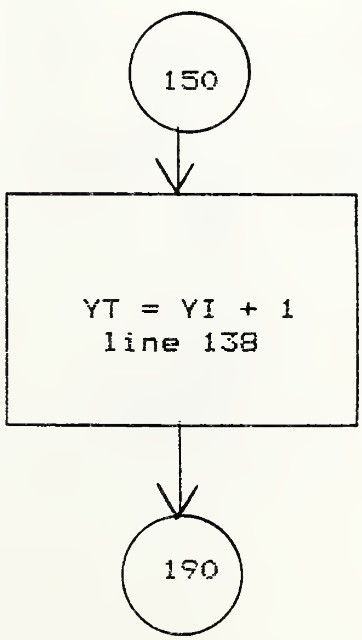

50
W(2) < 0 ?
line 120
Yes
95
No
Y = YI ?
line 121
Yes
90
No
95

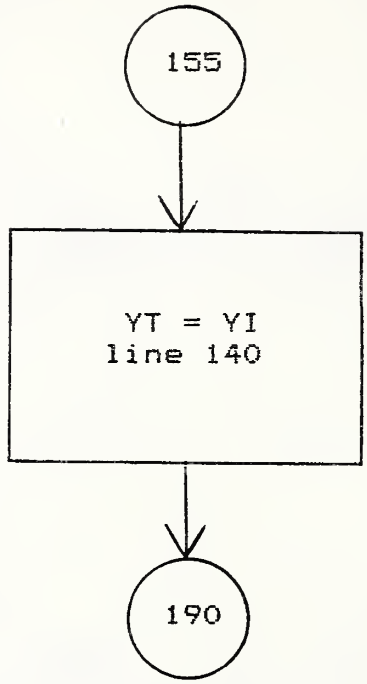

90
YI + 1
≥ YT ≥ YI?
line 123
Yes
800
100

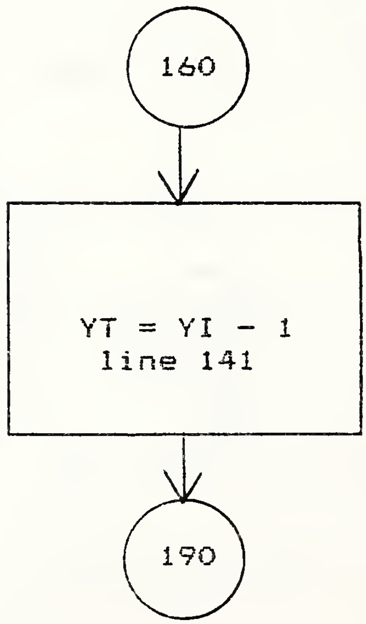

95
YI≥
YT ≥ YI - 1?
line 125
Yes
800
No
100

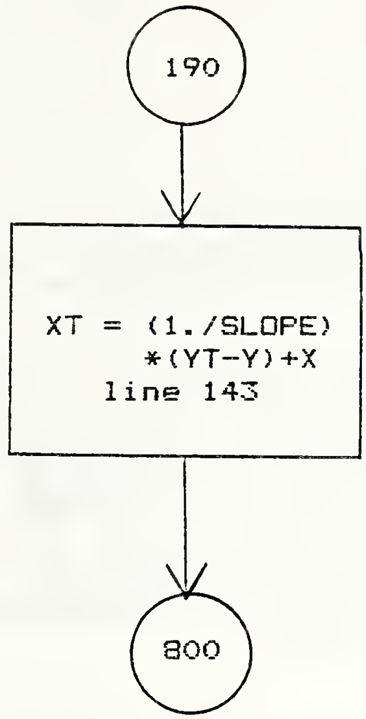

100
Y < 0 ?
line 131
Yes
130
No
W(2) ≥ 0 ?
line 132
Yes
150
No
Y = YI ?
line 133
Yes
160
No
155

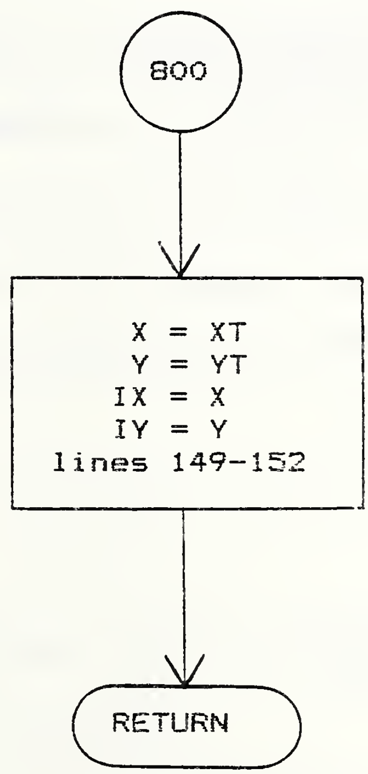

130
W(2) < 0?
line 135
Yes
160
No
Y = YI?
line 136
Yes
150
155


150
YT = YI + 1
line 138
190


155
YT = YI
line 140
190


160
YT = YI - 1
line 141
190


190
XT = (1./SLOPE)
*(YT-Y)+X
line 143
800


800
X = XT
Y = YT
IX = X
IY = Y
lines 149-152
RETURN

5.7.3 Listing

```
C**********************************************************************
      SUBROUTINE GNXY(X,Y,IX,IY,W,IFLG)
C**********************************************************************
C
C FUNCTION:
C     GIVEN A POINT IN A UNIT SQUARE AND A DIRECTION VECTOR,
C DETERMINE WHETHER THE POINT IS INTERIOR TO THE SQUARE OR ON
C THE BOUNDARY. IF IT IS INTERIOR, THEN RETURN THE EXIT
C BOUNDARY POINT OF THE DIRECTED LINE THROUGH THE POINT
C WITH DIRECTION VECTOR W.  IF IT IS A BOUNDARY POINT THEN THE
C DIRECTION VECTOR W EITHER POINTS INWARD OR OUTWARD.  IF INWARD,
C THEN RETURN THE EXIT POINT OF THE SAME UNIT SQUARE.  IF OUTWARD,
C THEN RETURN THE EXIT POINT OF THE NEIGHBORING UNIT SQUARE THROUGH
C WHICH THE DIRECTED LINE PASSES.
C
C INPUT:
C   X,Y    -    X,Y COMPONENTS OF POINT OF INTEREST
C   IX,IY  -    TRUNCATED VALUES OF X,Y RESPECTIVELY.
C               REPRESENTS THE CORNER OF THE UNIT SQUARE.
C   W(1),W(2)-  X,Y COMPONENTS OF A 3-D DIRECTION VECTOR.
C
C OUTPUT:
C   X,Y    -    X,Y COMPONENTS OF EXIT POINT FROM THE UNIT
C               SQUARE OR ADJACENT SQUARE.
C   IX,IY  -    NEW TRUNCATED VALUES
C   IFLG   -    = 0 IF THE PROPER POINT IS RETURNED
C               = 1 IF W(1) = 0, W(2) = 0
C
C**********************************************************************
      REAL W(3), X, Y
      INTEGER IFLG, IX, IY
C
C CONVERT INTEGER VALUES TO REAL FOR INTERNAL COMPARISONS
C
      XI = IX
      YI = IY
C
C INITIALIZE IFLG TO 0
C
      IFLG = 0
C
C IF BOTH W(1) AND W(2) ARE SMALL IN MAGNITUDE RETURN IFLG=1
C THIS INDICATES A STABLE POINT.
C
      W1 = ABS(W(1))
      W2 = ABS(W(2))
      IF (W1 .LE. 5.E-6 .AND. W2 .LE. 5.E-6) GO TO 5
      GO TO 10
5     CONTINUE
      IFLG = 1
      RETURN
```

```
C---------------------------------------------------------------------
C
C WHEN W(1) = 0 AND W(2) <> 0 USE THIS SECTION OF CODE
C
C---------------------------------------------------------------------
10       IF (W1 .GT. 5.E-6) GO TO 19
         IF (W(2) .LT. 0.) GO TO 13
         IF (Y .LT. 0.) GO TO 12
         GO TO 16
12       IF (Y .LT. YI) GO TO 17
         GO TO 16
13       IF (Y .LT. 0.) GO TO 18
         IF (Y .GT. YI) GO TO 17
         GO TO 18
C
C SET UP THE VERTICAL INTERCEPT Y VALUE
C
16       YT = YI + 1.
         XT = XI
         GO TO 800
17       YT = YI
         XT = XI
         GO TO 800
18       YT = YI - 1.
         XT = XI
         GO TO 800
C---------------------------------------------------------------------
C
C USE THIS SECTION OF CODE FOR W(1) <> 0
C
C---------------------------------------------------------------------
19       SLOPE = W(2)/W(1)
C
C STEP IN X TO THE NEXT UNIT BOUNDARY LINE
C
         IF (W(1) .LT. 0.) GO TO 25
C
C ENTER HERE IF W(1) > 0
C
         IF (X .LT. 0.) GO TO 20
         GO TO 28
20       IF (X .LT. XI) GO TO 29
         GO TO 28
C
C ENTER HERE IF W(1) < 0
C
25       IF (X .LT. 0.) GO TO 27
         IF (X .GT. XI) GO TO 29
C
C SET UP THE X VALUE FOR THE BOUNDARY INTERCEPT
C
```

```
27      XT = XI - 1.
        GO TO 30
28      XT = XI + 1.
        GO TO 30
29      XT = XI
C
C SET UP Y VALUE FOR THE BOUNDARY INTERCEPT
C
30      YT = SLOPE * (XT -X) + Y
C
C DOES THE DIRECTED LINE CROSS THE BOUNDARY LINE OUTSIDE OF THE
C UNIT SQUARE OF INTEREST?
C
        IF (Y .LT. 0.) GO TO 50
        IF (W(2) .GE. 0.) GO TO 90
        IF (Y .EQ. YI) GO TO 95
        GO TO 90
50      IF (W(2) .LT. 0.) GO TO 95
        IF (Y .EQ. YI) GO TO 90
        GO TO 95
90      IF (YI + 1. .GE. YT .AND. YT .GE. YI) GO TO 800
        GO TO 100
95      IF (YI .GE. YT .AND. YT .GE. YI - 1.) GO TO 800
C
C IF THE BOUNDARY IS CROSSED OUTSIDE OF THE UNIT SQUARE OF
C INTEREST FIND THE LARGEST X STEP THAT KEEPS IT WITHIN THE
C SQUARE.
C
100     IF (Y .LT. 0.) GO TO 130
        IF (W(2) .GE. 0.) GO TO 150
        IF (Y .EQ. YI) GO TO 160
        GO TO 155
130     IF (W(2) .LT. 0.) GO TO 160
        IF (Y .EQ. YI) GO TO 150
        GO TO 155
150     YT = YI + 1.
        GO TO 190
155     YT = YI
        GO TO 190
160     YT = YI - 1.
190     XT = (1./SLOPE) * (YT - Y) + X
C-----------------------------------------------------------------
C
C THIS UNIT OF CODE SETS UP THE OUTPUT VARIABLES AND RETURNS
C
C-----------------------------------------------------------------
800     X = XT
        Y = YT
        IX = X
        IY = Y
        RETURN
        END
```

5.8 Subroutine WRPIC

5.8.1 Summary

This subroutine transfers NPIXEL number of pixels to the image processor channel refresh memory, with bitmap channel number in FILE(3), beginning in IROW row and ICOL column and proceeding to the right. The error flag is not used in this version. The data is transferred through the array BUF with one pixel per word. The calling sequence is:

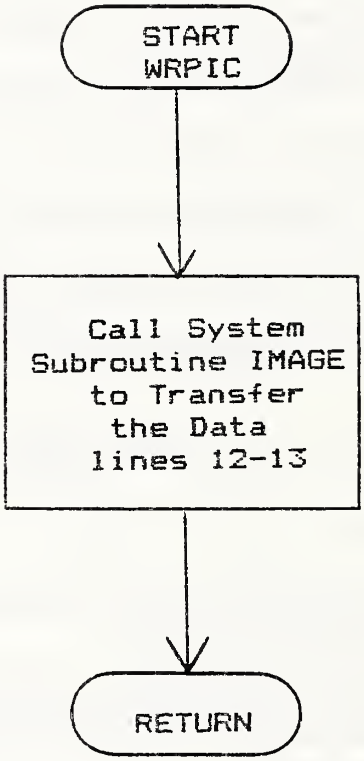
```
CALL WRPIC (FCB, FILE, BUF, IROW, ICOL, NPIXEL, IERR).
```

The parameters passed are:

FCB	–	System Function Control Block. INTEGER*2 Array
FILE	–	Array containing the bitmap for the desired refresh memory in element 3. INTEGER Array
BUF	–	A buffer array that contains the transferred pixel data one pixel per word. INTEGER*2 Array
IROW	–	Row index from 0 to 511. INTEGER
ICOL	–	Column index from 0 to 511. INTEGER
NPIXEL	–	Number of pixels to transfer. INTEGER
IERR	–	Error flag. Not used.

WRPIC calls the subroutine IMAGE.

5.8.2 Flow Chart



5.8.3 Listing

```
C*******************************************************************
      SUBROUTINE WRPIC(FCB,FILE,BUF,IROW,ICOL,NPIXEL,IERR)
C*******************************************************************
      INTEGER*2 BUF(NPIXEL),FCB(2048)
      INTEGER FILE(7)
C
C     IMAGE METROLOGY WRPIC   7/30/80
C     WRITES FROM DISPLAY DEVICE
C     FILE(3) IS CHANNEL NUMBER
C     WRITES A ROW FROM LEFT TO RIGHT
C
      CALL IMAGE(FCB,BUF,ICOL,IROW,NPIXEL,0,
     *FILE(3),-1,0,1,0,0,0,0,0)
      IERR = 0
      RETURN
      END
```

5.9 Subroutine RDPIC

5.9.1 Summary

This subroutine transfers NPIXEL number of pixels from the image processor refresh memory with bitmap in FILE(3) to BUF, one byte per word, beginning in IROW row and ICOL column. The error flag is not used in this version. The calling sequence for this subroutine is:

```
CALL RDPIC (FCB, FILE, BUF, IROW, ICOL, NPIXEL, IERR).
```

The parameters passed are:

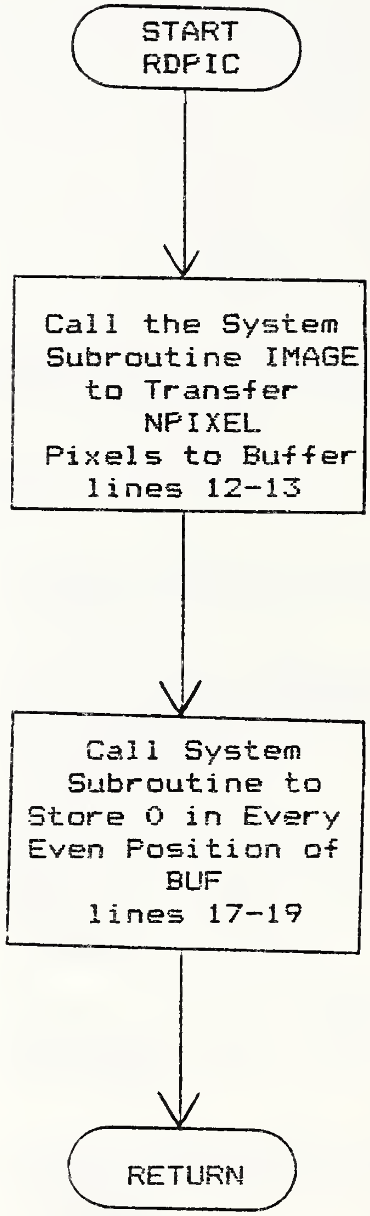
FCB	–	System Function Control Block. INTEGER*2 Array
FILE	–	Array that contains the bitmap for the desired refresh memory in FILE(3). INTEGER Array
BUF	–	Buffer array that receives the data from the transfer, one byte per word. INTEGER*2 Array
IROW	–	Beginning row number of the refresh memory for data transfer. INTEGER
ICOL	–	Beginning column number of the refresh memory for data transfer. INTEGER
NPIXEL	–	Number of pixels to transfer. INTEGER
IERR	–	Error flag. Not used in this version.

RDPIC calls the following subroutines:

IMAGE

ISBYTE .

5.9.2 Flow Chart



5.9.3 Listing

```
C*********************************************************************
      SUBROUTINE RDPIC(FCB,FILE,BUF,IROW,ICOL,NPIXEL,IERR)
C*********************************************************************
      INTEGER*2 BUF(NPIXEL),FCB(2048)
      INTEGER FILE(7)
C
C     IMAGE METROLOGY RDPIC   7/30/80
C     READS FROM DISPLAY DEVICE
C     FILE(3) IS CHANNEL NUMBER
C     READS A ROW FROM LEFT TO RIGHT
C
      CALL IMAGE(FCB,BUF,ICOL,IROW,NPIXEL,0,
     *FILE(3),255,0,1,0,0,0,0,1)
C
C   TEMPORARY FIX FOR UNPACKED READ
C
      DO 10 I=1,NPIXEL
         CALL ISBYTE(0,BUF,2*(I-1))
 10      CONTINUE
C
C   ***********************************
C
      IERR = 0
      RETURN
      END
```

5.10 Subroutine EXTREM

5.10.1 Summary

Based upon the direction vector $\vec{W}$ of the rays (either light or viewer) and the vertices of the rectangle of interest, this subroutine returns, in the arrays EX,EY, the extreme points seen by the rays. The calling sequence for this subroutine is:

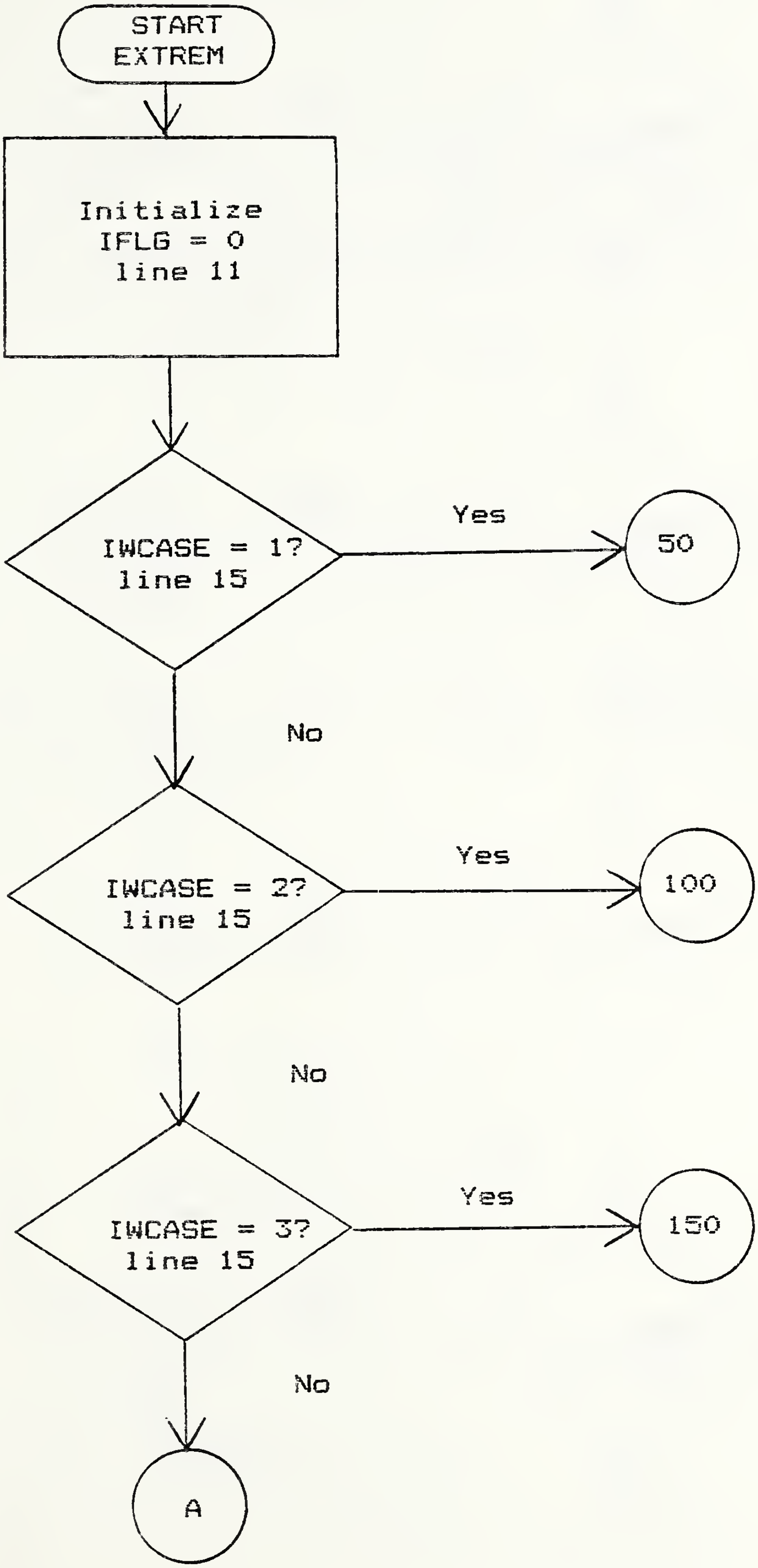
```
CALL EXTREM (IWCASE, VRX, VRY, EX, EY, IFLG) .
```

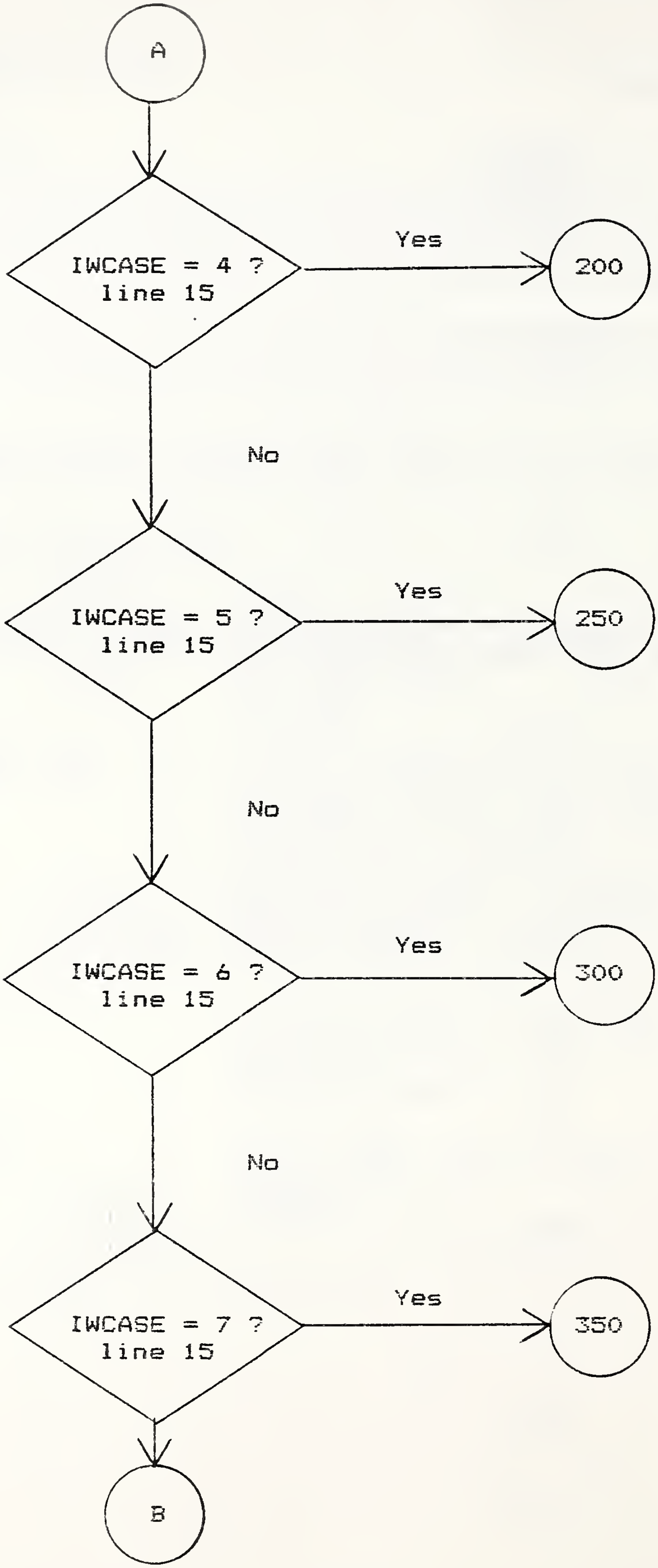
The parameters passed are:

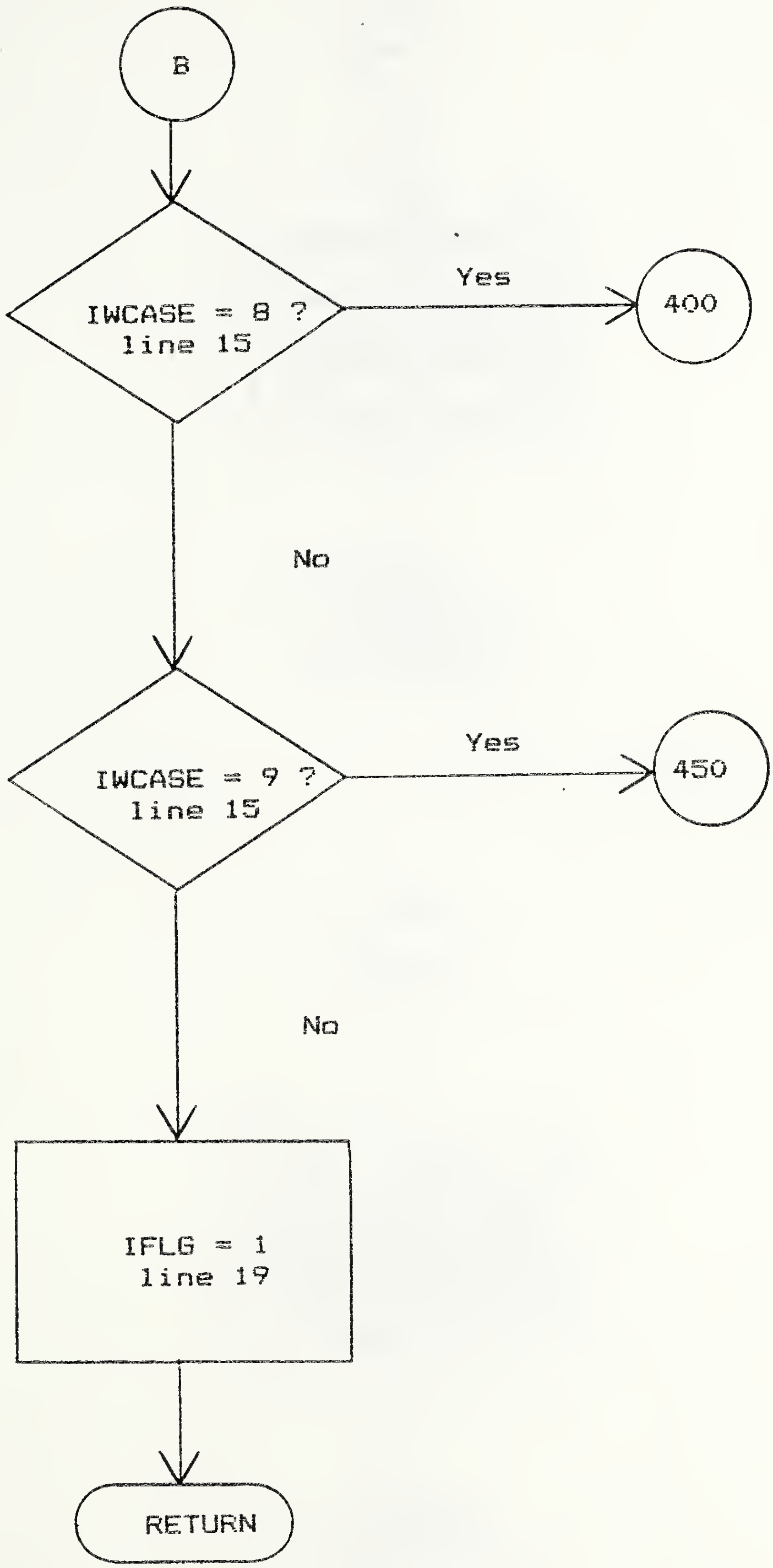
IWCASE	-	A case number that depends on the signs and magnitudes of W(1) and W(2). INTEGER
VRX, VRY	-	X and Y components of the vertices of the rectangle of interest. Starting in the upper left corner and proceeding counterclockwise the vertices are indexed: (1,1), (2,1), (2,2), (1,2). REAL arrays
EX, EY	-	X and Y components of the extreme values. There are only two in each case. REAL arrays
IFLG	-	Error flag. Set to 1 if W(1) = W(2) = 0, 0 otherwise. INTEGER

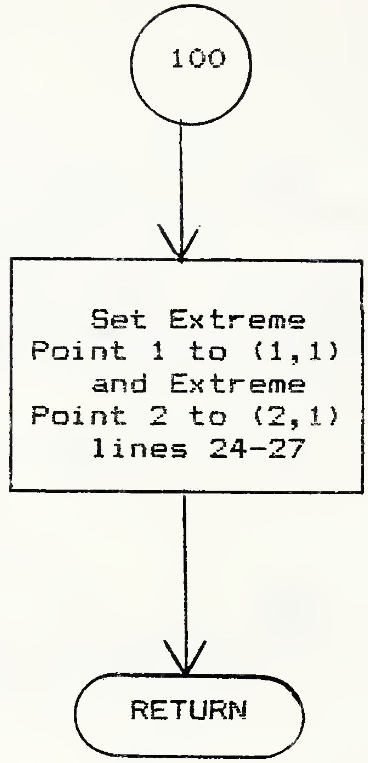
EXTREM does not call any subroutines.

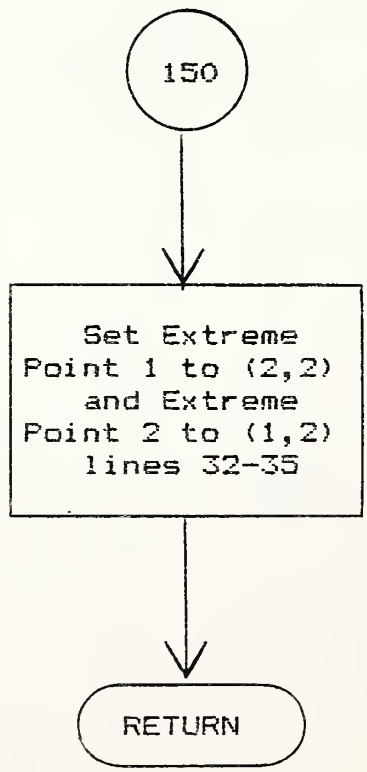
5.10.2 Flow Chart

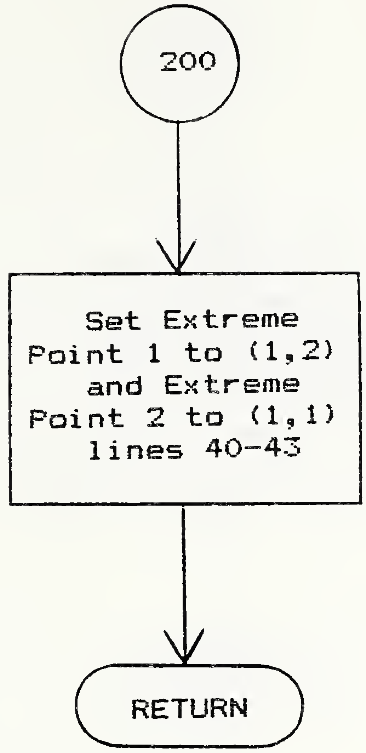


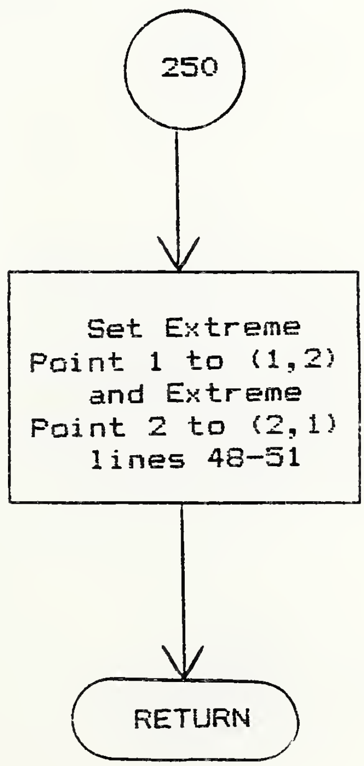

A
IWCASE = 4 ?
line 15
Yes
200
No
IWCASE = 5 ?
line 15
Yes
250
No
IWCASE = 6 ?
line 15
Yes
300
No
IWCASE = 7 ?
line 15
Yes
350
B

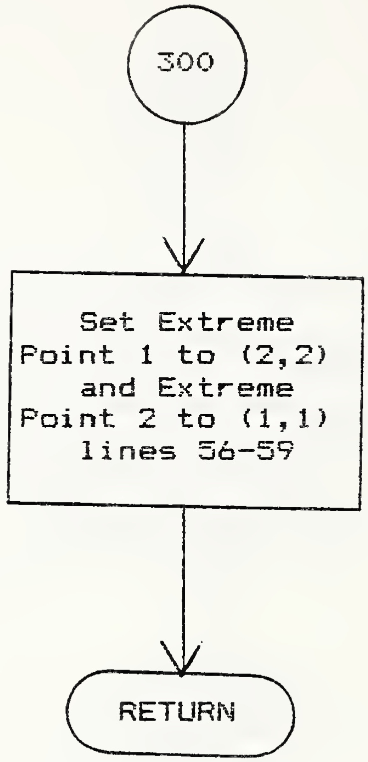

B
IWCASE = 8 ?
line 15
Yes
400
No
IWCASE = 9 ?
line 15
Yes
450
No
IFLG = 1
line 19
RETURN

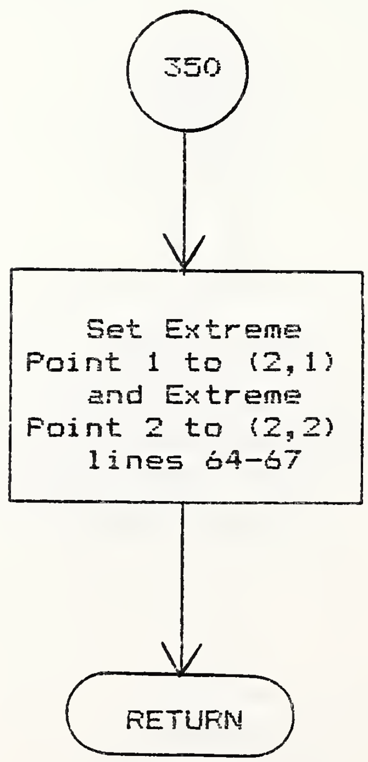

100
Set Extreme
Point 1 to (1,1)
and Extreme
Point 2 to (2,1)
lines 24-27
RETURN

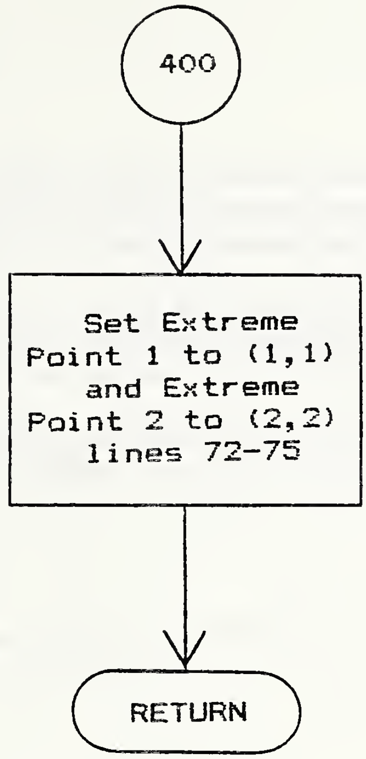

150
Set Extreme
Point 1 to (2,2)
and Extreme
Point 2 to (1,2)
lines 32-35
RETURN

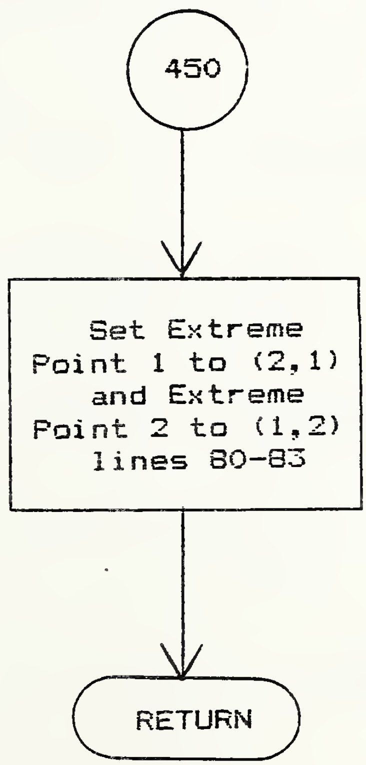

200
Set Extreme
Point 1 to (1,2)
and Extreme
Point 2 to (1,1)
lines 40-43
RETURN


250
Set Extreme
Point 1 to (1,2)
and Extreme
Point 2 to (2,1)
lines 48-51
RETURN


300
Set Extreme Point 1 to (2,2) and Extreme Point 2 to (1,1) lines 56-59
RETURN


350
Set Extreme Point 1 to (2,1) and Extreme Point 2 to (2,2) lines 64-67
RETURN


400
Set Extreme
Point 1 to (1,1)
and Extreme
Point 2 to (2,2)
lines 72-75
RETURN


450
Set Extreme
Point 1 to (2,1)
and Extreme
Point 2 to (1,2)
lines 80-83
RETURN

5.10.3 Listing

```
C**********************************************************************
        SUBROUTINE EXTREM(IWCASE, VRX, VRY, EX, EY, IFLG)
C**********************************************************************
C
C BASED UPON THE DIRECTION VECTOR W OF THE RAYS OF INTEREST AND
C THE VERTICES OF THE PICTURE RECTANGLE THIS SUBROUTINE RETURNS
C THE EXTREME POINTS SEEN BY THE RAYS IN THE ARRAYS EX, EY
C
C**********************************************************************
        REAL  VRX(2), VRY(2), EX(2), EY(2)
        IFLG = 0
C
C BRANCH ON IWCASE
C
        GO TO (50, 100, 150, 200, 250, 300, 350, 400, 450), IWCASE
C
C CASE 1: W(1) = W(2) = 0
C
50      IFLG = 1
        RETURN
C
C CASE 2: W(1) = 0, W(2) > 0
C
100     EX(1) = VRX(1)
        EY(1) = VRY(1)
        EX(2) = VRX(2)
        EY(2) = VRY(1)
        RETURN
C
C CASE 3: W(1) = 0, W(2) < 0
C
150     EX(1) = VRX(2)
        EY(1) = VRY(2)
        EX(2) = VRX(1)
        EY(2) = VRY(2)
        RETURN
C
C CASE 4: W(1) > 0, W(2) = 0
C
200     EX(1) = VRX(1)
        EY(1) = VRY(2)
        EX(2) = VRX(1)
        EY(2) = VRY(1)
        RETURN
C
C CASE 5: W(1) > 0, W(2) > 0
C
250     EX(1) = VRX(1)
        EY(1) = VRY(2)
        EX(2) = VRX(2)
```

```
      EY(2) = VRY(1)
      RETURN
C
C CASE 6:  W(1) > 0, W(2) < 0
C
300   EX(1) = VRX(2)
      EY(1) = VRY(2)
      EX(2) = VRX(1)
      EY(2) = VRY(1)
      RETURN
C
C CASE 7:  W(1) < 0, W(2) = 0
C
350   EX(1) = VRX(2)
      EY(1) = VRY(1)
      EX(2) = VRX(2)
      EY(2) = VRY(2)
      RETURN
C
C CASE 8:  W(1) < 0, W(2) > 0
C
400   EX(1) = VRX(1)
      EY(1) = VRY(1)
      EX(2) = VRX(2)
      EY(2) = VRY(2)
      RETURN
C
C CASE 9:  W(1) < 0, W(2) < 0
C
450   EX(1) = VRX(2)
      EY(1) = VRY(1)
      EX(2) = VRX(1)
      EY(2) = VRY(2)
      RETURN
      END
```

5.11 Subroutine GETROW

5.11.1 Summary

Let the world coordinate point (XO,YO,ZO) be projected to the screen point (SYO,SXO) by a parallel projection. Note that the screen coordinate system is an inverted coordinate system so that SYO represents the row of the projected point. Then the unit vector $\vec{K}$ sitting at (XO,YO,ZO) is directed in such a way that its coefficients represent an increment or decrement of a row number from the initial row set by SYO. The calling sequence is:

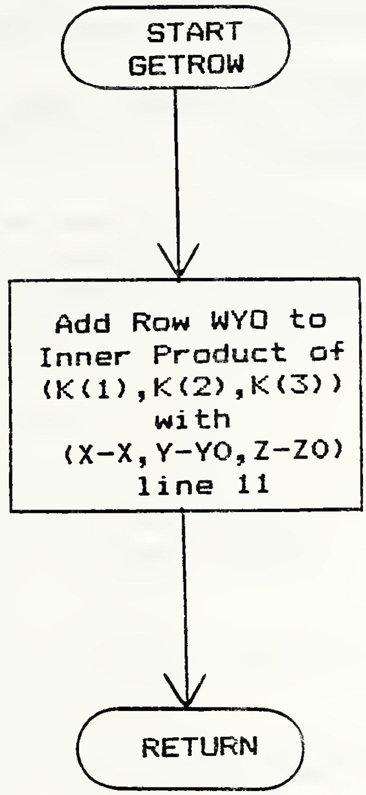
```
CALL GETROW (XO, YO, ZO, SYO, K, X, Y, Z, SY) .
```

The parameters passed are:

XO,YO,ZO	-	Components of the center of solid of interest. REAL
SYO	-	Projection row of XO,YO,ZO on the viewplane. REAL
K	-	Unit vector directed in such a way that coefficients index screen rows. REAL Array
X,Y,Z	-	Point for which row must be found. REAL
SY	-	Screen row for X,Y,Z. REAL

GETROW calls no subroutines.

5.11.2 Flow Chart



5.11.3 Listing

```
C**********************************************************************
      SUBROUTINE GETROW(XO,YO,ZO,SYO,K,X,Y,Z,SY)
C**********************************************************************
C
C ASSUME THAT AT THE VECTOR (XO,YO,ZO) THE UNIT VECTOR IS
C DIRECTED IN SUCH A MANNER THAT ITS COEFFICIENT REPRESENTS
C A ROW NUMBER OF THE MONITOR.
C
C**********************************************************************
      REAL XO,YO,ZO,SYO,K(3),X,Y,Z,SY
      SY = SYO + (X-XO)*K(1) + (Y-YO)*K(2) + (Z-ZO)*K(3)
      RETURN
      END
```

5.12 Subroutine GETZ

5.12.1 Summary

Given a point (X,Y) on the world coordinate Z = 0 plane, this subroutine returns the Z value at (X,Z) on a directed line row indexed by V. The calling sequence for this subroutine is:

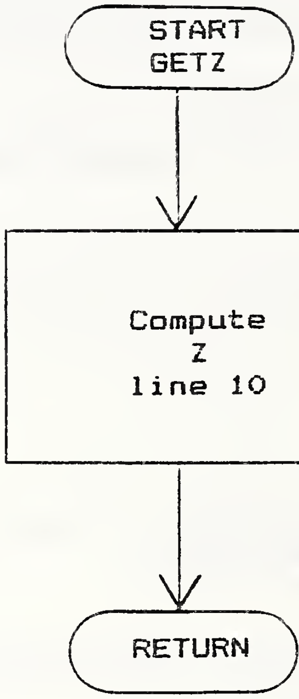
```
CALL GETZ (X, Y, V, K, Z) .
```

The parameters passed are:

X,Y	–	Components of the Z = 0 plane point. REAL
V	–	Row index specified. REAL
K	–	Vector used to index rows on the viewplane. REAL Array
Z	–	Height of ray above (X,Y). REAL

GETZ does not call any subroutines.

5.12.2 Flow Chart



5.12.3 Listing

```
C*********************************************************************
       SUBROUTINE GETZ(X,Y,V,K,Z)
C*********************************************************************
C
C GIVEN (X,Y) THIS SUBROUTINE RETURNS THE Z-VALUE AT (X,Y) ALONG
C THE RAY INDEXED BY V
C
C*********************************************************************
       REAL X, Y, V, K(3), Z
       Z = (1./K(3)) * (V - X*K(1) - Y*K(2))
       RETURN
       END
```

5.13 Subroutine GETR

5.13.1 Summary

Given (X,Y,Z) in the world coordinate system, this subroutine returns the multiple of the unit vector $\vec{W}$ pointing along the ray that intercepts the (X,Y,Z) point. The calling sequence for this subroutine is:

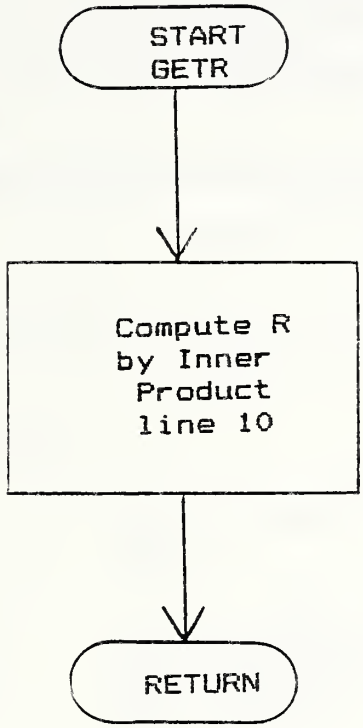
CALL GETR (X, Y, Z, W, R) .

The parameters passed are:

X,Y,Z	–	World coordinate point. REAL
W	–	Unit vector pointing along rays. REAL Array
R	–	Multiple of W-vector. REAL

GETR does not call any subroutines.

5.13.2 Flow Chart



5.13.3 Listing

```
C*********************************************************************
      SUBROUTINE GETR(X,Y,Z,W,R)
C*********************************************************************
C
C GIVEN (X,Y,Z) RETURN THE MULTIPLE OF THE W VECTOR RAY THAT
C INTERCEPTS THE POINT
C
C*********************************************************************
      REAL X, Y, Z, W(3), R
      R = X*W(1) + Y*W(2) + Z*W(3)
      RETURN
      END
```

5.14 Subroutine GETV

5.14.1 Summary

Given a point (X,Y,Z) in the world coordinate system, this subroutine returns the multiple of the K-vector that indexes the ray that intercepts (X,Y,Z). The calling sequence for this subroutine is:

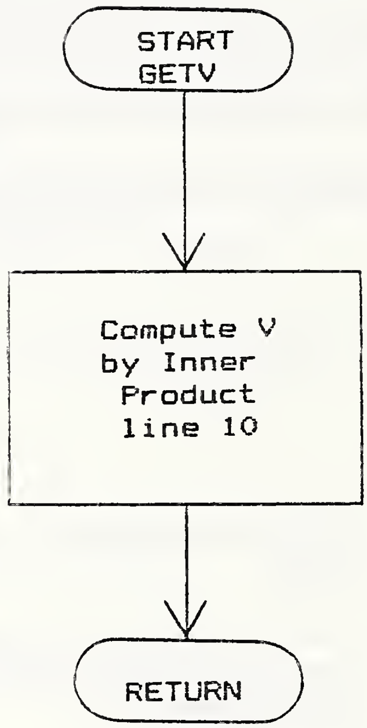
```
CALL GETV (X, Y, Z, K, V) .
```

The parameters passed are:

X,Y,Z	–	World coordinate point. REAL
K	–	Vector used to index rays in a vertical column of the viewplane. REAL Array
V	–	Multiple of K that indexes the vector. REAL

GETV does not call any subroutines.

5.14.2 Flow Chart



5.14.3 Listing

```
C*******************************************************************
      SUBROUTINE GETV(X,Y,Z,K,V)
C*******************************************************************
C
C GIVEN (X,Y,Z) RETURN THE MULTIPLE OF THE K VECTOR OF THE RAY
C THAT INTERCEPTS THE POINT
C
C*******************************************************************
      REAL X, Y, Z, K(3), V
      V = X*K(1) + Y*K(2) + Z*K(3)
      RETURN
      END
```

5.15 Subroutine GETH

5.15.1 Summary

Given a point (X,Y,Z) in the world coordinate system, this subroutine returns the multiple of the $\vec{K} \times \vec{W}$ vector for the ray that intercepts the point. In effect this selects the column or plane of rays that intersects (X,Y,Z). The calling sequence for this subroutine is:

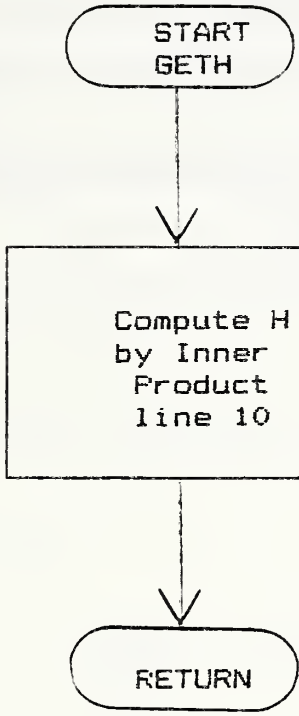
```
CALL GETH (X, Y, Z, KXW, H) .
```

The parameters passed are:

X,Y,Z	–	Components of the world coordinate system point. REAL
KXW	–	Vector orthogonal to the K vector and lying in the viewplane. REAL Array
H	–	Multiple of KXW. REAL

GETH does not call any subroutines.

5.15.2 Flow Chart



5.15.3 Listing

```
C*********************************************************************
      SUBROUTINE GETH(X,Y,Z,KXW,H)
C*********************************************************************
C
C GIVEN (X,Y,Z) RETURN THE MULTIPLE OF THE KXW VECTOR OF THE RAY
C THAT INTERCEPTS THE POINT
C
C*********************************************************************
      REAL X, Y, Z, KXW(3), H
      H = X*KXW(1) + Y*KXW(2)
      RETURN
      END
```

5.16 Subroutine GETXYZ

5.16.1 Summary

Let (SYO,SXO) be the orthogonal projection screen coordinates of the point (XO,YO,ZO) in the world coordinate frame of reference. Let (SY,SX) be a given screen coordinate. This subroutine transforms the screen point (SY,SX) into its associated world coordinate system point (X,Y,Z). The calling sequence for this subroutine is:

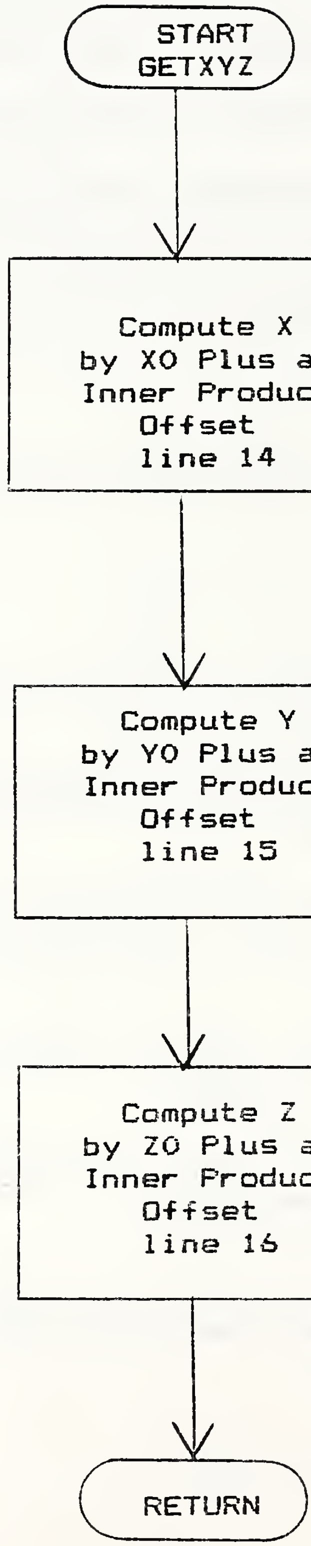
```
CALL GETXYZ (XO, YO, ZO, SYO, SXO, SY, SX, K, KXW, X, Y, Z) .
```

The parameters passed are:

XO,YO,ZO	-	Center of solid of interest. REAL
SYO,SXO	-	Screen coordinates of the projection of XO,YO,ZO. REAL
SY,SX	-	Screen coordinates of the selected screen point. REAL
K	-	Vector used to select screen row. REAL Array
KXW	-	Vector used to select screen column. REAL Array
X,Y,Z	-	World coordinate point associated with SY,SX. REAL

GETXYZ does not call any subroutines.

5.16.2 Flow Chart



5.16.3 Listing

```
C*********************************************************************
      SUBROUTINE GETXYZ(XO,YO,ZO,SYO,SXO,SY,SX,K,KXW,X,Y,Z)
C*********************************************************************
C
C LET (SYO,SXO) BE THE ORTHOGONAL PROJECTION SCREEN COORDINATES
C OF THE POINT (XO,YO,ZO) IN THE WORLD COORDINATE FRAME.  LET
C (SY,SX) BE A GIVEN SCREEN COORDINATE.  FIND THE ASSOCIATED
C WORLD COORDINATE POINT (X,Y,Z)
C
C*********************************************************************
      REAL XO, YO, ZO, SYO, SXO, SY, SX, K(3), KXW(3), X, Y, Z
      C1 = SY - SYO
      C2 = SX - SXO
      X = XO + C1*K(1) + C2*KXW(1)
      Y = YO + C1*K(2) + C2*KXW(2)
      Z = ZO + C1*K(3) + C2*KXW(3)
      RETURN
      END
```

5.17 Subroutine WCASE

5.17.1 Summary

This subroutine returns an index number from 1 to 9 in the variable IWCASE. This index points to each possible case combination of the first two components of the $\vec{W}$-vector that points along rays. The vector (W(1),W(2)) represents the direction vector of the projected directed line through $\vec{W}$ onto the Z=0 plane in the world coordinate space. The calling sequence for this subroutine is:

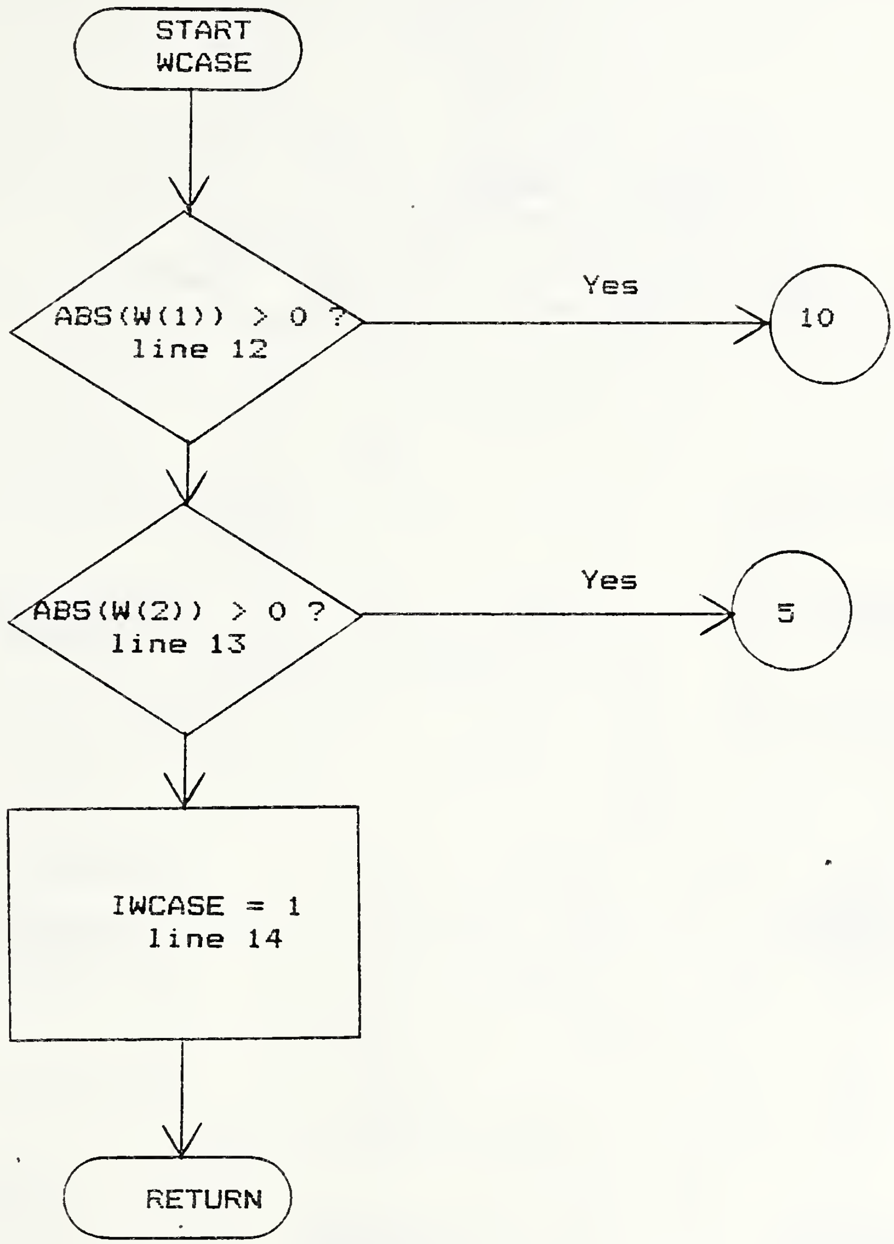
```
CALL WCASE (W, IWCASE) .
```

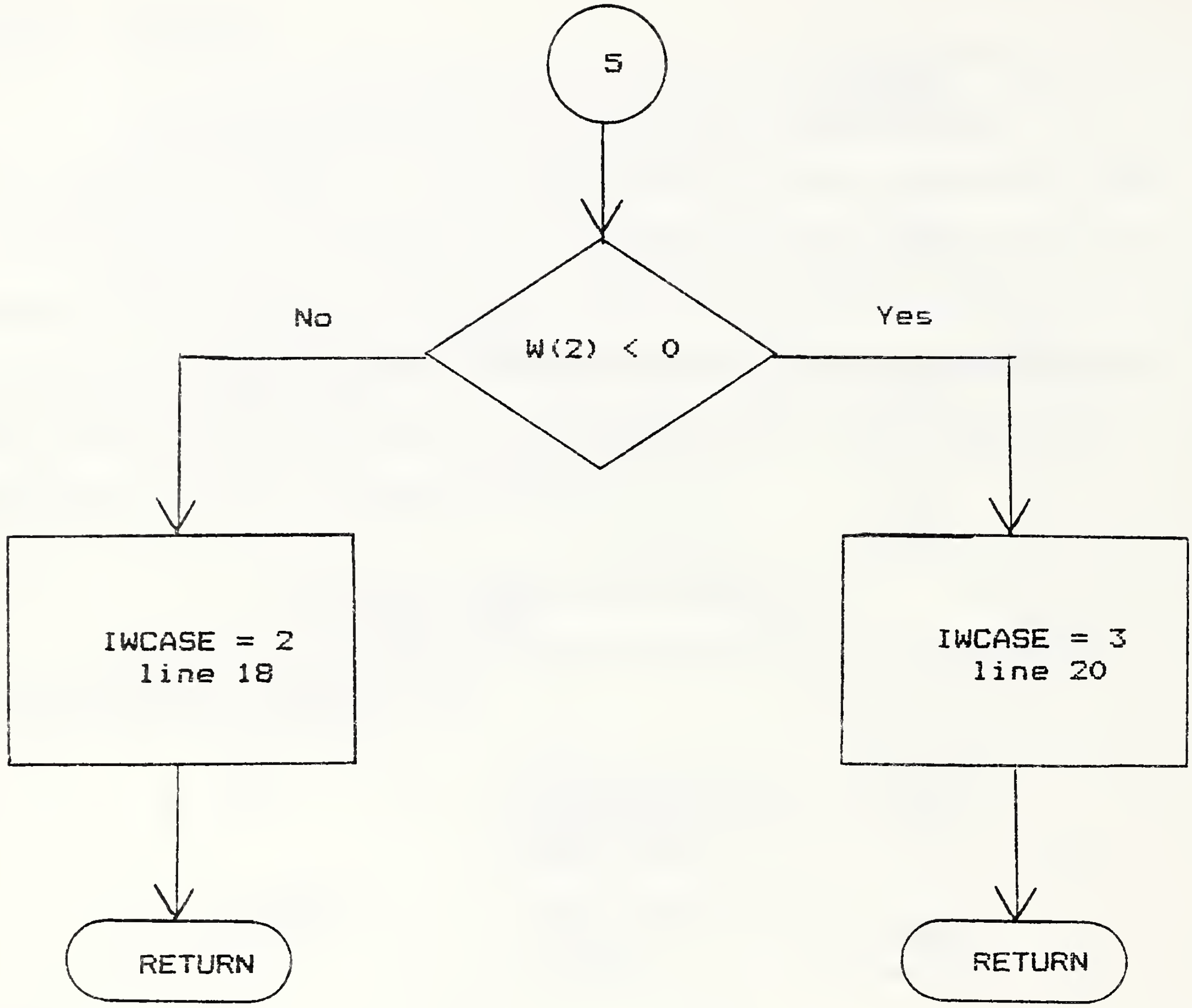
The parameters passed are:

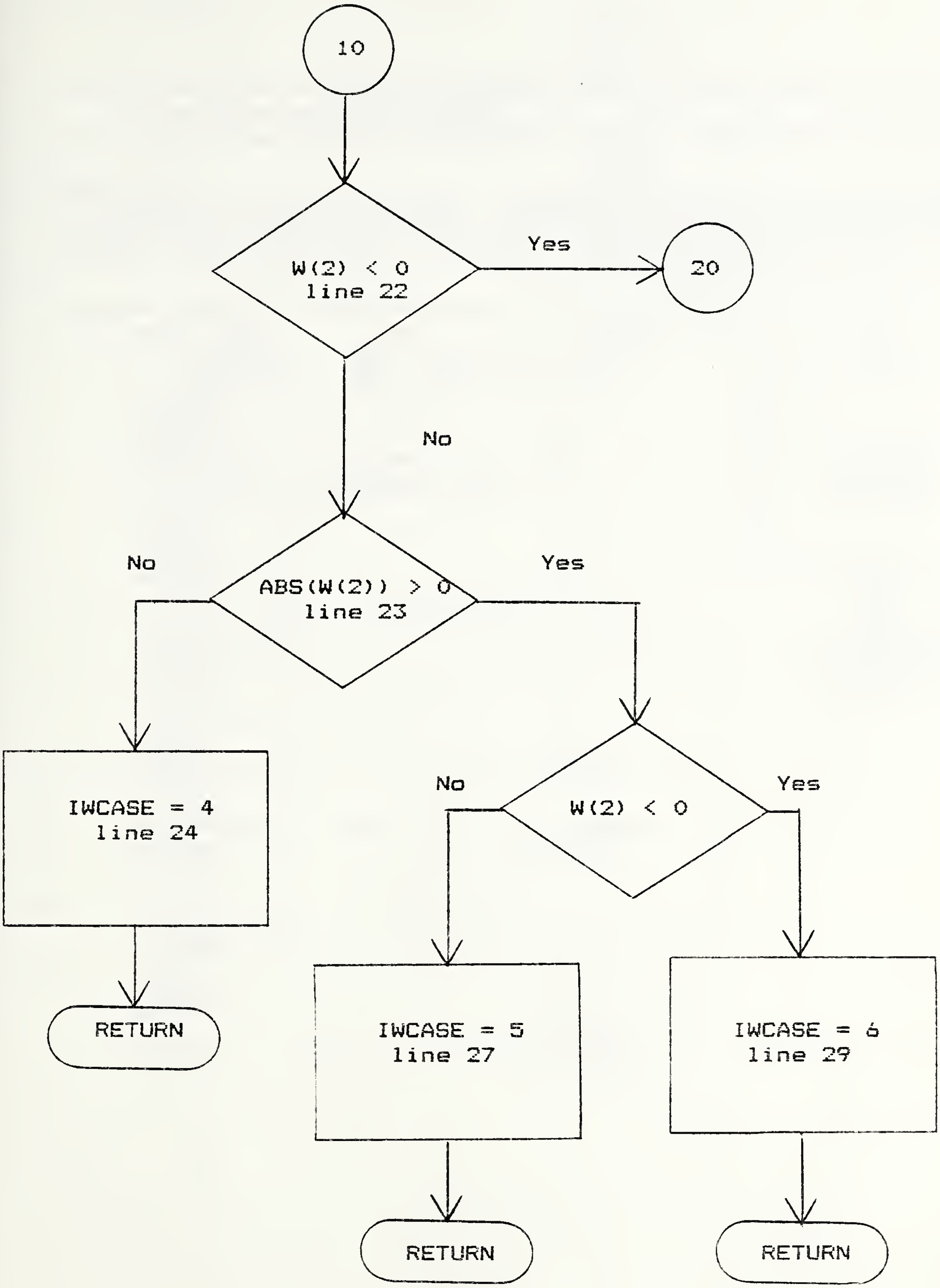
W	–	Direction vector pointing along rays. REAL Array
IWCASE	–	Case number from 1 to 9. INTEGER

WCASE does not call any subroutines.

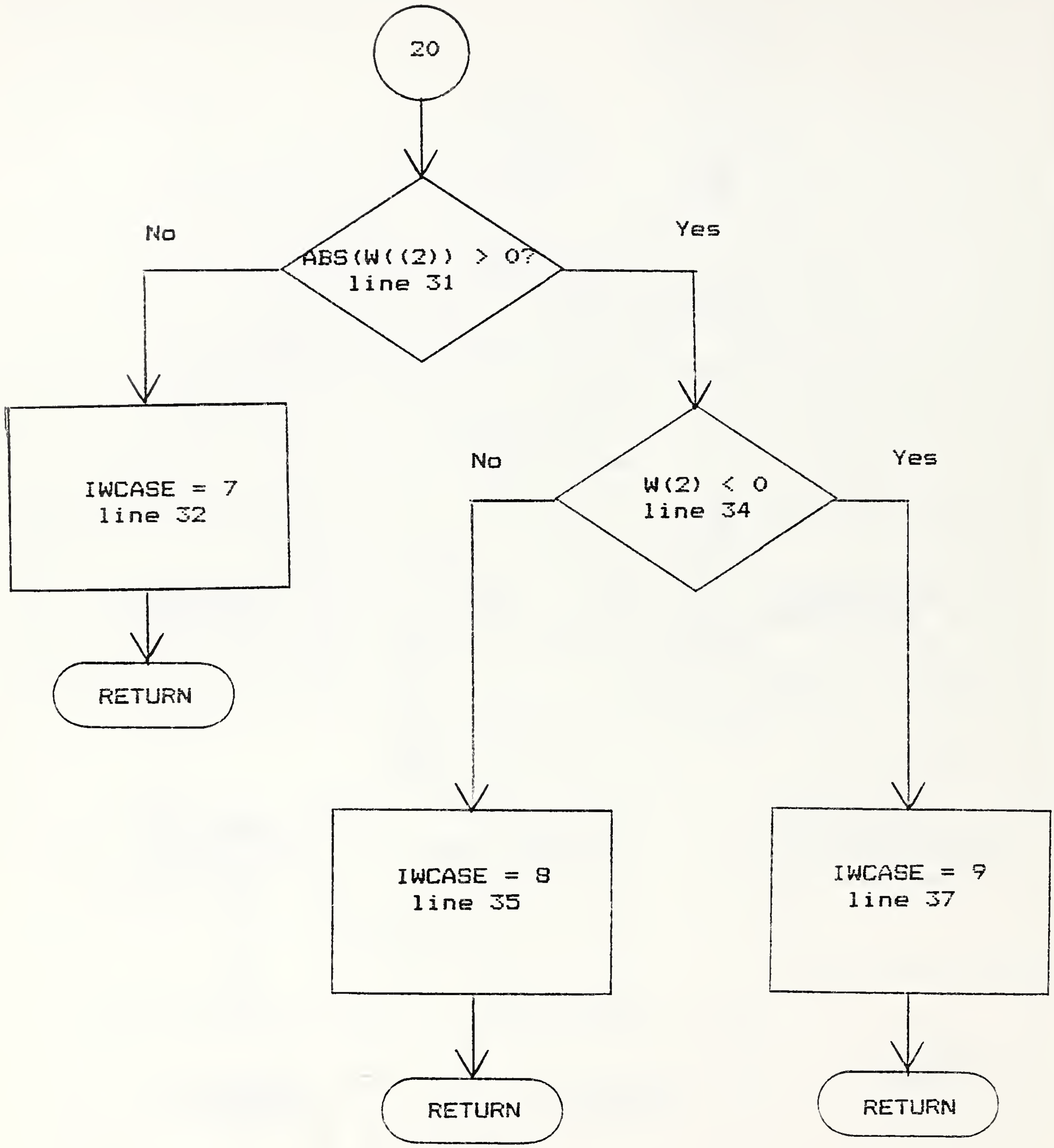
5.17.2 Flow Chart




5
W(2) < 0
No
Yes
IWCASE = 2
line 18
IWCASE = 3
line 20
RETURN
RETURN



10
W(2) < 0
line 22
Yes
20
No
No
ABS(W(2)) > 0
line 23
Yes
IWCASE = 4
line 24
RETURN
No
W(2) < 0
Yes
IWCASE = 5
line 27
IWCASE = 6
line 29
RETURN
RETURN


20
No
Yes
ABS(W((2)) > 0?
line 31
IWCASE = 7
line 32
RETURN
No
Yes
W(2) < 0
line 34
IWCASE = 8
line 35
RETURN
IWCASE = 9
line 37
RETURN

5.17.3 Listing

```
C*********************************************************************
      SUBROUTINE WCASE(W,IWCASE)
C*********************************************************************
C
C FUNCTION:
C  TO RETURN AN INDEX, IWCASE, THAT POINTS TO EACH POSSIBLE CASE
C COMBINATION OF THE FIRST TWO COMPONENTS OF THE W-VECTOR WHICH
C POINTS ALONG THE RAYS
C
C*********************************************************************
      REAL W(3)
      IF (ABS(W(1)) .GE. 5.E-6) GO TO 10
      IF (ABS(W(2)) .GE. 5.E-6) GO TO 5
      IWCASE = 1
      RETURN
5     CONTINUE
      IF (W(2) .LT. 0.) GO TO 7
      IWCASE = 2
      RETURN
7     IWCASE = 3
      RETURN
10    IF (W(1) .LT. 0.) GO TO 20
      IF (ABS(W(2)) .GE. 5.E-6) GO TO 15
      IWCASE = 4
      RETURN
15    IF (W(2) .LT. 0.) GO TO 17
      IWCASE = 5
      RETURN
17    IWCASE = 6
      RETURN
20    IF (ABS(W(2)) .GE. 5.E-6) GO TO 25
      IWCASE = 7
      RETURN
25    IF (W(2) .LT. 0.) GO TO 27
      IWCASE = 8
      RETURN
27    IWCASE = 9
      RETURN
      END
```

5.18 Subroutine XYIN

5.18.1 Summary

As the projections of the illuminating rays or viewing trace lines on the plane Z=0, some of the lines intersect the rectangle of interest. In the case of the shadowgraph, this rectangle is the base of the entire picture. In the case of the solid projection, it is the user selected rectangle. Assume that some projected ray enters the rectangle at (X,Y). This subroutine returns the next entry point or flags that an extreme point has been met. The calling sequence for this subroutine is:

```
CALL XYIN (IWCASE, EX, EY, IXIN, IYIN, XIN, YIN, IFLG) .
```

The parameters passed through the calling sequence are:

ON INPUT -

IWCASE	-	The case index for $\vec{W}$. INTEGER
EX,EY	-	Two element arrays representing extreme points. REAL Arrays
IXIN,IYIN	-	On entry to the subroutine these represent the current entry point to the rectangle. INTEGER
XIN,YIN	-	Real values of IXIN,IYIN. REAL

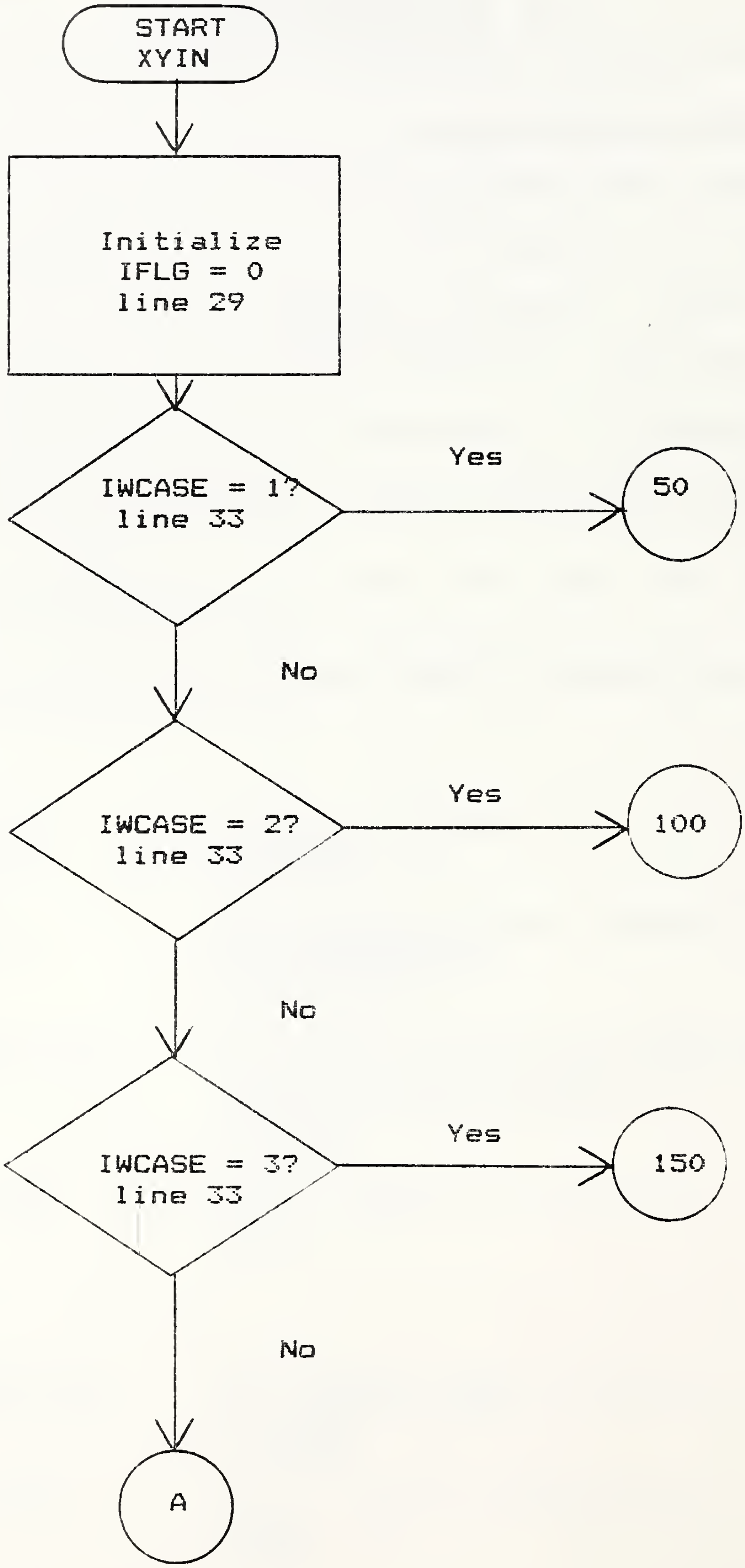
ON OUTPUT -

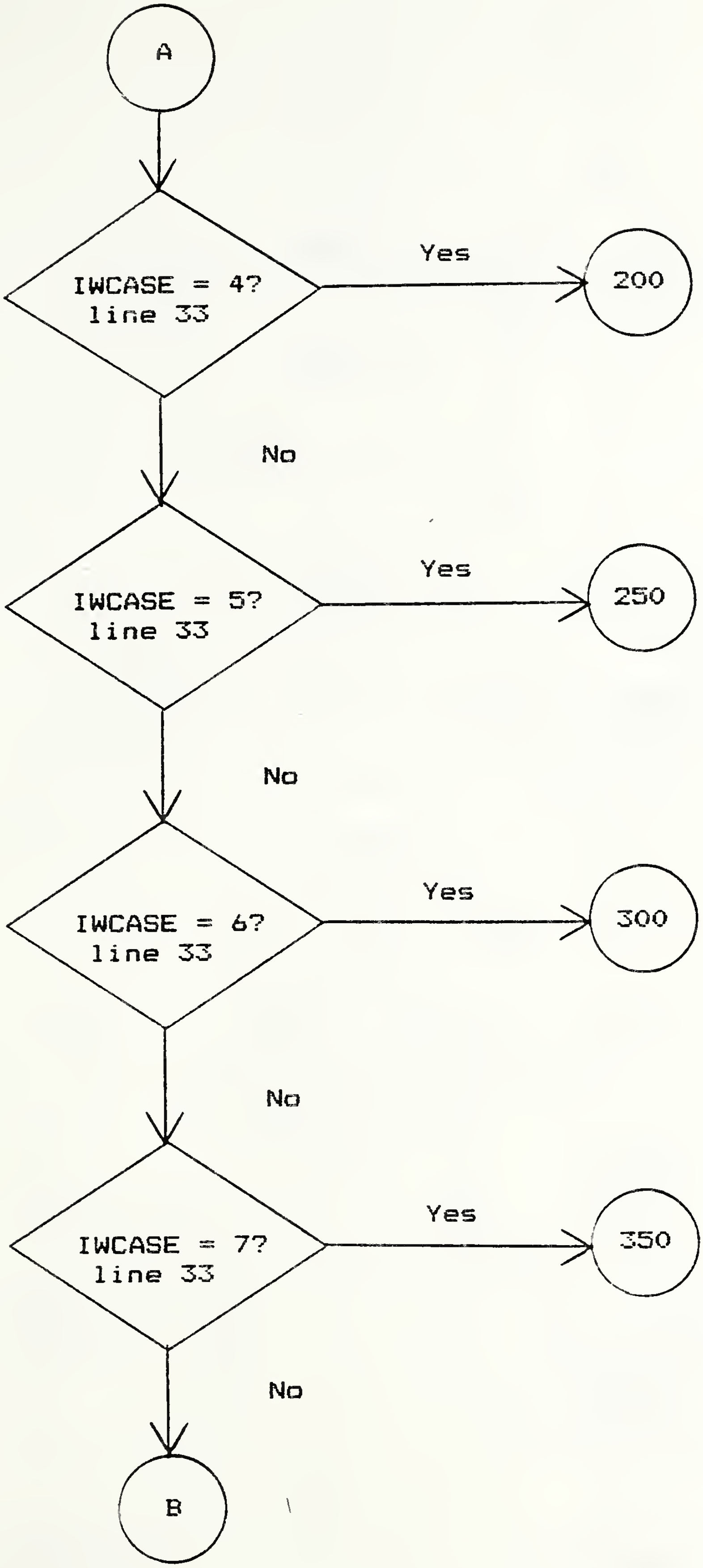
IXIN,IYIN	-	On output, these represent the next entry point. INTEGER
XIN,YIN	-	Real values of IXIN,IYIN. REAL

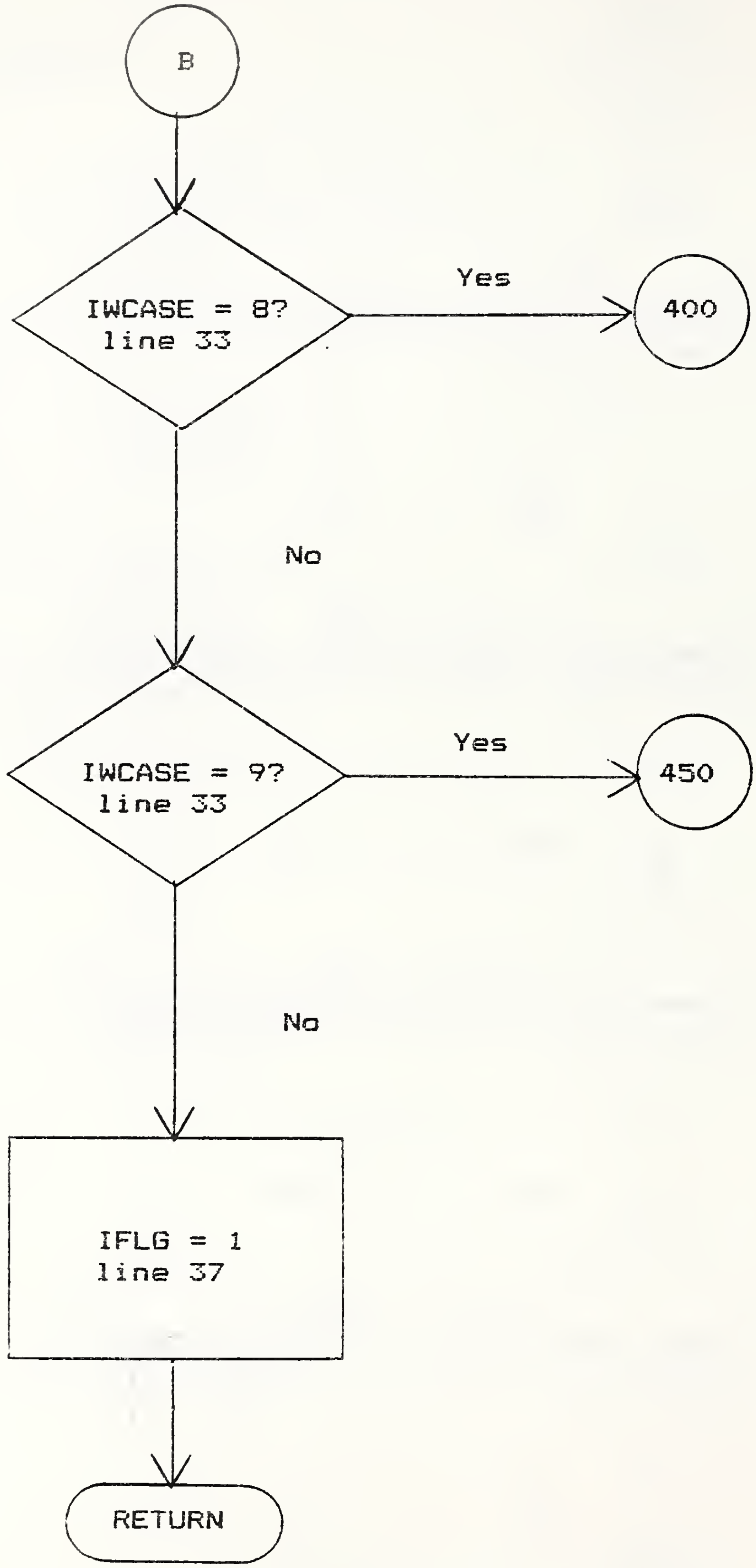
```
IFLG          -    = 0  if a new entry point
                        is returned.
                   = 1  if  W(1) = W(2) = 0.
                   = 2  if the extreme point
                        EX(2),EY(2) is met.
```

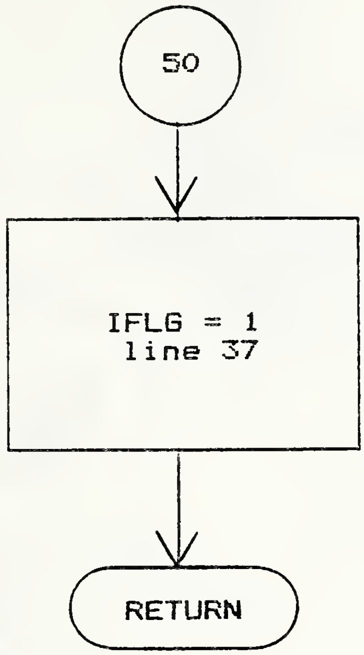
XYIN does not call any subroutines.

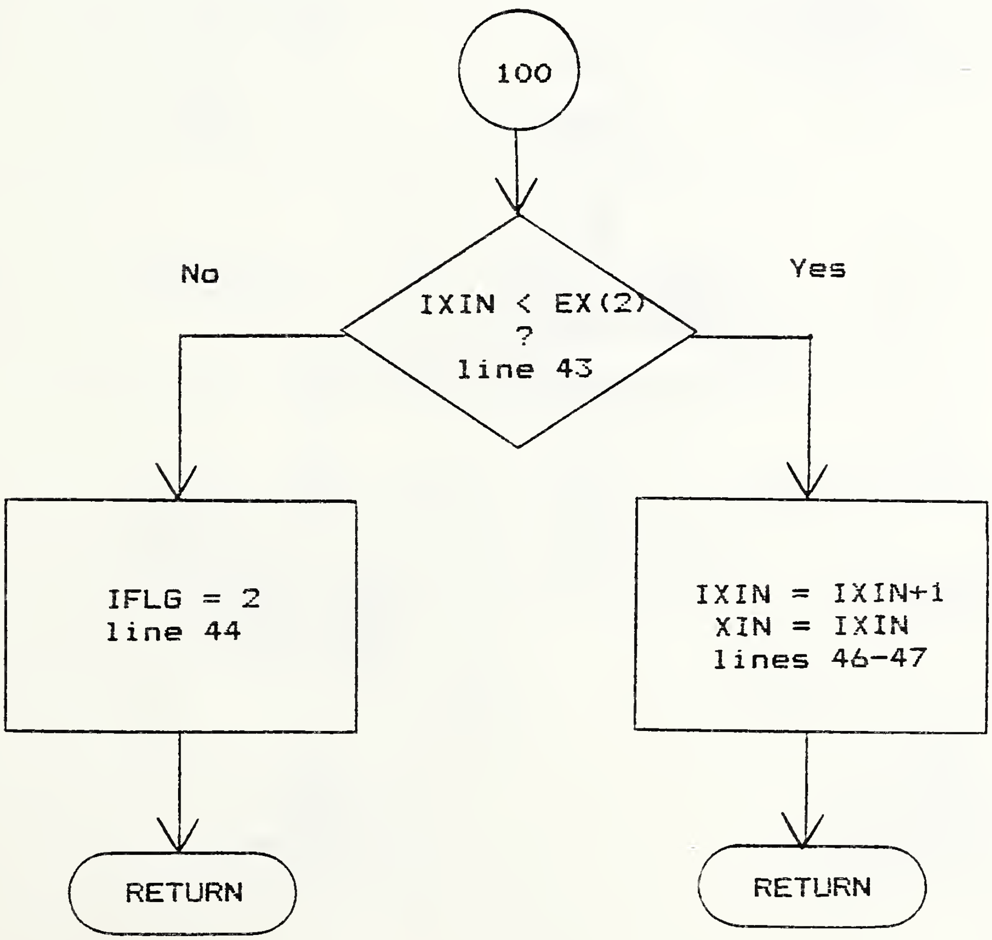
5.18.2 Flow Chart

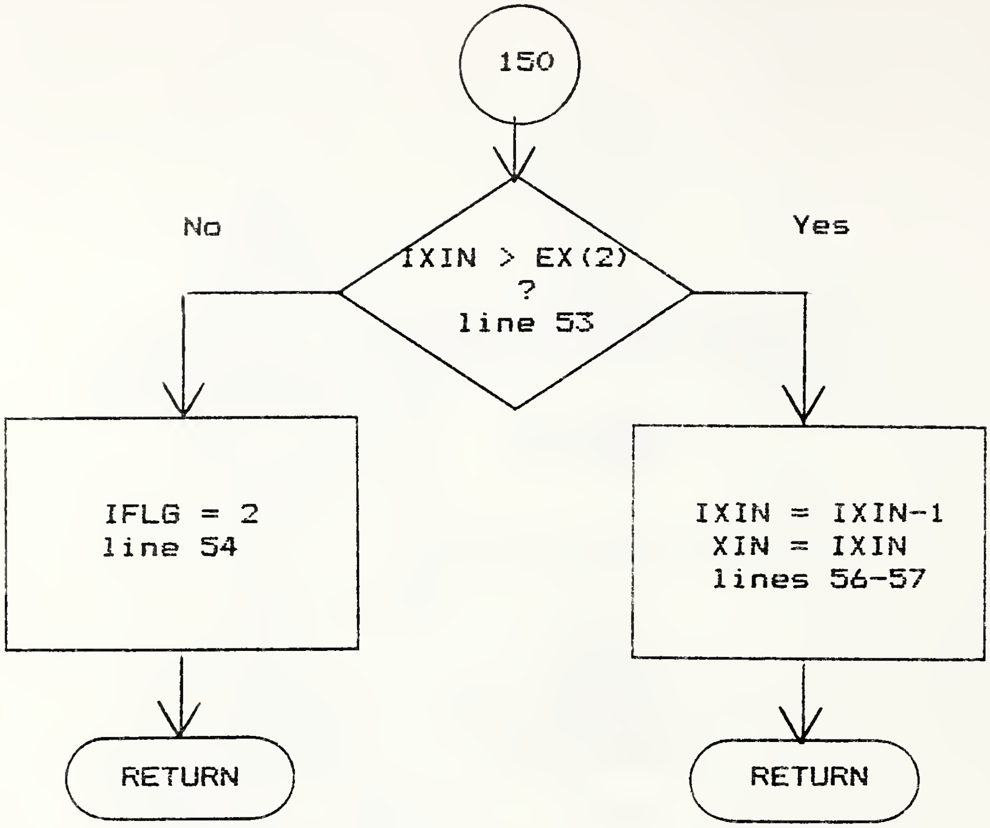


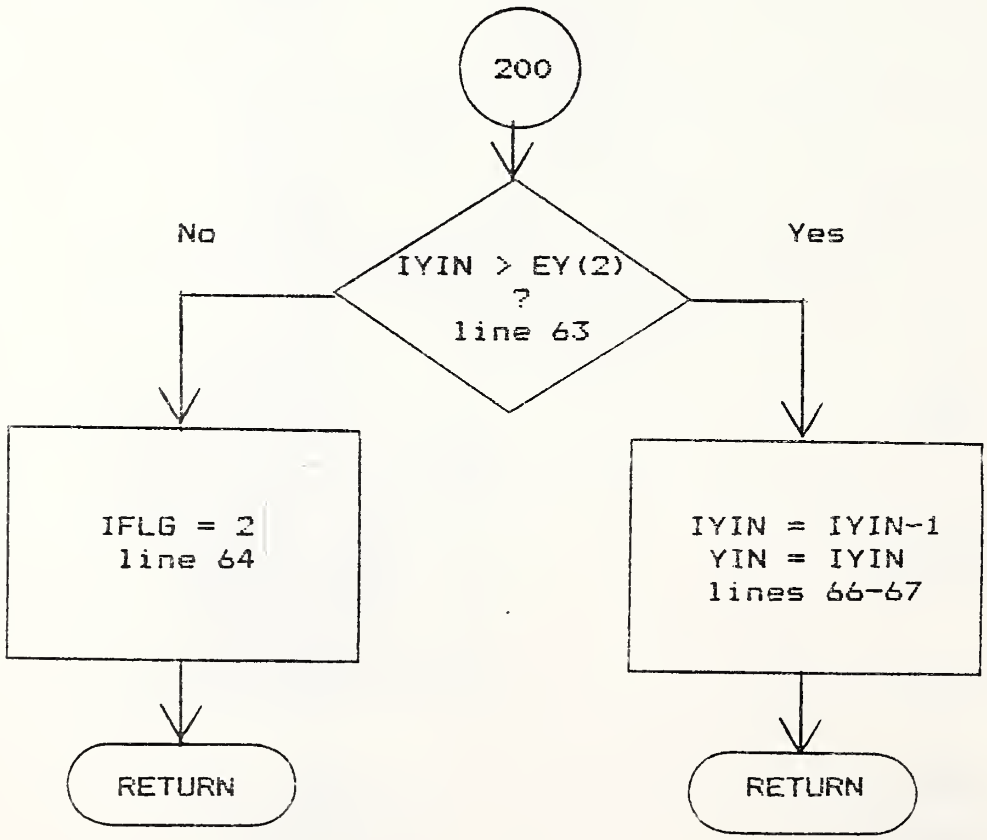

A
IWCASE = 4?
line 33
Yes
200
No
IWCASE = 5?
line 33
Yes
250
No
IWCASE = 6?
line 33
Yes
300
No
IWCASE = 7?
line 33
Yes
350
No
B

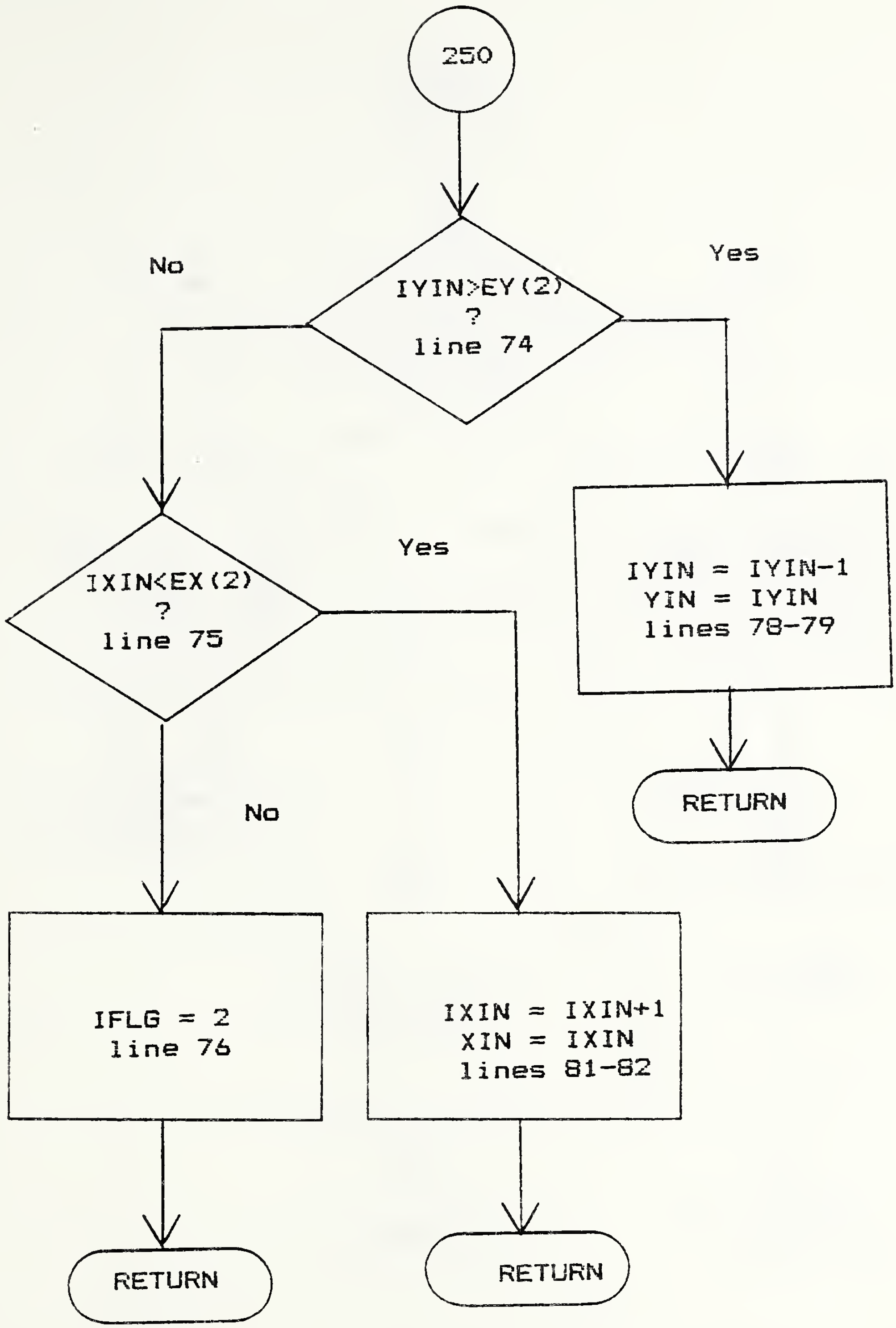

B
IWCASE = 8?
line 33
Yes
400
No
IWCASE = 9?
line 33
Yes
450
No
IFLG = 1
line 37
RETURN

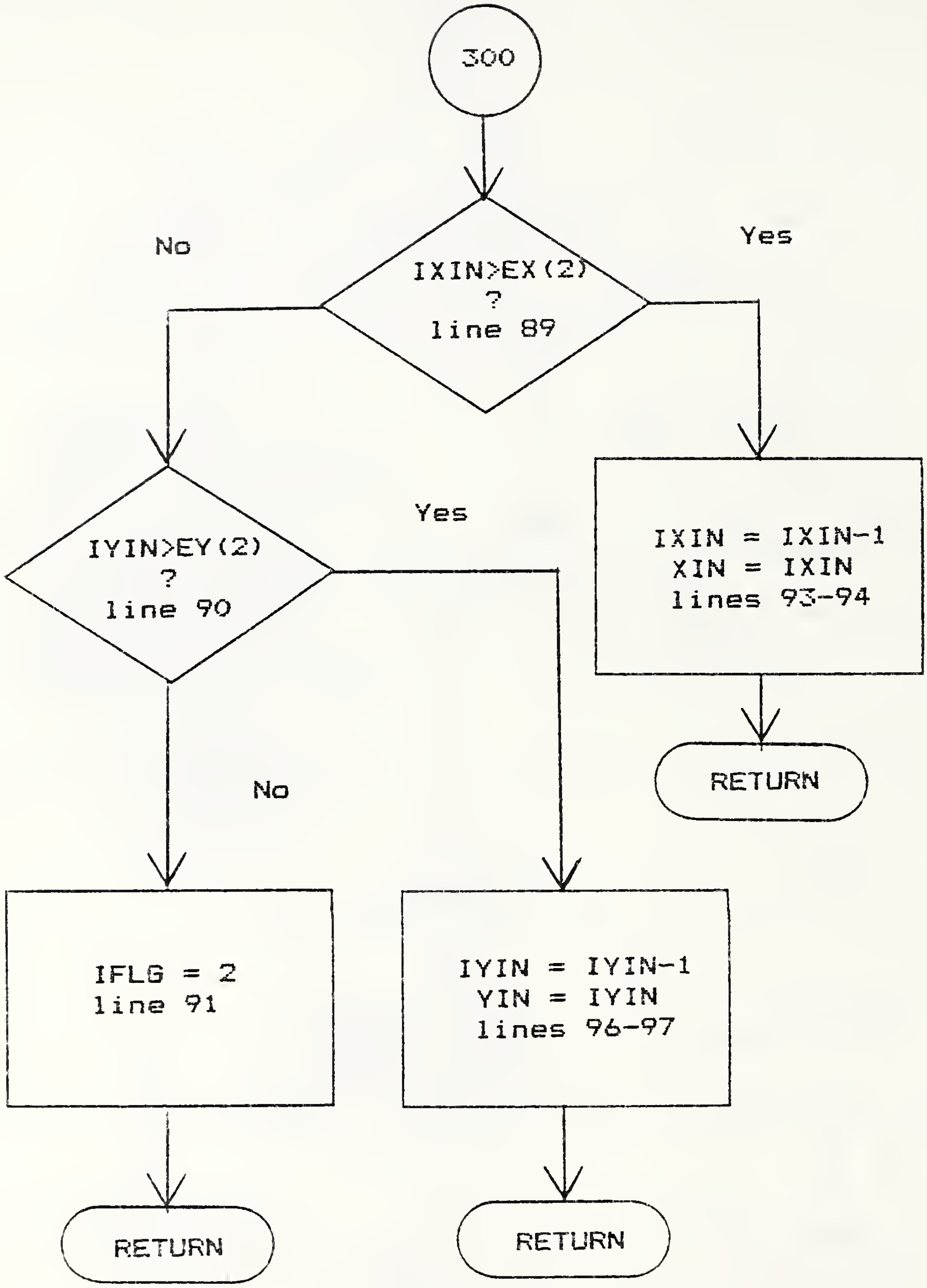

50
IFLG = 1
line 37
RETURN

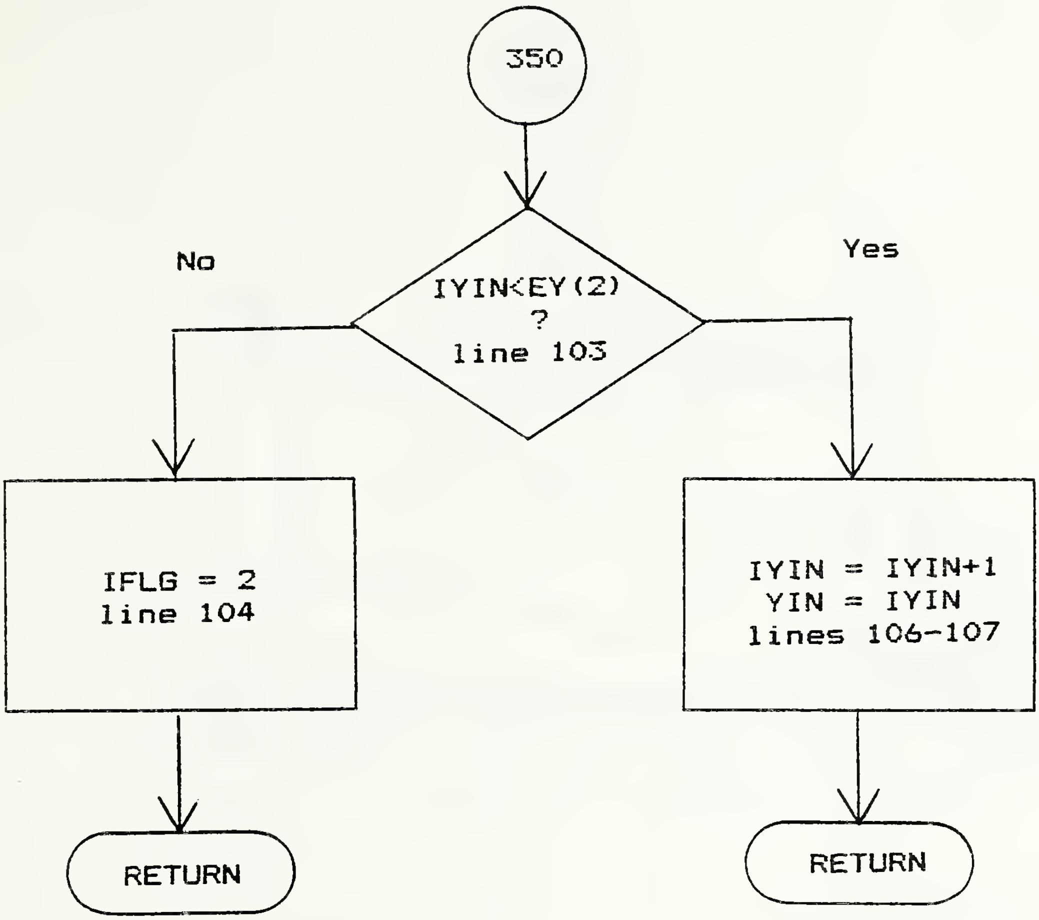

100
No
Yes
IXIN < EX(2)
?
line 43
IFLG = 2
line 44
IXIN = IXIN+1
XIN = IXIN
lines 46-47
RETURN
RETURN

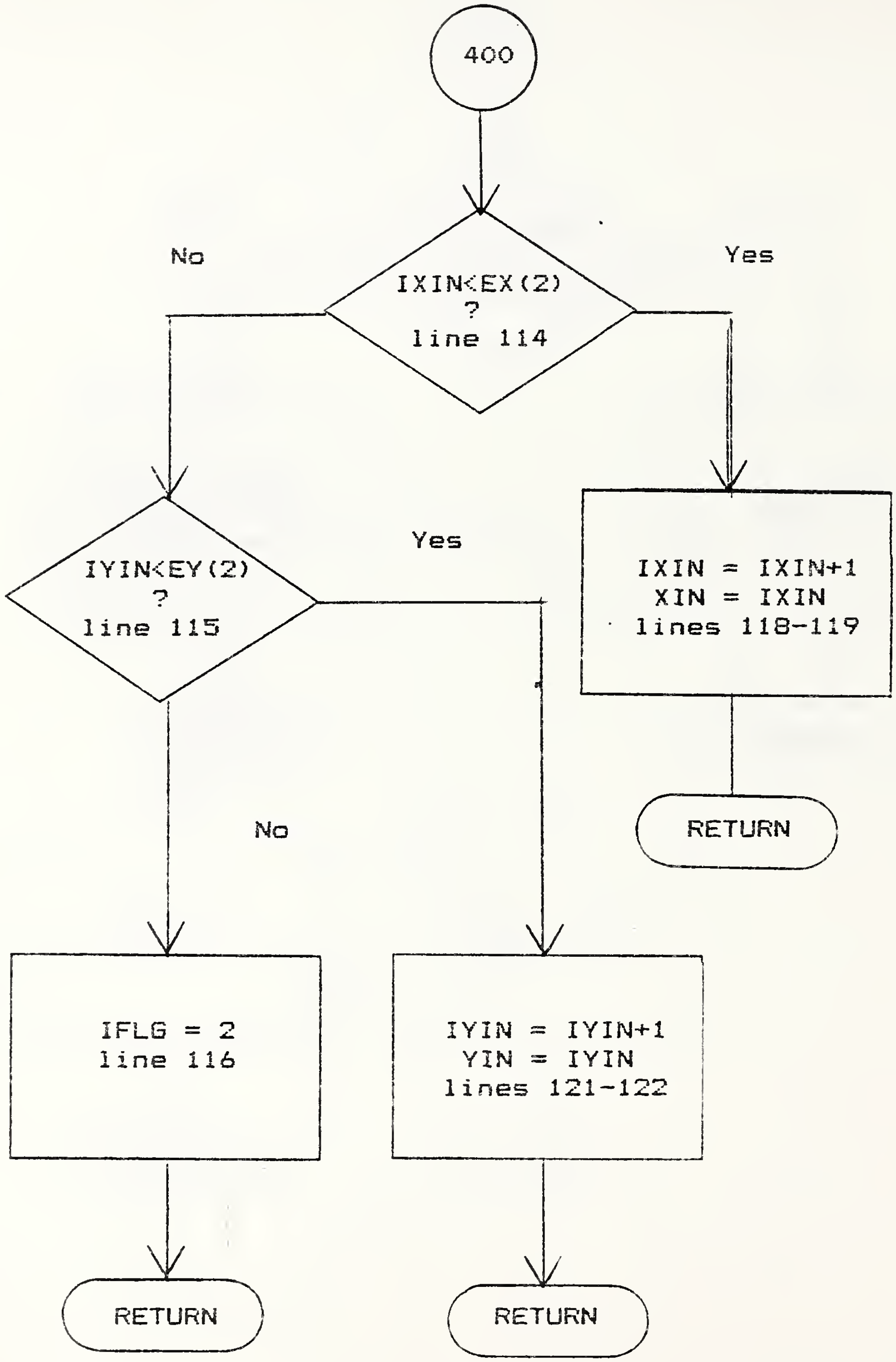

150
No
Yes
IXIN > EX(2)
?
line 53
IFLG = 2
line 54
IXIN = IXIN-1
XIN = IXIN
lines 56-57
RETURN
RETURN

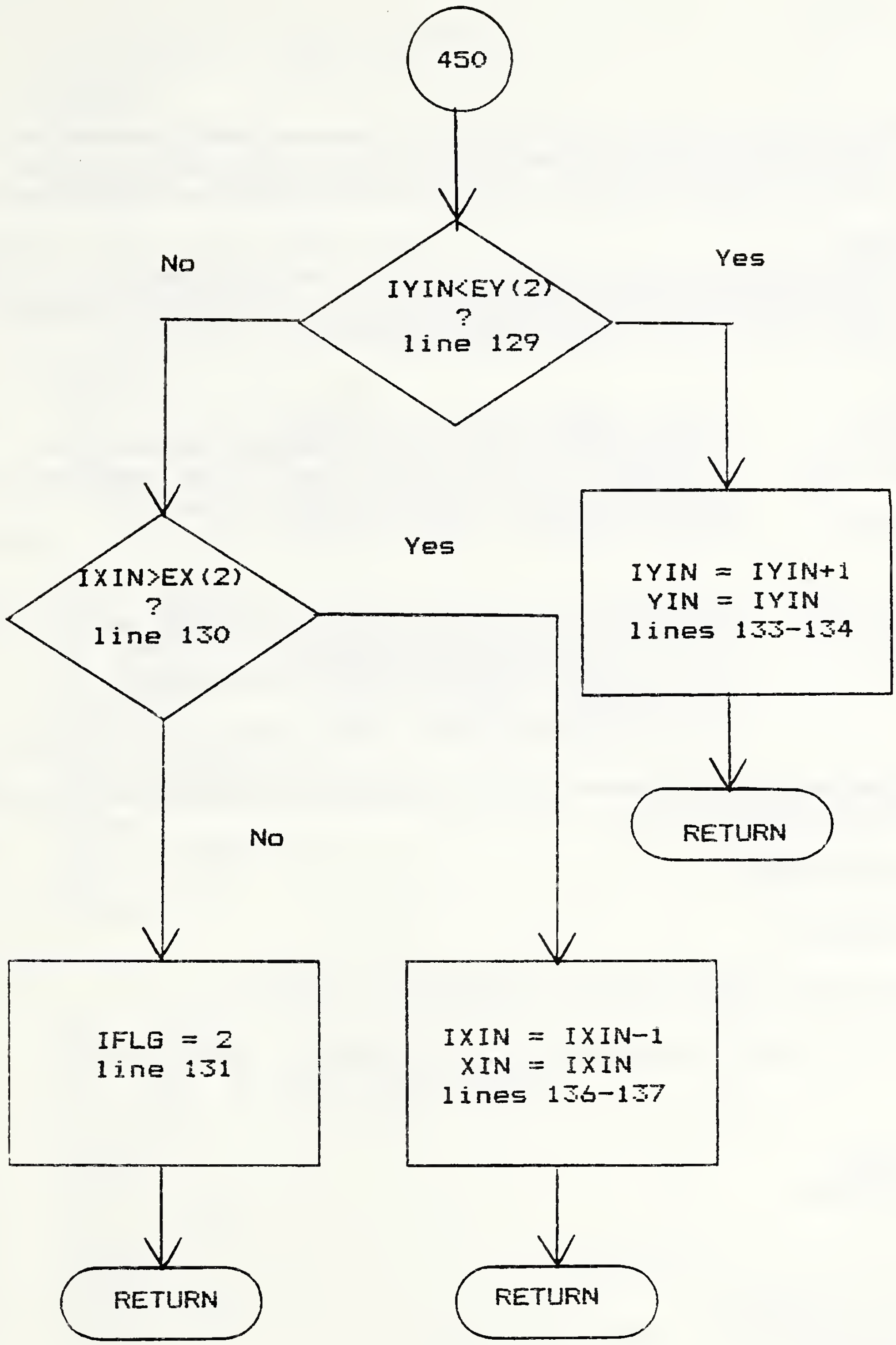

200
No
Yes
IYIN > EY(2)
?
line 63
IFLG = 2
line 64
IYIN = IYIN-1
YIN = IYIN
lines 66-67
RETURN
RETURN


250
IYIN>EY(2)
?
line 74
No
Yes
IYIN = IYIN-1
YIN = IYIN
lines 78-79
RETURN
IXIN<EX(2)
?
line 75
Yes
No
IFLG = 2
line 76
IXIN = IXIN+1
XIN = IXIN
lines 81-82
RETURN
RETURN


300
No
Yes
IXIN>EX(2)
?
line 89
Yes
IYIN>EY(2)
?
line 90
IXIN = IXIN-1
XIN = IXIN
lines 93-94
RETURN
No
IFLG = 2
line 91
IYIN = IYIN-1
YIN = IYIN
lines 96-97
RETURN
RETURN


350
No
Yes
IYIN<EY(2)
?
line 103
IFLG = 2
line 104
IYIN = IYIN+1
YIN = IYIN
lines 106-107
RETURN
RETURN


400
No
Yes
IXIN<EX(2)
?
line 114
Yes
IYIN<KEY(2)
?
line 115
IXIN = IXIN+1
XIN = IXIN
lines 118-119
No
RETURN
IFLG = 2
line 116
IYIN = IYIN+1
YIN = IYIN
lines 121-122
RETURN
RETURN


450
IYIN<EY(2)
?
line 129
No
Yes
IYIN = IYIN+1
YIN = IYIN
lines 133-134
RETURN
IXIN>EX(2)
?
line 130
Yes
No
IFLG = 2
line 131
RETURN
IXIN = IXIN-1
XIN = IXIN
lines 136-137
RETURN

5.18.3 Listing

```
C**********************************************************************
      SUBROUTINE XYIN(IWCASE,EX,EY,IXIN,IYIN,XIN,YIN,IFLG)
C**********************************************************************
C
C FUNCTION:
C  GIVEN THE CURRENT X,Y ENTRY POINT TO THE RECTANGLE OF INTEREST
C THIS SUBROUTINE RETURNS THE NEXT ENTRY POINT OR FLAGS THAT
C AN EXTREME POINT HAS BEEN ENCOUNTERED.
C
C INPUT:
C   IWCASE   -    CASE INDEX FOR THE VECTOR W
C   EX,EY    -    TWO ELEMENT ARRAYS OF EXTREME POINTS
C   IXIN,IYIN-    ON ENTRY TO THE SUBROUTINE THESE REPRESENT THE
C                 CURRENT ENTRY POINT TO THE RECTANGLE
C   XIN,YIN  -    REAL VALUES OF IXIN,IYIN
C
C OUTPUT:
C   IXIN,IYIN-    ON OUTPUT THESE REPRESENT THE NEXT ENTRY POINT
C   XIN,YIN  -    REAL VALUES OF IXIN,IYIN
C   IFLG     -    = 0 IF A NEW ENTRY POINT IS RETURNED
C                 = 1 IF W(1) = W(2) = 0
C                 = 2 IF THE EXTREME POINT EX(2),EY(2) IS MET
C
C**********************************************************************
      REAL EX(2), EY(2)
C
C INITIALIZE FLAG
C
      IFLG = 0
C
C BRANCH TO THE CASE FOR THE CURRENT W - VECTOR
C
      GO TO (50,100,150,200,250,300,350,400,450), IWCASE
C
C CASE 1: W(1) = W(2) = 0
C
50    IFLG = 1
      RETURN
C
C CASE 2: W(1) = 0, W(2) > 0
C
100   IEX = EX(2)
      IF (IXIN .LT. IEX) GO TO 110
      IFLG = 2
      RETURN
110   IXIN = IXIN + 1
      XIN = IXIN
      RETURN
C
C CASE 3: W(1) = 0, W(2) < 0
C
150   IEX = EX(2)
```

```
          IF (IXIN .GT.  IEX) GO TO 170
          IFLG = 2
          RETURN
  170     IXIN = IXIN - 1
          XIN = IXIN
          RETURN
C
C CASE 4: W(1) > 0, W(2) = 0
C
  200     IEY = EY(2)
          IF (IYIN .GT.  IEY) GO TO 230
          IFLG = 2
          RETURN
  230     IYIN = IYIN - 1
          YIN = IYIN
          RETURN
C
C CASE 5: W(1) > 0, W(2) > 0
C
  250     IEX = EX(2)
          IEY = EY(2)
          IF (IYIN .GT.  IEY) GO TO 285
          IF (IXIN .LT.  IEX) GO TO 290
          IFLG = 2
          RETURN
  285     IYIN = IYIN - 1
          YIN = IYIN
          RETURN
  290     IXIN = IXIN + 1
          XIN = IXIN
          RETURN
C
C CASE 6: W(1) > 0, W(2) < 0
C
  300     IEX = EX(2)
          IEY = EY(2)
          IF (IXIN .GT.  IEX) GO TO 345
          IF (IYIN .GT.  IEY) GO TO 348
          IFLG = 2
          RETURN
  345     IXIN = IXIN - 1
          XIN = IXIN
          RETURN
  348     IYIN = IYIN - 1
          YIN = IYIN
          RETURN
C
C CASE 7: W(1) < 0, W(2) = 0
C
  350     IEY = EY(2)
          IF (IYIN .LT.  IEY) GO TO 360
          IFLG = 2
          RETURN
```

```
 360    IYIN = IYIN + 1
        YIN = IYIN
        RETURN
C
C CASE 8:  W(1) < 0,  W(2) > 0
C
 400    IEX = EX(2)
        IEY = EY(2)
        IF (IXIN .LT. IEX) GO TO 435
        IF (IYIN .LT. IEY) GO TO 440
        IFLG = 2
        RETURN
 435    IXIN = IXIN + 1
        XIN = IXIN
        RETURN
 440    IYIN = IYIN + 1
        YIN = IYIN
        RETURN
C
C CASE 9:  W(1) < 0,  W(2) < 0
C
 450    IEX = EX(2)
        IEY = EY(2)
        IF (IYIN .LT. IEY) GO TO 475
        IF (IXIN .GT. IEX) GO TO 480
        IFLG = 2
        RETURN
 475    IYIN = IYIN + 1
        YIN = IYIN
        RETURN
 480    IXIN = IXIN - 1
        XIN = IXIN
        RETURN
        END
```

5.19 Subroutine PUTFIL

5.19.1 Summary

This subroutine opens a new file in mass storage and transfers an image of 512 records by 512 bytes per record. The user interactively specifies the file name for the new file prior to the subroutine opening it. The calling sequence for the subroutine is:

```
CALL PUTFIL (FCB, BUFFER, TABLE, CHANLS) .
```

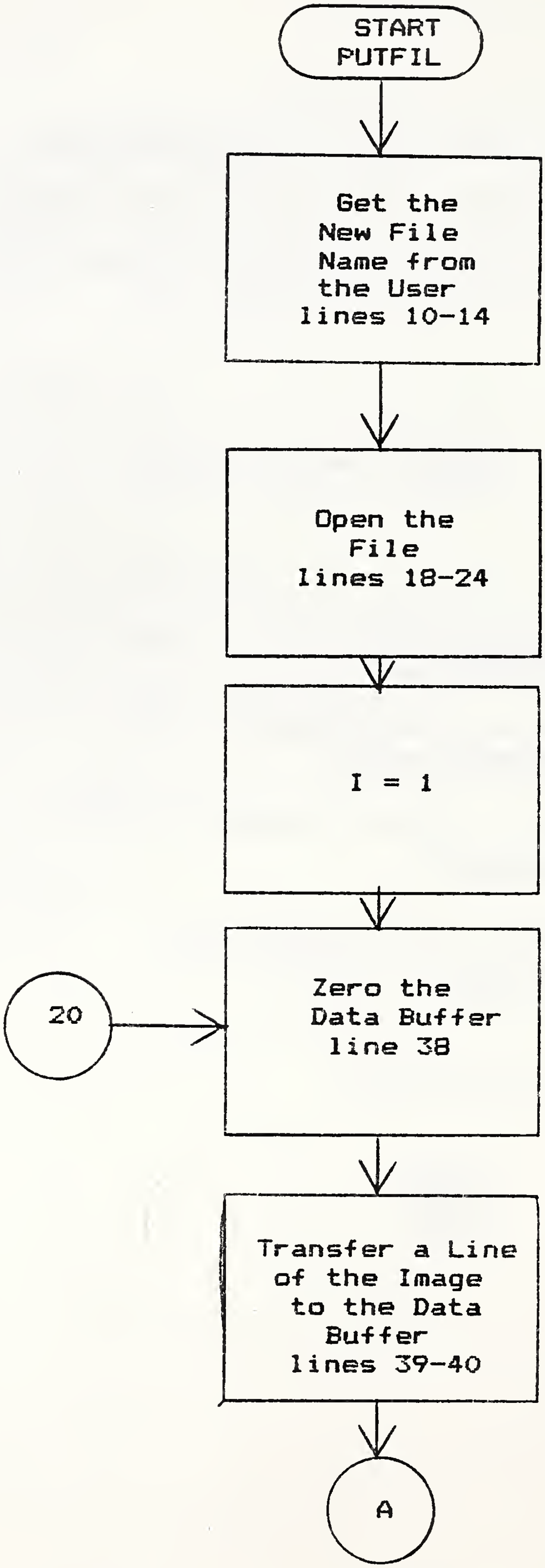
The parameters passed are:

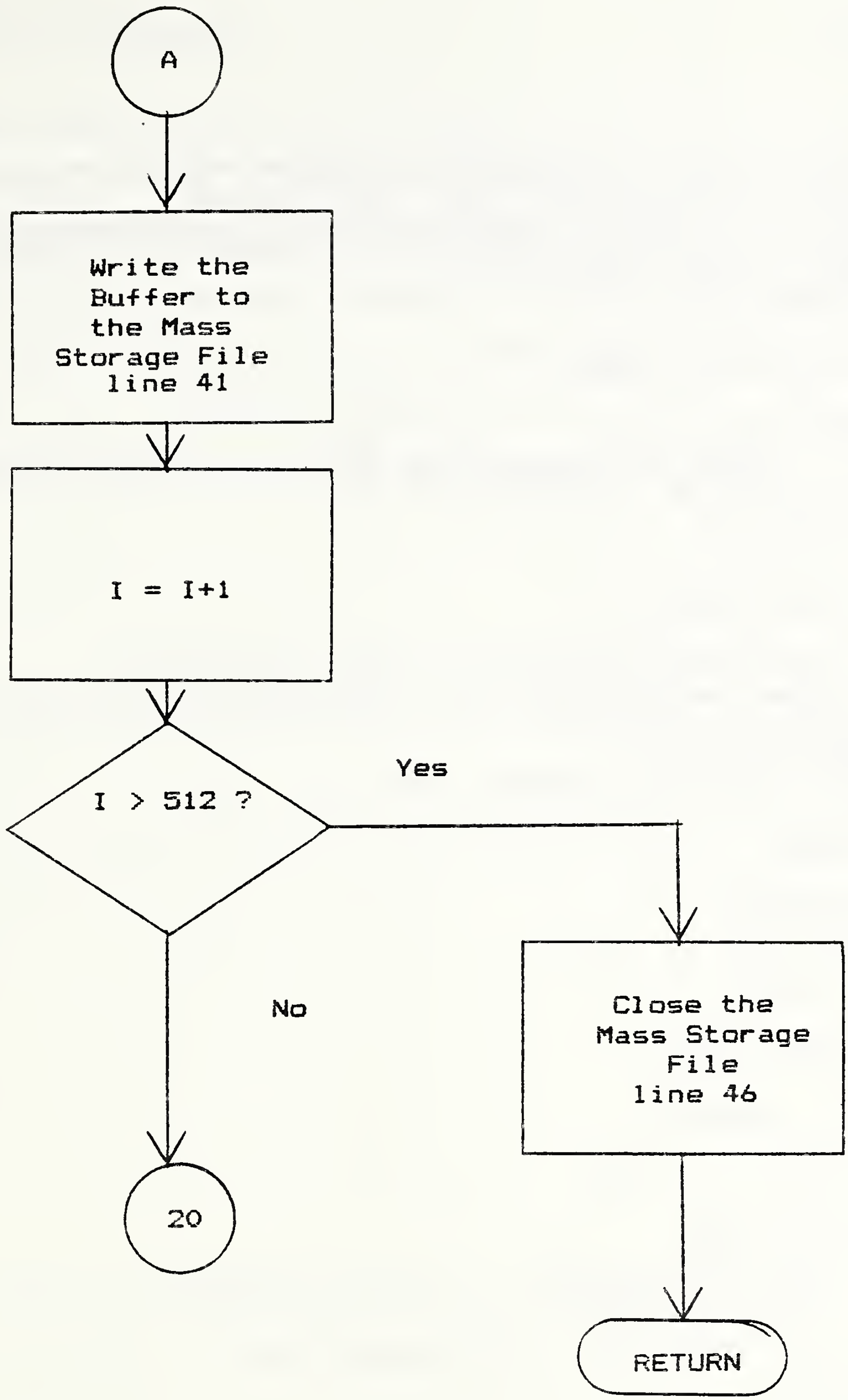
FCB	–	System Function Control Block. INTEGER*2 Array
BUFFER	–	System buffer array. INTEGER*2 Array
TABLE	–	Refresh Memory Channel to use: 1, 2 or 3. INTEGER*2
CHANLS	–	Bitmap for refresh memory specified in TABLE. INTEGER*2

PUTFIL calls the following subroutines:

ZBUFF
IMAGE
SYSIO

5.19.2 Flow Chart




A
Write the Buffer to the Mass Storage File line 41
I = I+1
I > 512 ?
Yes
No
20
Close the Mass Storage File line 46
RETURN

5.19.3 Listing

```
C***********************************************************************
      SUBROUTINE PUTFIL(FCB, BUFFER, TABLE, CHANLS)
C***********************************************************************
      INTEGER*2 FCB(2048), BUFFER(2048), CHANLS(16), TABLE(16)
      INTEGER CHCODE, PBLK(8)
      CHARACTER*16 FD
C
C GET THE PICTURE FILE NAME
C
      WRITE(5,10)
10    FORMAT(' ENTER THE NAME OF THE FILE YOU WISH TO CREATE,'/
     1        ' MAX OF 16 CHARACTERS.')
      READ(5,11) FD
11    FORMAT(C16)
C
C OPEN THE FILE TO UNIT 4
C
      OPEN(4,FILE=FD,STATUS='NEW',ACCESS='SEQUENTIAL',
     1     FORM='UNFORMATTED',RECL=512,BLOCKSIZE=512,IOSTAT=IOS)
      IF(IOS .EQ. 0) GO TO 15
      WRITE(5,12) IOS
12    FORMAT(' IOSTAT ON OPENING THE FILE =',I4)
      STOP
15    CONTINUE
C
C SET UP FILE SIZE
C
      NBYTES = 512
      NREC = 512
C
C GET THE CHANNEL CODE
C
      CHCODE = CHANLS(1)
C
C TRANSFER RECORD AT A TIME
C
      DO 20 I=1,NREC
      CALL ZBUFF(BUFFER,2048)
      CALL IMAGE(FCB,BUFFER,0,I-1,NBYTES,0,CHCODE,-1,1,1,0,
     1           0,0,0,1)
      CALL SYSIO(PBLK,Y'38',4,BUFFER,NBYTES,0)
20    CONTINUE
C
C CLOSE THE FILE
C
      CLOSE(4)
      RETURN
      END
```

6.0 Acknowledgements

The author would like to thank Dr. E. Clayton Teague for encouraging the use of graphics techniques in surface quality studies and insisting that the earlier versions of this program should be made more user friendly. Mr. Saul Baker and Mr. Neal Webber deserve credit for implementing one of those earlier programs. Dr. Theodore V. Vorburger contributed several helpful suggestions that helped make the exposition clearer.

7.0 Bibliography

[1] Appel, A. Some techniques for shading machine renderings of solids, AFIPS Conference Proceedings, Spring Joint Computer Conference, Vol. 31 (1968), pp. 37-45.

[2] Crow, F. C. Shadow algorithms for computer graphics, Computer Graphics, Vol. 11, No. 2 (Summer 1977), pp. 355-361.

[3] Dunham, C. B. The necessity of publishing programs, The Computer Journal, Vol. 25, No. 1 (1982), pp. 61-62.

[4] Foley, J. D. and Van Dam, A. Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Co., Reading, Mass., 1982.

[5] Goldstein, R. A. and Nagel, R. 3-D visual simulation, Simulation, January 1971, pp. 25-31.

[6] Teague, E. C., et. al. Three-dimensional stylus profilometry, Wear, Vol. 83 (1982), pp. 1-12.

[7] Webber, R. L. and Nagel, R. N. Three-dimensional enhancement of two-dimensional images, Jour. Clinical Eng., Vol. 5, No. 1 (Jan.-Mar. 1980), pp. 41-50.

[8] Wyle, C., et. al. Half-tone perspective drawing by computer, AFIPS Conference Proceedings, Fall Joint Computer Conference, Vol. 27 (1967), pp. 49-58.

APPENDIX A

IMAGE PROCESSOR FUNCTIONS

A.1 Summary of Operations

Image data can be transferred from the host computer to any one of three refresh memories either directly or by way of an input function memory. See Figure A1. The task of the input function memory is to directly control the scaling of data in order to ensure that it falls within the 0-255 range, or 8 bits. Once in the refresh memory, it remains there unchanged. Actual image processing is performed by controlling the individual pipeline processor channels, the feedback unit, the histogram generator and the graphics channel.

The individual pipeline processors contain several hardware capabilities. See Figure A2. Along with the ability to scroll an image and change magnification through the zoom hardware, the user may transform the image signals by loading the look-up tables and the output function memory. These, along with the Min-Max register, the constant register and range register, give the user several ways to control the image output to the monitor.

A.2 Some Detailed Capabilities

A.2.1 Input Function Memory

This is a host programmable look-up table that is applied to the data as it is transferred to a refresh memory or graphics memory, both from the host or during an image feedback operation. It is an optional look-up table and can be bypassed if the programmer so chooses. It is used to compact data of up to 13 bits to numbers of 8 bits or fewer and speeds up processing by not requiring data to be scaled in the host computer.

A.2.2 Refresh Memory

Each refresh memory consists of 512 x 512 x 8 bits of random access data storage. This allows the host computer to access any pixel (or bit within a pixel) randomly. Images may be read or written vertically or horizontally by incrementing the location addresses either by rows or columns.

A.2.3 Pipeline Processor Channel

The three parallel pipeline processing channels can perform array arithmetic for each of the three primary colors. Any refresh memory channel (or any combination of refresh memory channels) can be assigned to any of the pipelines (which in turn supply the RGB primary color). The pipelines can add, subtract, multiply and divide image data at real-time rates. The internal capabilities of the pipeline processors will be detailed below.

A.2.3.1 Pipeline Look-Up Tables

Three look-up table memories are provided with each pipeline channel, giving a total of nine. One look-up table in each pipeline channel affects its associated refresh memory. These look-up tables (LUT's) are one of the two programmable processing elements following the refresh memories. The data for the LUT's is loaded by the host computer. The tables are used to implement the four basic arithmetic processes at real-time rates as well as affect image contrast.

A.2.3.2 The Adder Array

This takes the two's complement sum of the look-up table outputs. Three sets are available, one for each primary pipeline.

A.2.3.3 Output Function Memory

Each pipeline contains an output function memory which transforms the outputs of the range registers to generate the final red, green and blue data streams.

A.2.3.4 Min-Max Registers

The Min-Max registers examine the data stream as it emerges from the adder array and determines the dynamic range of the data by finding the minimum and maximum pixel values. These registers are read by the host computer and are used in determining how to set the range register to process the data by the output function memory.

A.2.3.5 Range and Constant Registers

The range registers are used to reduce the data stream from the adder array to a stream for the output function memory. The constant register allows the addition and subtraction of a constant from the data stream before it enters the range register.

A.2.3.6 Hardware Zoom

This allows magnification by way of pixel replication of the displayed image by a factor of 2, 4 or 8 around an arbitrary location. The specification of the center point of the area to be magnified and the magnification factor is accomplished from the host computer. Zoom is nondestructive, in that the original data in the refresh memory is not destroyed.

A.2.4 Color Monitor

This monitor provides both full color and monochrome presentation.

A.2.5 Graphics Refresh Memory

This memory consists of five 512 x 512 one-bit graphics overlay planes. They are treated as an additional refresh memory for the purposes of reading and writing from the host computer. The graphics data, along with the cursor data stream, are fed to the graphics multiplexor. Under program control, this multiplexor can select between displaying graphics or graphics with cursor superimposed.

A.2.6 Programmable Cursor

The host computer can command the cursor position or read back the cursor position at any time. The cursor can be displayed with a constant intensity or blinked. The host computer can also link the cursor position to the trackball unit.

A.2.7 Trackball

The trackball is used to selectively control the X-Y position of the cursor on the monitor screen. It is designed to allow the user to move the cursor in one pixel increments. Four function buttons are provided on the trackball housing. When pushed, the buttons indicate a state change to the host. These states are stored in a register that can be read by the host computer.

A.2.8 Color Assignment Function Memory

This assigns one of the possible 32,768 colors to each graphics plane and dynamically changes the assigned colors under programmatic control as the graphics planes overlay each other. The host computer can program the graphics colors by loading a map into the color assignment function memory. This map defines what color is to be displayed when any one graphics plane is on and also defines a different color to be displayed for each of the possible graphics plane combinations. The ability to dynamically change color assignments for overlapping regions guarantees that each graphics overlay can be distinguished from other graphics overlays at all times.

A.2.9 Histogram Generator

This unit is sometimes called a videometer and is a processing unit that rapidly computes the grey level histogram of the processed data streams just prior to their conversion to video signals at the output of a pipeline. It can generate the histogram of the entire image or of a defined subarea of the image.

A.2.10 Feedback

Except for the image data scaling performed by the Input Function memory, various transformations performed in the hardware do not actually modify the image data which is stored in the refresh memory. If the user wishes to retain the actual processed image data, it may be transferred back by the feedback unit to a refresh memory by way of the Input Function memory. This capability allows the processor to perform iterative operations on an image.

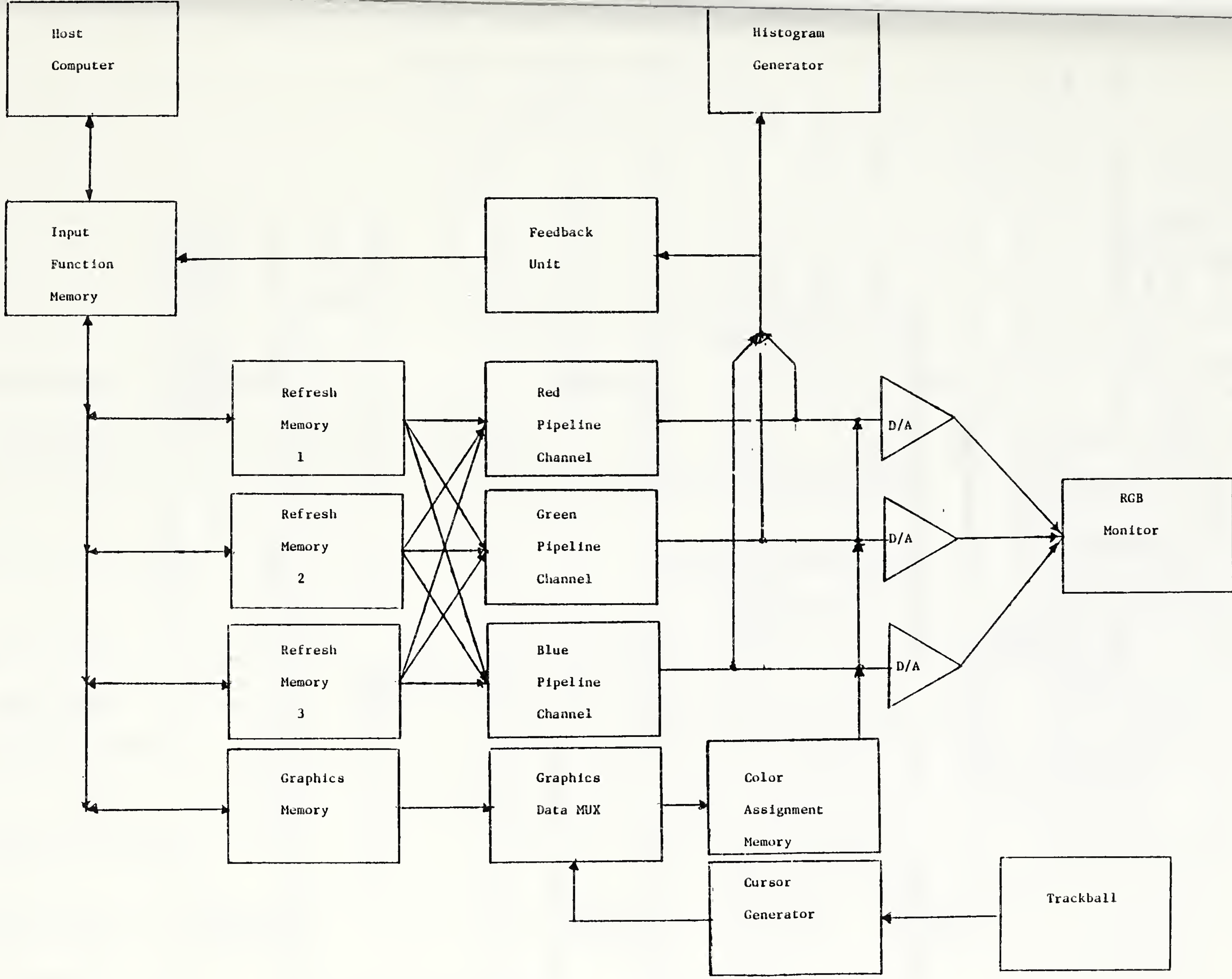


Figure A-1

Image Processor

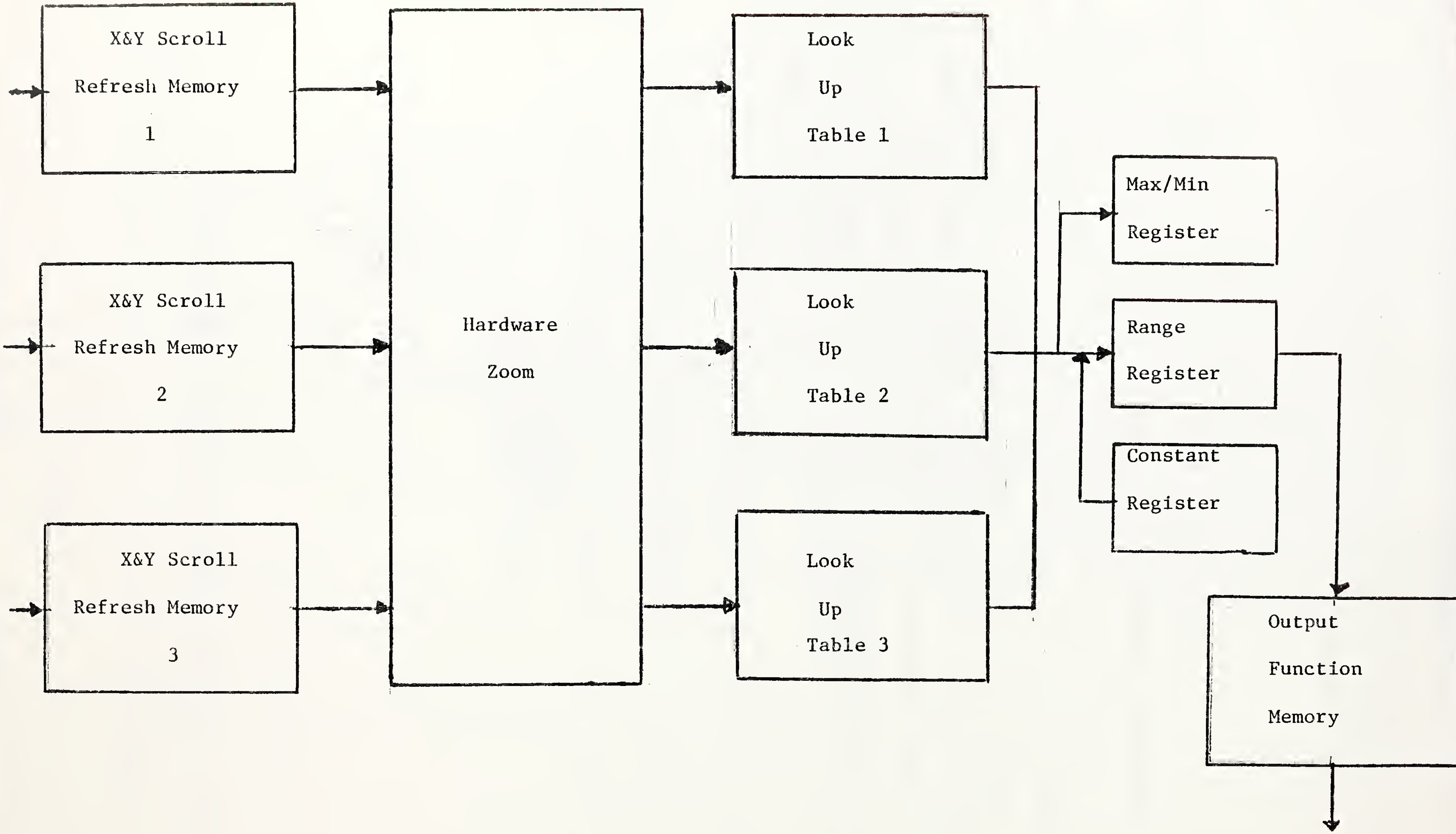


Figure A-2

Pipeline Channel Architecture

APPENDIX B

SYSTEM SUPPORT PROGRAMS

This appendix is devoted to listing the names, functions, calling sequences and the relevant comment portions of the system specific source programs used in the solid generation program. These programs are not available for public use and depend on the architectures of the image processor and the host computer facility. This section is presented so that anyone desiring to implement the SOLID program can understand the functions performed by the various calls not fully documented in this volume. The subroutine calls are divided into image processor subroutines and host computer subroutines.

B.1 Image Processor Subroutines

B.1.1 Subroutine BCHAN

This subroutine blanks an image channel. It is used to turn off a channel link to the monitor.

```
SUBROUTINE BCHAN(FCB, BUFFER, CHCODE, BITPLN)

ROUTINE TO BLANK IMAGE CHANNELS

CHCODE = BIT MAP FOR CHANNELS TO BE BLANKED

INTEGER CHCODE, BITPLN
INTEGER VRSION
INTEGER *2 FCB(1), BUFFER (1)
```

B.1.2 Subroutine CRCTL

This subroutine reads or writes the cursor control register. This register is used to enable or disable the cursor.

```
      SUBROUTINE CRCTL (FCB,ON,RATE,LINKX,LINKY,BUTTON,BEEP,
     1                  MOVE,VRTRTC,READ)

      SUBROUTINE READS OR WRITES THE CURSOR CONTROL REGISTER.


      ARGUMENT DECLARATIONS:

      INTEGER ON,RATE,LINKX,LINKY
      INTEGER*2 FCB(1)
      INTEGER VRTRTC,READ,MOVE,BEEP,BUTTON

      ARGUMENT DESCRIPTIONS:

      ON     -  0 TURNS CURSOR OFF, 1 TURNS CURSOR ON
      RATE   -  0 CURSOR STEADY,
                1 FAST BLINK,
                2 MEDIUM BLINK,
                3 SLOW BLINK.
      LINKX  -  0 CURSOR STATIONARY IN THE X DIRECTION,
                1 CURSOR X POSITION CONTROLLED BY TRACKBALL
      LINKY  -  0 CURSOR STATIONARY IN THE Y DIRECTION,
                1 CURSOR Y POSITION CONTROLLED BY TRACKBALL
      BEEP   -  0 => ENABLE BEEPER, 1 => DISABLE BEEPER
      MOVE   -  0 => NO MOVEMENT, 1 => CURSOR HAS MOVED (READ ONLY)
      BUTTON -  BUTTON WORD (READ ONLY)
      READ   -  0 IMPLIES WRITE, 1 IMPLIES READ.
```

B.1.3 Subroutine DADRS

This subroutine converts display channel numbers to display channel masks. A channel mask represents a 1 in a register bit that addresses the desired refresh memory.

```
SUBROUTINE DADRS (CHMASK, CHANNO, CHCODE, NBANDS)

INTEGER CHCODE, NBANDS
INTEGER*2 CHMASK(1), CHANNO(1)


SUBROUTINE TO CONVERT DISPLAY CHANNEL NUMBERS (0 THRU 15)
TO DISPLAY CHANNEL MASKS (A 1 IN THE CORRESPONDING BIT)

CHMASK - INTEGER ARRAY IN WHICH DISPLAY CHANNEL MASKS ARE
         RETURNED

CHANNO - INTEGER ARRAY CONTAINING DISPLAY CHANNEL NUMBERS TO
         BE CONVERTED

CHCODE - INTEGER MASK WHICH IS THE LOGICAL OR OF ALL DISPLAY
         CHANNEL MASKS

NBANDS - NUMBER OF DISPLAY BANDS
```

B.1.4 Subroutine DCURS

This subroutine turns on the programmable cursor and defines its shape.

```
SUBROUTINE DCURS (FCB, BUFFER, SHAPE, SIZE)

SUBROUTINE TO GENERATE THE PROGRAMMABLE CURSOR

INTEGER SHAPE
INTEGER*2 FCB(1)
REAL SIZE

SHAPE:  1 => SQUARE
        2 => CIRCLE
        3 => PLUS
        4 => CROSS
        5 => BLANK CURSOR

SIZE:   PARAMETER DEFINING THE SIZE OF THE CORRESPONDING
        CURSOR SHAPE.  SQUARE = HEIGHT, CIRCLE = DIAMETER,
        PLUS = HEIGHT, CROSS = HEIGHT.
```

B.1.5 Subroutine DEXEC

This subroutine clears the Function Control Block of all commands.

```
     SUBROUTINE DEXEC (FCB)
     INTEGER*2 FCB(41)

     THIS ROUTINE IS USED TO DUMP ANY DATA
     STILL RESIDING IN THE BUFFER TO THE
     MODEL 70.  IF BUFFER IS NOT BEING USED,
     THE ROUTINE RETURNS IMMEDIATELY TO THE
     CALLING PROGRAM.

     FCB LAYOUT

     FCB(1)  = "FC"
     FCB(2)  = "B"
     FCB(3)  = BUFFER SIZE
     FCB(4)  = NUMBER OF WORDS IN BUFFER
     FCB(5)  = DUMP FLAG
     FCB(6)  = SAVE AREA FOR BUFFER SIZE DURING DUMP
     FCB(7)  = FCB(40) RESERVED
     FCB(41) = BUFFER AREA
```

B.1.6 Subroutine DPLUS

This subroutine is used to draw a plus mark at a specified point in the graphics memory.

```
   SUBROUTINE DPLUS (FCB, BUFFER, CHANNL, PLANES, X, Y, SIZE)

   INTEGER CHANNL, PLANES, X, Y, SIZE
   INTEGER*2 FCB(1), BUFFER(1)

   SUBROUTINE TO WRITE A PLUS AT (X,Y) POSITION

   PARAMETERS:

       CHANNL - MASK OF CHANNELS TO WRITE
       PLANES - MASK OF BIT PLANES TO WRITE
       X      - X POSITION (0 REL)
       Y      - Y POSITION (0 REL)
       SIZE   - WIDTH OF PLUS
```

B.1.7 Subroutine DUNIT

This subroutine initializes the look-up tables for the channel specified and sets various registers needed in order to display an image.

```
    SUBROUTINE DUNIT (FCB, BUFFER, CHANLS, NCHAN, LEVELS)

       THIS ROUTINE REESTABLISHES THE DISPLAY ENVIRONMENT
       REQUIRED IN ORDER TO DISPLAY THE CONTENTS OF THE
       REFRESH MEMORIES WITHOUT ANY RADIOMETRIC CHANGES.

    GLOBAL VARIABLES

       FCB     -  AN INTEGER ARRAY FOR SYSTEM DEPENDENT INFO

       BUFFER  -  A 1024+ WORD INTEGER ARRAY USED AS A WORK
                  AREA FOR THE DESIRED PROCESSING.

       CHANLS  -  AN INTEGER ARRAY CONTAINING THE CHANNEL
                  NUMBERS OF THE CHANNELS TO BE PROCESSED.

       N CHAN  -  THE NUMBER OF CHANNELS TO BE PROCESSED.
```

```
   LEVELS  -  THE NUMBER OF QUANTIZATION LEVELS
              FOR WHICH THE REFRESH MEMORIES ARE
              CONFIGURED.  FOR 8 BIT MEMORIES,
              LEVELS = 2**8 = 256.

 INTEGER*2 FCB(1), BUFFER(1), CHANLS(1)
 INTEGER NCHAN, LEVELS
```

B.1.8 Subroutine DVECT

This subroutine is used to draw a line between two points in the graphics memory.

```
  SUBROUTINE DVECT (FCB, X1, Y1, X2, Y2, CHCODE,
 1                   PLCODE, BUFFER)

  USED TO DRAW A LINE BETWEEN THE POINT (X1,Y1) AND THE
  POINT (X2,Y2).

  INPUTS:

  FCB     -  ARRAY FOR SYSTEM DEPENDENT INFO.
  X1,Y1   -  THE STARTING COORDINATES
  X2,Y2   -  THE ENDING COORDINATES
  CHCODE  -  BIT MAP DESIGNATING IMAGE CHANNEL(S) TO
             BE FILLED IN; WILL USUALLY BE 2**15 FOR
             GRAPHICS (CHANNEL NUMBER 15)
  PLCODE  -  BIT-PLANE BITMAP (I.E., 4 => PLANE 2
             (ZERO REL.))
  BUFFER  -  ARRAY WHOSE ELEMENTS CONTAIN THE WORDS TO
             BE WRITTEN (I.E., BUFFER(K) = -1 OR
             BUFFER(K) = 0 FOR WHITE OR BLACK
             RESPECTIVELY).
             256 ELEMENTS SHOULD BE LOADED.

  INTEGER X1, Y1, X2, Y2, CHCODE, PLCODE
  INTEGER*2 BUFFER(1), FCB(1)
```

B.1.9 Subroutine GRAFE

This subroutine controls any input and output to the graphics control registers. Its function is to enable or disable the graphics display.

```
 SUBROUTINE GRAFE (FCB,DCURSR,DVIDEO,DGRAPH,BLOTCH,STATUS,
1                  STVID,VRTRTC,READ)

 SUBROUTINE WRITES THE GRAPHICS CONTROL REGISTER.


 ARGUMENT DECLARATIONS:

 INTEGER READ, STVID, VRTRTC
 INTEGER*2 FCB(1)
 INTEGER DCURSR, DVIDEO, DGRAPH, BLOTCH, STATUS


 ARGUMENT DESCRIPTIONS:

 DCURSR - DISABLES CURSOR OPTION AND SWITCHES IN GRAPHICS PLANE 7
 DVIDEO - UNCONDITIONALLY TURNS SCREEN BLACK
 DGRAPH - TURNS OFF ALL GRAPHICS CAPABILITY INCLUDING CURSOR
 BLOTCH - SELECT BLOTCH PLANE
 STATUS - SELECT STATUS PLANE
 STVID  - SETS STATUS VIDEO ON
 READ   - READ GRAPHICS REGISTER WHEN SET
```

B.1.10 Subroutine IMAGE

This subroutine writes data from the host computer to a refresh memory or reads a refresh memory in order to transfer data to the host.

```
  SUBROUTINE IMAGE (FCB, PIXELS,
 1                  X1NIT, Y1NIT, NPIXEL, DIRECT,
 2                  CHANNL, PLANES,
 3                  PACKED, BYPIFM, BYTE, ADDWRT, ACCUM,
 4                  VRTRTC, READ)

  SUBROUTINE READS OR WRITES IMAGE DATA.
```

ARGUMENT DECLARATIONS:

```
INTEGER BURST, X1NIT, Y1NIT, NPIXEL, DIRECT
INTEGER*2 FCB(1), PIXELS(1)
INTEGER CHANNL, PLANES
INTEGER PACKED,BYPIFM,BYTE,ADDWRT,VIDORD,ACCUM,VRTRTC,READ
```

ARGUMENT DESCRIPTIONS:

```
PIXELS - AN INTEGER ARRAY TO RECEIVE/CONTAIN THE IMAGE DATA
XINIT  - THE X-COORDINATE OF THE FIRST PIXEL TRANSFERRED
            (0 REL)
YINIT  - THE Y-COORDINATE OF THE FIRST PIXEL TRANSFERRED
            (0 REL)
NPIXEL - THE TOTAL NUMBER OF PIXELS TO TRANSFER
DIRECT - 0 IMPLIES READ/WRITE PROCEEDING TO THE RIGHT,
         1 IMPLIES READ/WRITE PROCEEDING DOWNWARD
CHANNL - A BIT MAP SELECTING THE CHANNEL(S) TO READ/WRITE:
              1 -> IMAGE 0
              2 -> IMAGE 1
              4 -> IMAGE 2
              ETC
         16384 -> IMAGE 14
        -32768 -> IMAGE 15 (GRAPHICS)
              WHEN WRITING ONLY, THESE CODES MAY BE COMBINED
              TO WRITE THE SAME DATA INTO TWO OR MORE CHANNELS.
              FOR EXAMPLE, CHANNL = -32758 WOULD MEAN CHANNELS
              1, 3, & 15
PLANES - A BIT MAP SELECTING THE BIT PLANES TO READ/WRITE,
            NORMALLY -1, IE. ALL BITS.  THE EXCEPTION TO THIS
            RULE IS WHEN WRITING IN THE GRAPHICS CHANNEL
PACKED - 0 IMPLIES 1 BYTE/WORD, 1 IMPLIES 2 BYTES/WORD
BYPIFM - 0 IMPLIES USE IFM, 1 IMPLIES BYPASS IFM
BYTE   - 0 IMPLIES NORMAL, 1 IMPLIES 8 PIXELS/BYTE,
            IE. BINARY DATA.
            **NOTE - X1NIT MUST BE A MULTIPLE OF 8.
ADDWRT - 0 IMPLIES NORMAL, 1 IMPLIES THAT THE DATA IN
            MEMORY(S) IS ORE'ED TO THE DATA PRESENTED FROM
            THE COMPUTER AND THE RESULT IS STORED IN
            THE MEMORY(S).
            **NOTE - USED WHEN WRITING ONLY!!
ACCUM  -  0 IMPLIES NORMAL TRANSFER, 1 IMPLIES 16 BIT
            ACCUMULATOR MODE.
            **NOTE - THE CHANNEL SELECT OR CHANNL
                     PARAMETER MUST BE SET TO SELECT
                     BOTH THE LSB AND THE MSB.  NOTE
                     THAT THE LSB MUST BE IN AN EVEN
                     LOCATION AND THE MSB MUST BE THE
                     NEXT CHANNEL.
VRTRTC -  0 IMPLIES WRITE ANYTIME.
          1 IMPLIES WRITE DURING VERTICLE RETRACE ONLY.
READ   -  0 IMPLIES WRITE, 1 IMPLIES READ.
```

B.1.11 Subroutine INFCB

This subroutine initializes the Function Control Block. See Section B.1.5 for the structure of the Function Control Block.

```
          SUBROUTINE INFCB (FCB, BUFSIZ, LUN)
          INTEGER*2 FCB(40)
          INTEGER BUFSIZ, LUN

          THIS ROUTINE IS USED TO INITIALIZE THE
          FCB ARRAY BEFORE ANY CALLS TO INTERFACE
          ROUTINES OR PRIMITIVES.
```

B.1.12 Subroutine LTCNT

This subroutine reads or writes to the look-up table masks in order to enable or disable them.

```
     SUBROUTINE LTCNT (FCB, MASK, COLOR, VRTRTC, READ)

     SUBROUTINE TO READ OR WRITE THE LUT MASK(S)

     ARGUMENT DECLARATIONS:

     INTEGER MASK, COLOR, VRTRTC, READ
     INTEGER*2 FCB(1)

     ARGUMENT DESCRIPTIONS:

     MASK    -   AN INTEGER WHOSE BIT MAP DETERMINES
                    WHICH LOOK UP TABLES ARE ENABLED
                    AND DISABLED
                    LSB = 1 ==> ENABLE 0TH MEMORY
                    ...ETC.

     COLOR   -   A CODE INDICATING WHICH LUT MASK TO READ/WRITE:
                    1  -   BLUE
                    2  -   GREEN
                    4  -   RED
                    7  -   RED + GREEN + BLUE

     READ    -   0 IMPLIES WRITE, 1 IMPLIES READ.
```

B.1.13 Subroutine MSTCL

This subroutine sends a character from the host computer to the interface board in the image processor in order to clear the interface registers. This is done so that a new command can be sent from the host.

```
SUBROUTINE MSTCL (FCB)
INTEGER *2 FCB(8)
```

B.1.14 Subroutine ONCUR

This subroutine turns on the cursor so that it may be displayed on the monitor.

```
      SUBROUTINE ONCUR (FCB, BUFFER, RED, GREEN,
     1                  BLUE, XPOS, YPOS, BLINK)

      INTEGER XPOS, YPOS, BLINK
      INTEGER*2 FCB(1), BUFFER(1)
      REAL RED, GREEN, BLUE

      ROUTINE TO TURN ON THE CURSOR

      FCB(*)      -  ARRAY FOR SYSTEM DEPENDENT INFO
      BUFFER(*)   -  SCRATCH BUFFER DIMENSIONED <= 1024
      RED         -  FLOATING POINT RED WEIGHT
      GREEN       -  FLOATING POINT GREEN WEIGHT
      BLUE        -  FLOATING POINT BLUE WEIGHT
      XPOS        -  XPOSITION (0, 511)
      YPOS        -  YPOSITION (0, 511)
      BLINK       -  0 => STEADY CURSOR
                     1 => FAST BLINK
                     2 => MEDIUM BLINK
                     3 => SLOW BLINK

      ALL WEIGHTS MUST BE IN RANGE  0. ==> 1.
```

B.1.15 Subroutine RBUTN

This subroutine is used to read the location of the cursor. The viewer interacts with the image processor by pushing a button on the trackball housing. The image processor then locates the cursor.

```
SUBROUTINE RBURN (FCB, BUTTON, X, Y)

ROUTINE TO READ BUTTON WORD AND
CURSOR POSITION

INTEGER BUTTON, X, Y
INTEGER*2 FCB(1)
```

B.1.16 Subroutine STCOL

This subroutine is used to identify what colors should be displayed for each graphics bit plane. This does not enable or disable the planes.

```
SUBROUTINE STCOL (FCB, BUFFER, PLANE, RED, GREEN, BLUE, INSERT)

SUBROUTINE TO SET COLOR OF GRAPHICS PLANES

INTEGER PLANE, INSERT
INTEGER*2 FCB(1), BUFFER(1)
REAL RED, GREEN, BLUE

PLANE  -  GRAPHICS PLANE DESIRED. (0 <= PLANE <= 7)
RED    -  INTENSITY VALUE FOR RED COMPONENT (0 <= RED <= 1.)
GREEN  -  INTENSITY VALUE FOR GREEN COMPONENT (0 <= GREEN <= 1.)
BLUE   -  INTENSITY VALUE FOR BLUE COMPONENT (0 <= BLUE <= 1.)
INSERT -  0 => OVERLAY, 1 => INSERT
```

B.1.17 Subroutine XCOLR

When several bitplanes have different colors and a graphics memory pixel is turned on for several bitplanes, then this subroutine defines what color should be displayed.

```
SUBROUTINE XCOLR (FCB, BUFFER, PLANE, INSERT)

THIS ROUTINE IS USED TO DEFINE THE COLORS FOR AREAS OF
INTERSECTION BETWEEN GRAPHICS PLANES

A DISTINCT COLOR IS OBTAINED BY DOING AN EXCLUSIVE OR
OF ALL THE COLOR WORDS CORRESPONDING TO THE INTERSECTING
PLANES

INTEGER PLANE, INSERT
INTEGER*2 FCB(1), BUFFER(1)

PLANE  -  PLANE FOR WHICH WE ARE DEFINING THE INTERSECTIONS
INSERT -  OVERLAY MODE, INSERT = 1, OVERLAY = 0
```

B.1.18 Subroutine ZBUFF

This subroutine writes zeroes into a buffer specified by BUFFER. COUNT represents the number of zeroes written.

```
SUBROUTINE ZBUFF (BUFFER, COUNT)

INTEGER COUNT
INTEGER *2 BUFFER (1)
```

B.2 Host Computer Subroutines

B.2.1 Subroutine ISBYT

This subroutine stores a byte from the low order position of one argument to any arbitrary position in another argument.

```
CALL ISBYTE (K, M, N)
```

where:

K - is the input argument whose least significant byte is to be stored. INTEGER

M - is the output argument into which a byte is stored. INTEGER

N - is the number of the byte in M where storing takes place. INTEGER

B.2.2 Subroutine SVC7

This subroutine is used to correct an error in the FORTRAN INQUIRE statement for the host processor. It is a local correction only. The subroutine fetches the file control block for a specified file.

```
CALL SVC7 (IFCB)
```

where:

IFCB - INTEGER Array of at least 6 elements representing the file control block.

B.2.3 Subroutine SYSIO

This subroutine performs input/output at the byte level.

```
CALL SYSIO (FBLK,FC,LU,START,NBYTES,RANADD)
```

Arguments:

FBLK is an INTEGER*4 array of at least five elements.

FC is an INTEGER*4 argument that specifies the I/O function to be performed.

LU	is an INTEGER*4 argument that specifies the logical unit on which to perform I/O.
START	is any type of argument except character that specifies the starting address of the buffer used in the I/O transfer.
NBYTES	is an INTEGER*4 argument specifying the number of bytes to be transferred in this I/O operation.
RANADD	is an INTEGER*4 argument that specifies the logical record number (starting at 0) to be accessed on data transfer requests when bit 5 of FC is set. This argument should be a zero if random I/O is not being used.

APPENDIX C

IMAGE DATA ACQUISITION

The image data used as input to the program documented in this report came from a three-dimensional profilometry instrument. Three-dimensional profilometry is the process of obtaining a topographic map of a surface from many parallel traverses of a stylus (See Teague, et.al. [6]). The number of data values required to represent a topographic map adequately can, depending on the spatial resolution desired, be as large as 0.25×10^6. This large amount of data poses a formidable problem in acquisition, processing, and displaying.

This problem can be surmounted with the use of a large minicomputer system interfaced to both a specially designed stylus stage and a raster graphics array processor and display unit. The electrical analog output of the stylus transducer is converted into an intensity value at a corresponding point on the screen of a television monitor. A schematic diagram of the system for acquiring three-dimensional stylus profilometry data and for displaying the data as an intensity image is shown in Figure C1. The system is composed of a commercial stylus transducer, a precision X-Y stage built at NBS and a 32-bit minicomputer system with a core memory of 4 million bytes, a mass storage of 160 million bytes and a raster graphics display unit which contains hardware for video rate memory refresh of a color television mmonitor and video rate iterative processing of data stored in the refresh memories (see Appendix A).

As the stage moves the test specimen beneath a fixed stylus location, the electrical signal from the stylus transducer is converted to 12-bit digital values at an array of 512 x 512 X-Y positions and stored on a disk. This is the normal array size, being controlled by the size of the graphics refresh memory size. Using the graphics display unit, the optimum 8 bits of the data are then selected for storage in the refresh memory and for display and processing. From this array of digital values, an image of the topography is generated in which the intensity on the monitor screen is proportional to the surface height of the specimen at the corresponding surface location. Once the data are in refresh memory, a variety of transformations may be applied to enhance visual perception of surface features. The program documented in this volume is one of them.

A schematic picture of the stage is given in Figure C2. For more details on the stage design, the reader is referred to the article by Teague, et.al. [6]. Motion in both axes is produced by stepping motors under control of the minicomputer. Position determination of the X-Y stage is done by way of a commercial interferometer system. Scanning areas cover approximately 1 mm^2.

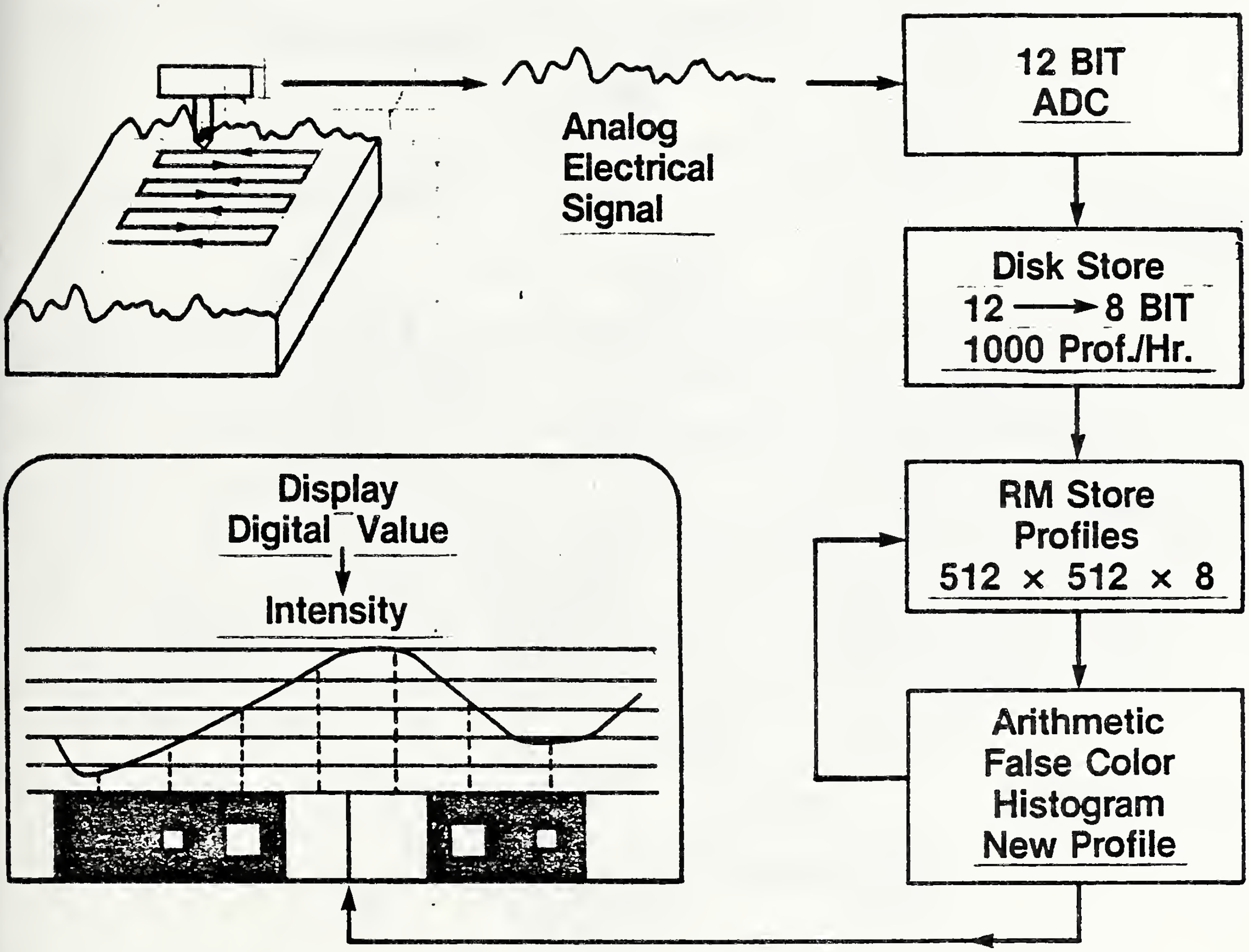


Figure C-1

Surface Data Transfer

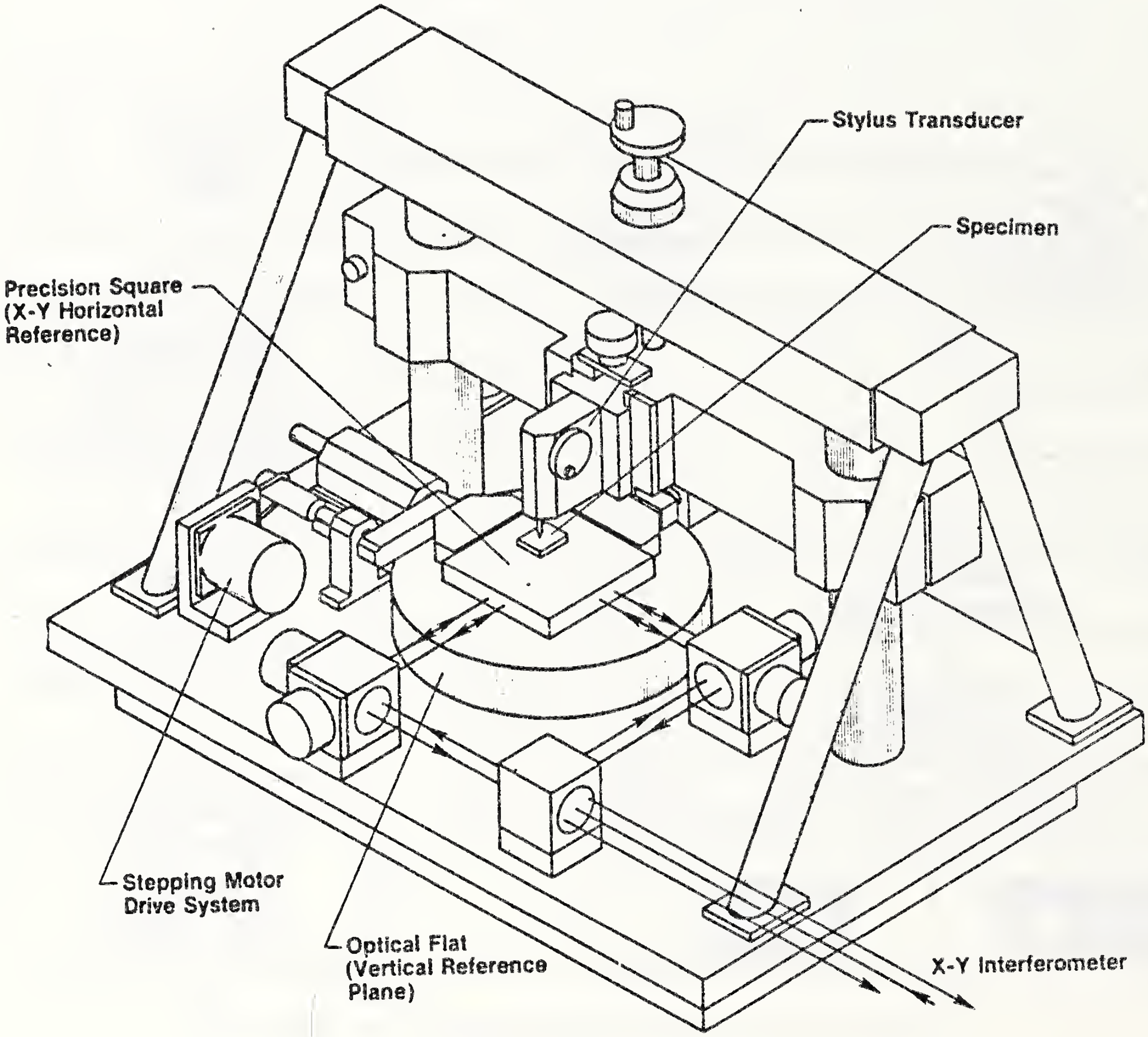


Figure C-2

Three Dimensional Surface
Profilometer

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 85-3121	2. Performing Organ. Report No.	3. Publication Date June 1985

4. TITLE AND SUBTITLE

SOLID - A FORTRAN Program for Displaying Three-Dimensional Surface Topographies

5. AUTHOR(S)

David E. Gilsinn

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

Graphic images of digital surface topography can be viewed as topographic maps. The intensities displayed on computer monitors represent surface height. Hidden surface graphics techniques can be used to project a solid projection of the surface on a high resolution computer monitor. The program documented here implements a multi-purpose ray tracing algorithm to a) cast shadows, b) remove hidden surfaces, and c) project a solid image on a monitor. Although the algorithm is written in FORTRAN, its implementation depends on a unique image processor. This report details the underlying geometry, program logic, and host computer-to-image processor interface requirements, and should enable a user to modify the program for a different image processor implementation.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

graphics; hidden surface algorithm; image processing; ray tracing algorithm; surface topography; three-dimensional display

13. AVAILABILITY

[X] Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

[X] Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

235

15. Price

$20.50