A PORTABLE GRAPHICS PACKAGE FOR THREE-DIMENSIONAL SURFACE RECONSTRUCTION

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston - University Park

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Yu Ping Wing

December, 1987

ACKNOWLEDGEMENT

I would like to express my gratitude to my thesis advisor, Dr. Goffredo Pieroni for his guidance and assistance, and also to Dr. Ramez Elmasri and Dr. Atam Dhawan for serving on the committee.

A great many people have helped me in this project in a variety of ways. They are Chris Yam who helped me use the word processor and the laser jet printer, and Tom Hicks who provided his time in editing my grammar.

Finally, I thank my fiancee, Bernice Sit, for her patience in typing and testing my programs and her encouragement during the development stages.

A PORTABLE GRAPHICS PACKAGE FOR THREE-DIMENSIONAL SURFACE RECONSTRUCTION

An Abstract of a Thesis
Presented to
the Faculty of the Department of Computer Science
University of Houston - University Park

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

By
Yu Ping Wing
December, 1987

ABSTRACT

The CORE protocol provides a list of standard features for constructing a program in dealing with two-dimensional and three-dimensional object representations. The most important features of a such system are linear transformation, windowing, clipping, viewing transformation, projection, hidden-line elimination and hidden-surface removal.

In this thesis, the implementation of the CORE system is based on Steven Harrington's book "Computer Graphics - A Programming Approach" by McGraw Hill. A list of basic algorithms is critically analyzed. Errors were discovered in some important algorithms as proposed by Harrington. A corrected version is presented and implemented.

Finally a set of experiments for constructing 3D objects has been performed by using different output devices : Lexidata, Tektronix and Printronix. The principal one consists of displaying a 3D surface of a human heart obtained by a sequence of PET images.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The main purpose of this graphics package is to provide users a very portable system which does not require specialized display hardware. Even the low resolution of an ordinary dot matrix printer is adequate to show how graphics work. The software is built around the graphics standard, the CORE system. Most of the graphics principles are based on simple analytic geometry.

The CORE system is organized in three areas, namely dimension, input and output. Dimensionally speaking, two levels are represented, two-dimensional operations in the lower level and the three-dimensional operations in the higher level. They both can be addressed at the same time. Output-wise, a temporary display file is at the basic level. Buffered output is employed to retain display attributes of segments while dynamic output buffer is for segments dealing with image transformation. Input can take three different forms: 1) no input, 2) synchronous and 3) asynchronous interaction. In this implementation, device initialization is needed. The only user inputs are from the

keyboard and the data file.

This CORE version as proposed by Steve Harrington can allow the user to create a two-dimensional or three-dimensional representation of an object. Curves construction is also implemented. The system aims to specify the basic graphics capabilities, providing a foundation for more advanced techniques.

During the process of implementation of this system, errors were found, both simple and complex, in Harrington's book. They will be discussed in detail in the chapters that follow. Of course algorithms presented by Harrington are not necessarily unique and the most efficient. The primary objective is to provide the ease of understanding the principles. However, a list of basic and standard alogrithms is analyzed here along with those of Harrington's to give the reader a broader view of the graphics principles. References have been drawn from several authors. Among them are Donal Hearn and M. Pauline Baker, Roy A. Plastock and Gordon Kalley, Pavlidis, Newman and Sproull.

In the application, several techniques have been

employed to construct the three-dimensional surface representation. These techniques are common in image processing. They are triangulation, convex hull construction and graph theory. The proposed algorithms apply specifically to a given set of two-dimensional parallel cross-sections of an object. Computation complexity is not analyzed in this thesis. The result of the implementation of this CORE protocol as demonstrated by the experiments is found very satisfactory in three different display hardware systems; these are namely Lexidata, Tektronics and Printronix. Better display quality is obtained with a device of higher resolution. Modification for use in different devices is kept minimal in order to make the system protable. The system is implemented in high level language FORTRAN 77. Future enhancement, for example sophisticated shading algorithm and splined surfaces, can be added to this system.

CHAPTER 2

GRAPHICS SYSTEM IMPLEMENTATION

2.1 INTRODUCTION

Computer graphics has made tremendous progess in the past ten years. High resolution display hardware and software are now affordable even at the personal computing level. Computer graphics improves the communication between human and machine. Many applications now rely heavily on computer graphics. To mention a few, flight simulation training pilots, graphical presentation of data, computer aided design in the car industry, architectural design, VLSI cirucit design, video games and animation in the entertainment sector. The main reasons for the effectiveness of computer graphics in these applications are the speed and cost.

2.2 GRAPHICS DESIGN PHILOSOPHY

A graphics system can be defined as a collection of hardware and software designed to make it easier to use graphic input and output in computer programs. The software package is a set of subroutines or functions used by an

application program to generate pictures on the display device. The construction of the system has to meet the basic requirements; simplicity, portability, consistency, completeness, robustness, performance and economy. On the other hand, the design should not be unduly influenced by hardware features.

Functionally as a whole, the system can be divided into sets, each set handling a particular kind of task. These functions are summarized as follows:

1. Graphics primitives which are used to display straight lines, text strings, polygons and other simple graphical items.
2. Windowing and clipping which allow the programmer to choose his viewing coordinate system and to define the visible boundary of the picture.
3. Segmenting functions which provides dynamic manipulation of subpicture structure.
4. Transformation functions which include scaling, translation and rotation in both two-dimension and three-dimension, and projection in three-dimension.

No system can do without graphics primitives and details of the screen coordinates have to be transparent to the user. Therefore the purpose of a graphics system is to make

programming easier for the user.

2.3 GRAPHICS DISPLAY COMPONENTS

The modern graphics system consists simply of three elements: A frame buffer which is an internal memory array storing the image as a matrix of intensiy or color; a cathode-ray tube monitor where image is displayed; a simple display controller which is an interface between the frame buffer and the TV monitor (figure 2.1).

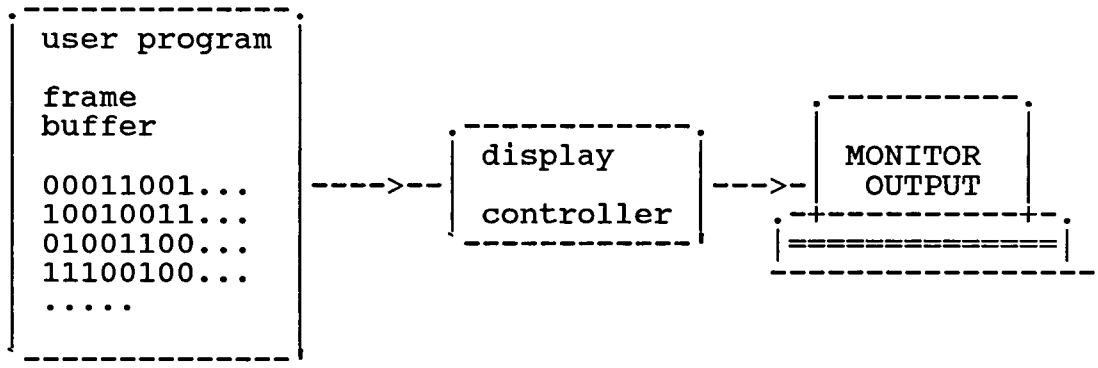


Figure 2.1 Basic graphics system

Inside the frame buffer, the image is stored as a pattern of binary digital numbers which represent a rectangular array of picture elements called pixels. Each pixel requires at least one bit of intensity information,

light or dark, and further bits are needed if shades of grey or different colors are desired.

The display controller simply reads each successive byte of data from the frame buffer and converts its bits, 0s and 1s into the corresponding video signal. This signal is then fed to the display monitor producing the desired image on the screen. The display controller repeats this operation 30 times a second in order to maintain a steady picture on the screen.

In order to change the displayed picture, all that is needed is to modify the frame buffer's content.

2.4 BASIC GEOMETRICAL ELEMENTS

A display screen can be viewed as a cartesian coordinate system with horizontal as the x-axis and vertical as the y-axis by convention. Such arrangement defines a two-dimensional display. The following can now be defined precisely on the display screen:

1. Point: specified by (x,y) coordinate pair
2. Line segment: specified by joining two points (x1,y1) and (x2,y2)

3. Vector: specified by direction and length. It has no fixed position in space.

To apply the mathematical notion to actual graphics display, we limited ourselves to a finite set of values imposed by the physical dimensions of the display hardware system. A point is represented by a pixel, the smallest addressable screen element. Then a line segment is constructed from a finite number of points. The maximum number of distinguishable points which a line may have is called the resolution of the display screen. For example, a resolution of 100 dots (pixels) per inch indicates two dots 1/100 inch apart can be distinguished from each other. The greater the number of points the higher the resolution. If

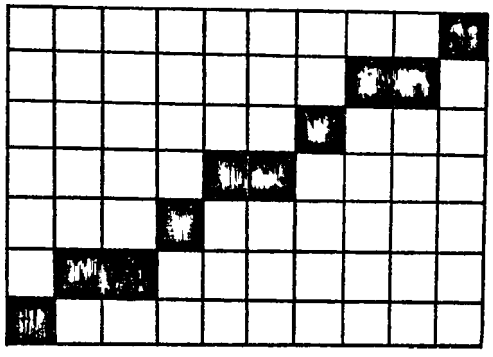


Figure 2.2 Staircase effect of a line

display unit has a low resolution, the line or curve will appear as a staircase (figure 2.2).

Pictures and shapes, whether two-dimensional or three dimensional, can be generated from these basic geometrical elements. Changes of the display will be performed by applying standard mathematical techniques such as affine transformation, clipping and projection.

2.5 VECTOR GENERATION - DDA

In order to generate a smooth line represented by pixels, an ordinary differential equation called digital differential analyzer DDA is used. The line represented by DDA is

$$dY/dX = (Y2 - Y1)/(X2 - X1)$$

where (X1,Y1) and (X2,Y2) are two points. This can be expressed in parametric form with two end points given.

$$X = X1 + (X2 - X1)\ u$$

$$Y = Y1 + (Y2 - Y1)\ u$$

where u is the parameter. When u = 0, X = X1, and Y = Y1. When u = 1, X = X2 and Y = Y2. The idea is to start plotting a point at u = 0 and increment u by small steps until it reaches 1. In terms of a coordinate pair, (X,Y) starts at (X1,Y1) and steps up by an amount

$$STEPS = MAX((X2 - X1), (Y2 - Y1))$$

$$XINCREMENT = (X2 - X1)/STEPS$$

$$YINCREMENT = (Y2 - Y1)/STEPS$$

until (X2,Y2) is reached. The following pseudocodes represent such iteration:

```
X = X1;   Y = Y1

For i = 1 to STEPS do
     X = X + XINCREMENT
     Y = Y + YINCREMENT
enddo
```

The line drawn with this simple DDA algorithm is shown in figure 2.3.

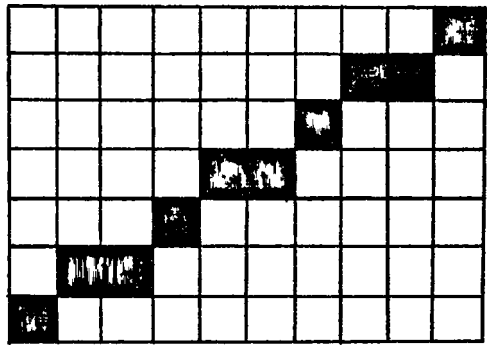


Figure 2.3 Straight line generated by DDA

2.6 CHARACTER GENERATION

Along with lines and points, characters strings are also necessary to convey the meaning of the drawing on the screen. There are two forms of character generation :

stroke=method and dot-matrix method. In this implementation, stroke-method is used in order to allow user to do transformation on characters, e.g. scaling, translation and rotation.

The character stroke sequences are coded in a file. Each character is defined by 5 pixels times 7 pixels in dimension (see figure 2.4). Each stroke is then coded by two operations: M for move to a position (X,Y) without drawing and L for draw a line from current position to a

```
6|- - * - -
5|- - - - -
4|* - - - *
3|- - - - -
2|* - - - *
1|- - - - -
0|* - - - *
  0 1 2 3 4
```

code	relative position	
	x	y
M	0	0
L	0	4
L	2	6
L	4	4
L	4	0
M	0	2
L	4	2
	0	0

Figure 2.4 Coding of character 'A'

point with (DX,DY) relative to the current. Coding is terminated by a triplet (space, 0, 0). The file is then converted to another data structure to be loaded into memory during system initialization. The structure contains two tables : A link list and the codes (see figure 2.5). The link list is to store the starting location of the

coded sequence of each stroke character. The ASCII value of the character is mapped to this list by an offset of 31. The content in the list will locate the sequence of the character codes in the character table. For instance, character 'A' has an ASCII value of 65. Its locator is at 65 - 31 = 33 of the link list. From there, index 53 is read. Hence all codes from location 53 down in the character table are for character 'A'.

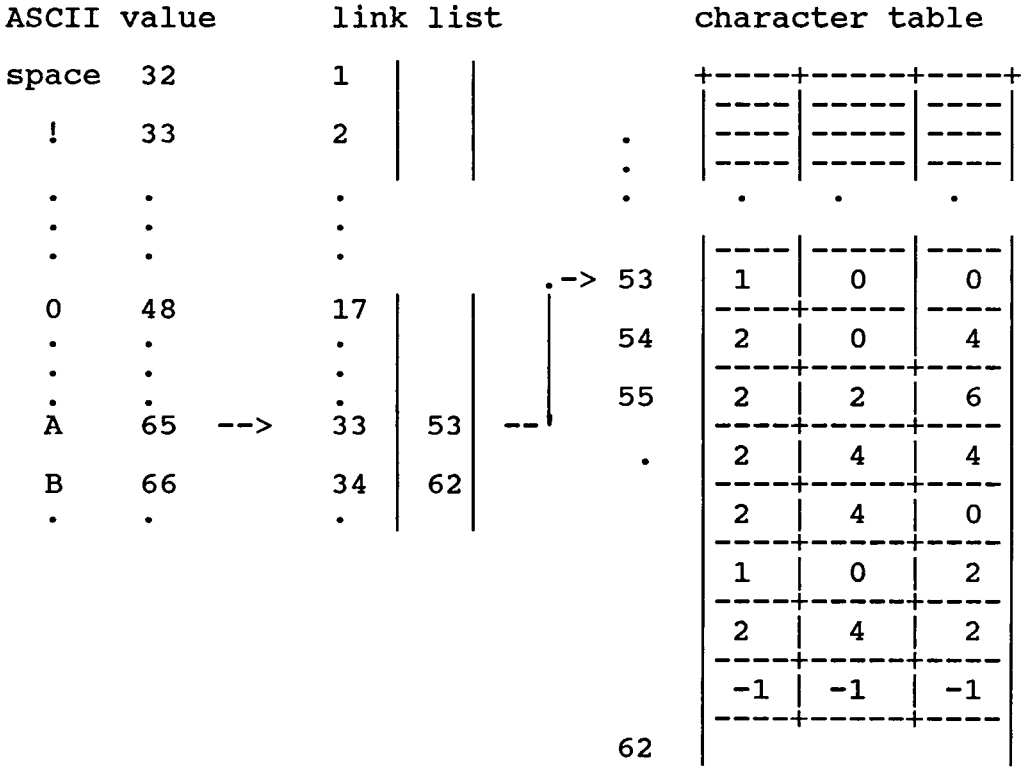


Figure 2.5 Mapping from ASCII to character table

2.7 GRAPHICS PRIMITIVES

When starting a graphics program, system initialization is performed. It includes hardware setup, physical dimensioning, clearing screen and storage allocation to hold the image for instance. All these are done in a routine called INIT_SYSTEM. Upon termination, device storage deallocation and other housekeeping routines have to be done. Regardless of the differences in display devices, a set of graphics primitives commands must exist as a basic tool to construct pictures on the screen. There are two different sets to draw lines but similar in function; one is for use in absolute position and the other is for relative position in space.

a. Absolute Commands

MOVE_ABS_2(X,Y)	move to a position (X,Y) on screen without drawing
LINE_ABS_2(X,Y,COLOR)	draw a straight line from the current position to the point (X,Y) with a certain color or intensity COLOR

b. Relative Commands

MOVE_REL_2(DX,DY)	move to a new position whose coordinates are DX,DY away

from the current position

LINE_REL_2(DX,DY,COLOR) draw a straight line from the current position to a new point which is (DX,DY) away from it with a color or intensity COLOR

Another primitive operation is to draw text. The string characters output may be drawn by either the dot-matrix or the stroke-method. On CRT and line-printer, characters are generated by hardware. With others like Lexidata and Tektronix, the stroke-method can be employed.

Utilities on string operations are given below :

TEXT(STR, COLORS) display the string STR of characters with individual colors starting at its lower left corner at the present position

SET_CHARUP(DX,DY) define the direction (DX,DY) of the string to be printed

SET_CHARSPACE(S) define the spacing between characters in terms of fraction of character

These basic primitives can be extended to include the drawing of polygons. A polygon is represented as a closed

figure consisting of straight line segments connected end to end. The figure can be either concave or convex. In this system, the polygon has a minimum of 3 sides and at most 31 sides. The command is

POLYGON_ABS_2(AX,AY,ACOL,N) draw an absolute polygon whose coordinates of vertices are in arrays AX,AY and side colors in ACOL and number of sides defined by N

POLYGON_REL_2(AX,AY,ACOL,N) draw a polygon relative to the current position

2.8 DISPLAY FILE

In graphics environment, it is often necessary to reconstruct pictures repeatedly in order to achieve dynamic picture changes, for example image transformations. Besides, it is also desirable to have a machine independent routine to deal with such display changes. This suggests a memory storage called display file to save instructions rather than the picture itself. In doing so, it takes up less memory and not every display device has a frame buffer. These instructions will generate the image by a display file interpreter. The structure of the display file is a multiple array set. It contains a sequence of

operation codes which indicate what kind of command, operands which are coordinates of a point, and the color or intensity at this position. The opcodes are defined as follows:

opcode < 0 : set line style display (line color)

opcode 1 : MOVE command

opcode 2 : draw LINE command

opcode 3 - 31 : draw POLYGON with sides 3 <= n <= 31

opcode > 31 : draw character command

The interpreter routine examines the display file and acts appropriately to cause the LINE or MOVE to be carried out on the display. Actual drawing depends on the hardware. If a frame buffer is used, DDA routine is called by the command to generate the image before output to the screen or printer. Otherwise, the command is sent to the display device directly.

There are two routines to allow user to control the display without knowing the existence of the display file. In other words display file is transparent.

NEW_FRAME to indicate that frame buffer should be cleared before showing the display file

MAKE_PICTURE_CURRENT to interpret the display file and then display the frame buffer

The logics of such a scheme is

MAKE_PICTURE_CURRENT -> INTERPRET -> DISPLAY
the display file the picture

2.9 FILLING POLYGONS

The display images become more appealing and interesting if they can be filled with color or light intensities rather than just plain line drawings. Many of the shapes can be represented by polygons. Coloring is possible with raster display devices because pixels are

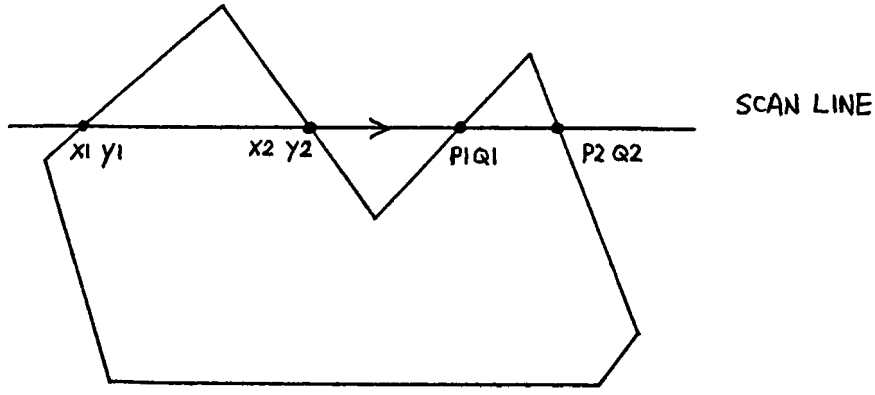


Figure 2.6 x values are paired and used for line drawing

addressable. An algorithm to fill the interior of a polygon is based on the inside test of points on the scan line which crosses the boundaries of the tested polygon. It begins by ordering the polygon sides on the largest y

value and scans down the polygon from the largest y. For each y, it computes the intersection with the polygon edges. If an intersection exists, the x values are sorted and paired. Such a pair represents a region of the scan line in which visible pixels should be displayed. The smallest x value will be the left polygon boundary (see figure 2.6).

CHAPTER THREE

TRANSFORMATIONS, SEGMENTATION AND CLIPPING

3.1 TWO-DIMENSIONAL TRANSFORMATION

Pictures in graphics system are basically represented by coordinate points. By applying appropiate geometric transformations to these coordinates, pictures can be changed in shape and position. This provide a useful complement to graphics design. The basic transformations are scaling, translation and rotation.

Using homogeneous coordinates, these three transformations are defined in terms of matrices as follows:

a. scaling where the size of an object is changed by a factor SX in x direction and SY in y direction

$$[X'\ Y'\ 1] = [X\ Y\ 1] \begin{bmatrix} SX & 0 & 0 \\ 0 & SY & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

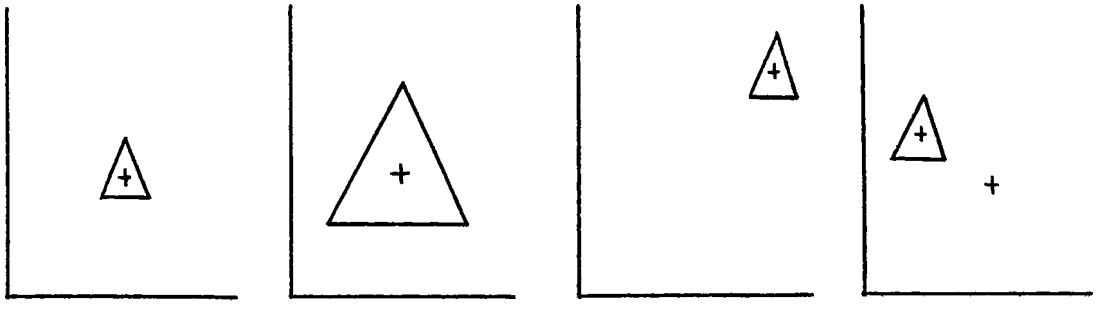
b. translation which changes the position of an object from one place to another along a straight line by (Tx,Ty) in distance

$$[X'\ Y'\ 1] = [X\ Y\ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 1 \end{bmatrix}$$

c. rotation which changes the position of an object along a circular path with a clockwise rotation angle A about the origin

$$[X'\ Y'\ 1] = [X\ Y\ 1] \begin{bmatrix} \cos A & -\sin A & 0 \\ \sin A & \cos A & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 3.1 shows these transformations.



original scaling translation rotation

Figure 3.1 Scaling, translation and Rotation

In the implementation, the transformation process is applied at the time the display file is interpreted (see figure 3.2). The user is restricted from building complex

set up transform matrix -> get point from display file -> transform the image point -> display it

Figure 3.2 Addition of transformation

transformations. Instead he is allowed to setup the scaling, translation and rotation at one time.

Rotation about an arbitrary point can be determined by matrix concatenation of operational matrices. The order is first to translate the center of rotation to the origin,

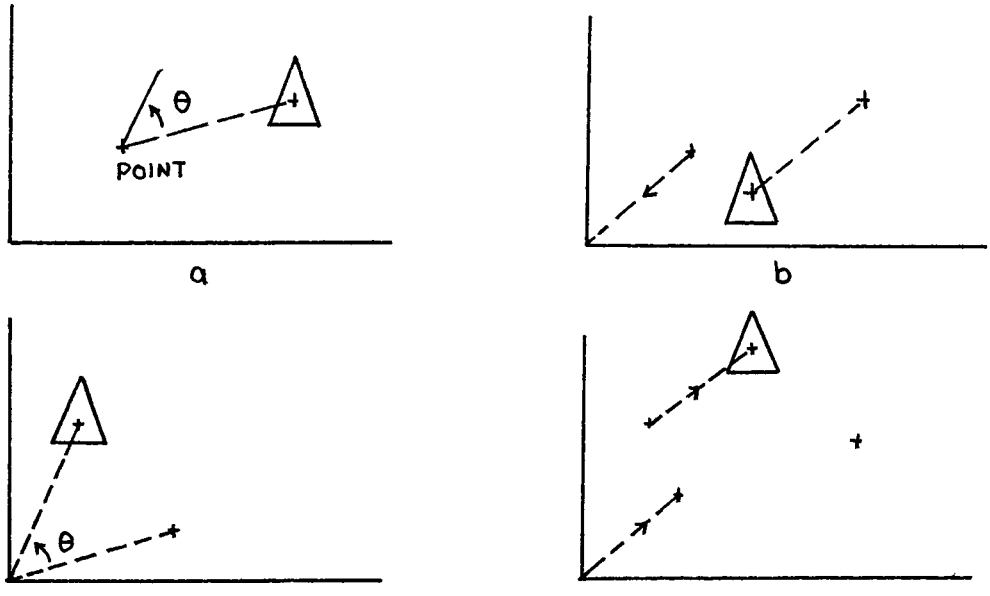


Figure 3.3 Rotation about an arbitrary point

second rotate about the origin of the desired angle, and third translate the center of rotation back to its original position. This sequence is presented by the matrix product

$$T1 = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xc & -Yc & 1 \end{matrix}$$

$$R = \begin{matrix} \cos A & \sin A & 0 \\ -\sin A & \cos A & 0 \\ 0 & 0 & 1 \end{matrix}$$

$$T2 = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xc & Yc & 1 \end{matrix}$$

$$T1\ R\ T2 = \begin{matrix} \cos A & \sin A & 0 \\ -\sin A & \cos A & 0 \\ -Xc * \cos A + Yc * \sin A + Xc & -Xc * \sin A - Yc * \cos A + Yc & 1 \end{matrix}$$

Figure 3.3 depicts such a transformation sequence.

The user defined routines are SCALE(Sx,Sy), TRANSLATE(Tx,Ty) and ROTATE(A).

3.2 SEGMENTATION

For many applications, images are often composed of several pictures. Each subpicture can be composed together to form a new image. Manipulation of pictures in terms of their component parts are more flexible and appealing to the user. By defining each object in a picture as a separate entity, a user can make modifications to the picture more easily. To reflect such subpicture structure,

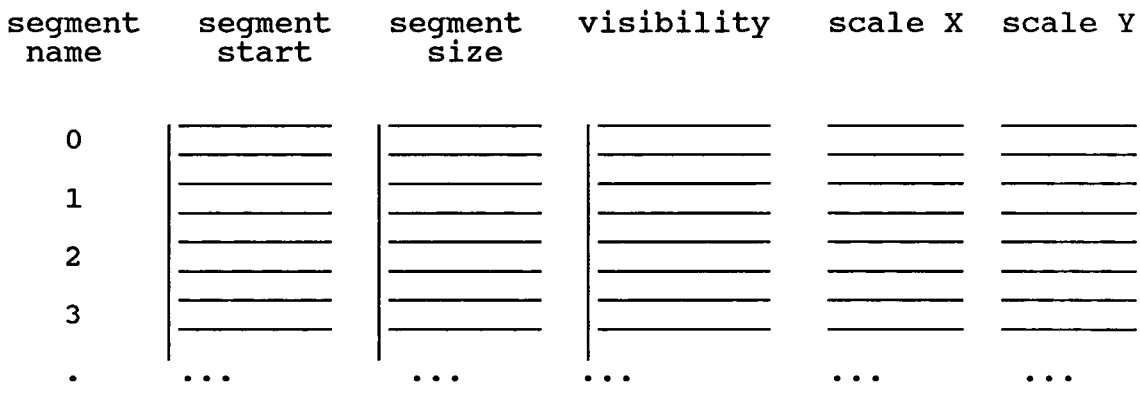


Figure 3.4 Segment table

the display file is reorganized by being divided into segments. Each segment corresponds to a component of the overall display. Associating with each segment is a set of attributes. These attributes are visibility, scaling, translation and rotation. Visibility is used to determine if the stored subpicture should be displayed or not while transformation is performed on each segment independently. This attribute information is represented by means of a segment table of simple arrays (see figure 3.4). Each segment is given a numeric name. The unnamed segment is named 0 and can be used for nonsegmented display system. Therefore, for each segment the display file interpreter will only interpret those visible ones.

This system does not permit two segments open at the same time and no two segments have the same name. Once a segment is created, all subsequent graphics commands will belong to this segment until a close segment command is issued. Basic segment related commands are :

```
CREATE_SEGMENT(Numeric_name)
CLOSE_SEGMENT
DELETE_SEGMENT(segment_name)
DELETE_ALL_SEGMENTS
RENAME_SEGMENT(old_name, new_name)
SET_VISIBILITY(name, on_off)
SET_IMAGE_TRANSLATION(name, Tx,Ty)
SET_IMAGE_TRANSFORMATION(name,Sx,Sy,Angle,Tx,Ty)
```

The display system with segment file organization is depicted in figure 3.5.

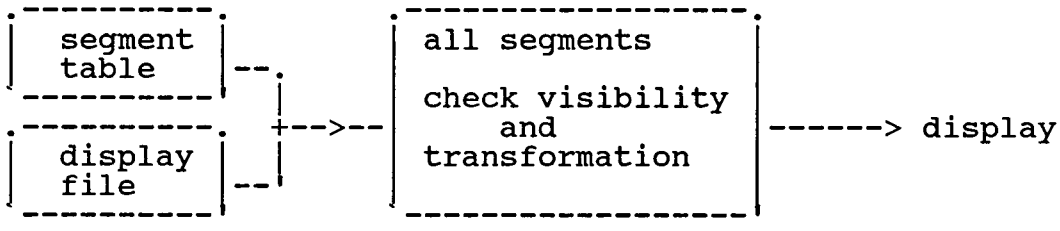


Figure 3.5 Segment organization.

3.3 WINDOWING AND CLIPPING

It is very useful if one can select an area of a picture to display and place it in a specified region of the screen. This transformation process involves operations for translating and scaling selected areas and for deleting picture parts outside the area. These operations are referred to as windowing and clipping.

3.3.1 VIEWING TRANSFORMATION

As noted, there are two coordinate systems: world and device systems. They are referred to as object space and

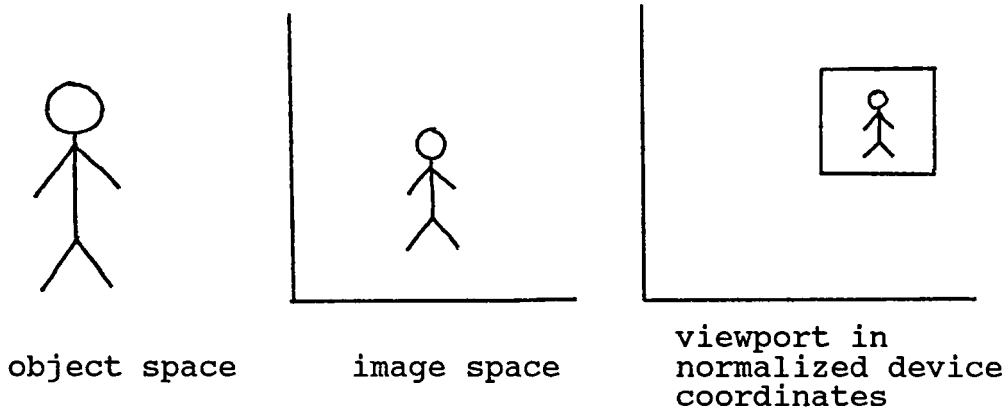


Figure 3.6 Relationship between window and viewport

image space (see figure 3.6). A rectangular area specified in the object space is called a window. The rectangular

area on the screen to which a window is mapped through transformation is called a view port. This mapping is then called a viewing transformation.

By changing the position of the viewport, objects can be displayed at different positions on the output device. Also by varying the size of viewports, the size and proportions of objects can be changed. Zooming in or out effect can be achieved when the size of the window varies.

The viewing transformation involves first translating the window with lower left corner to the origin, second, scaling the window to the size of the viewport and third, moving it to the viewport corner location. Commands to define the window and viewport sizes are :

```
SET_WINDOW(XL,XH, YL,YH)
SET_VIEWPORT(XL,XH, YL,YH)
```

Parameters in each function are used to define the boundaries of the rectangular area - left and right, bottom and top. In this system, both coordinate systems have a range of 0 to 1 (normalized units). Viewing parameters have to be specified right before the segment is created, and they cannot be changed in the middle of the segment.

3.3.2 CLIPPING

The process of clipping is to clip away lines outside the window. An algorithm used here is based on Sutherland and Hugman's method which can clip polygons, lines and characters. The entire figure is clipped against each window boundary in turn. That is each drawing command is clipped against the window starting from left, right, then bottom and finally top. If a vertex moves across the boundary from its preceding position, the intersection is saved as a new command.

There are four possible situations as illustrated in figure 3.7. Any point inside the window is saved. After

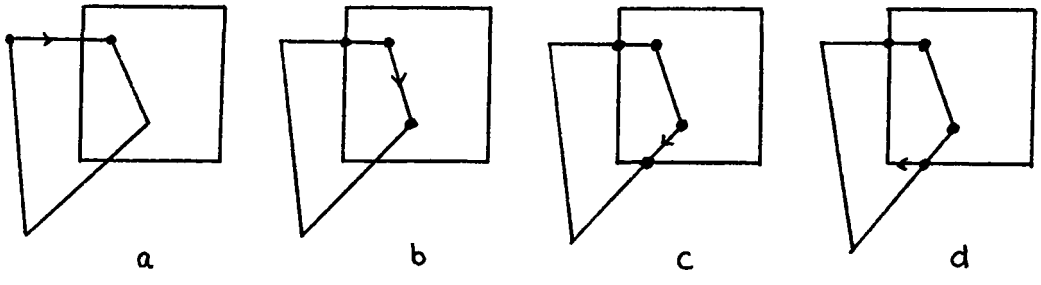


Figure 3.7 Four possible clipping cases

clipping against one boundary, the result is passed to the next boundary check until all four boundaries have been

considered. The last step will enter all commands into the display file.

Extra work is needed to clip the polygon because the number of sides of a polygon will be changed. The result is also limited to 31 sides. Before entering the polygon instructions into the display file, a temporary array is used to save the most recent point that was clipped for each window boundary. After all polygon vertices have been processed, one more clipping is performed on the line joining the first visible point created and the last point. This will then close the polygon (see figure 3.8).

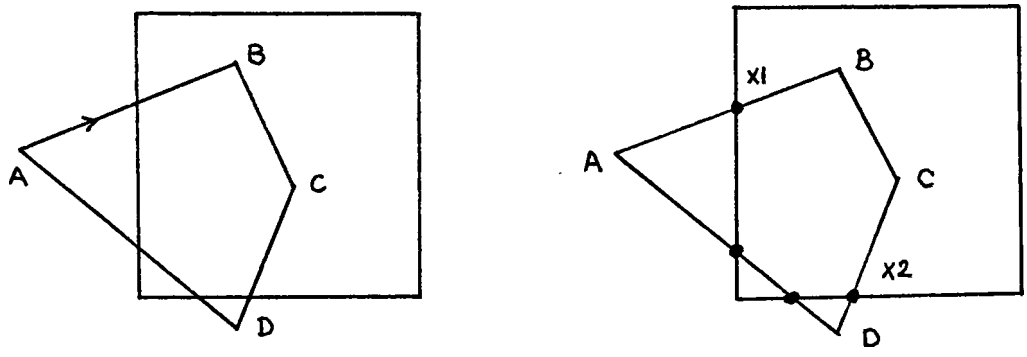


Figure 3.8 Clipping polygon

In figure 3.8, starting from vertex A clockwise, the clipping process will generate the following sequence of

new polygon vertices.

last point	current point	points saved		
A	B	X1	B	
B	C	C		
C	D	X2		
D	A	X3		D A cross two boundaries
X3	A	X4		
A	X1	X1		closing

The clipping algorithm is implemented inside the entering display file module. It is not user accessible. Opcodes greater than 1 (MOVE command) will be inspected first by clipping before entered into display file as shown below.

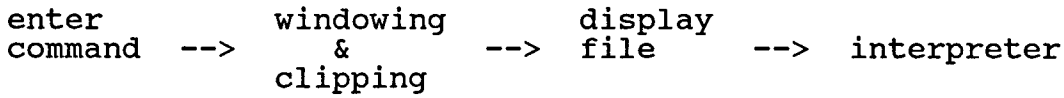


Figure 3.9

3.4 THREE-DIMENSIONAL TRANSFORMATION

Without three-dimensional representation, real live objects cannot be visualized. Hence a graphics system

should be generalized to handle realism of the three-dimensional objects.

In addition to the two-dimensional coordinates, a third z-axis is added to represent the depth of the system. Right handed convention is adopted.

Basic mathematical geometry required for building three-dimensional structures are as follows :

a. point : (X,Y,Z)

b. line : represented in parametric form

$$X = X1 + (X2 - X1)\ u$$

$$Y = Y1 + (Y2 - Y1)\ u$$

$$Z = Z1 + (Z2 - Z1)\ u$$

c. plane : represented by the equation

$$A\ X + B\ Y + C\ Z + D = 0$$

where the triplet (A, B, C) stands for the vector normal to the plane.

d. normal vector : a vector perpendicular to the plane

$$A\ X + B\ Y + C\ Z + D = 0$$

It is called outward normal if

$$A\ X + B\ Y + C\ Z + D > 0$$

else, it is inward normal.

e. angle between two vectors A(XA, YA, ZA) and

B(XB, YB, ZB) :

$$\text{angle} = \cos^{-1} \frac{A \cdot B}{|A|\ |B|}$$

where A . B is the dot product given by

A . B = XA XB + YA YB + ZA ZB

3.4.1 THREE-DIMENSIONAL PRIMITIVES

An extra variable array storing z-coordinates is all that needed in the three-dimensional system. These primitives are :

```
MOVE_ABS_3(X, Y, Z)
MOVE_REL_3(DX,DY,DZ)
LINE_ABS_3(X, Y, Z, COLOR)
LINE_REL_3(DX,DY,DZ, COLOR)
POLYGON_ABS_3(AX,AY,AZ, ACOL, N)
POLYGON_REL_3(AX,AY,AZ, ACOL, N)
```

In order to display a three-dimensional object on a two-dimensional viewing screen, a viewing transformation mapping is needed. In this way, the two-dimensional display file can remain unchanged in structure. Before giving the methods of viewing transformation, the two-dimensional

transformations are expanded here below in homogeneous form.

a. scaling : $S = \begin{matrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$

b. translation : $T = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{matrix}$

c. rotation about each axis through an angle A :

$$Rx = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & cosA & sinA & 0 \\ 0 & -sinA & cosA & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$Ry = \begin{matrix} cosA & 0 & -sinA & 0 \\ 0 & 1 & 0 & 0 \\ sinA & 0 & cosA & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$Rz = \begin{matrix} cosA & sinA & 0 & 0 \\ -sinA & cosA & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

d. rotation about any line L through a clockwise angle A

$$L : \quad X = X1 + A\,u$$
$$Y = Y1 + B\,u$$
$$Z = Z1 + C\,u$$

is :

$$Ra = T\ Rx\ Ry\ Rz\ (Ry)^{-1}\ (Rx)^{-1}\ (T)^{-1}$$

where

T = translation to the point (X1, Y1, Z1)

Rx = rotation at an angle $\sin^{-1} B/(B^2 + C^2)^{1/2}$

Ry = rotation at an angle $\sin^{-1} A/(A^2 + B^2 + C^2)^{1/2}$

Rz = rotation at an angle A

3.4.2 VIEWING TRANSFORMATION

Viewing transformation of three-dimension is the projection onto a two-dimensional view plane. The simplest form is parallel projection. Others commonly used are perspective and isometric projections. Only parallel and perspective projections are implemented in this system.

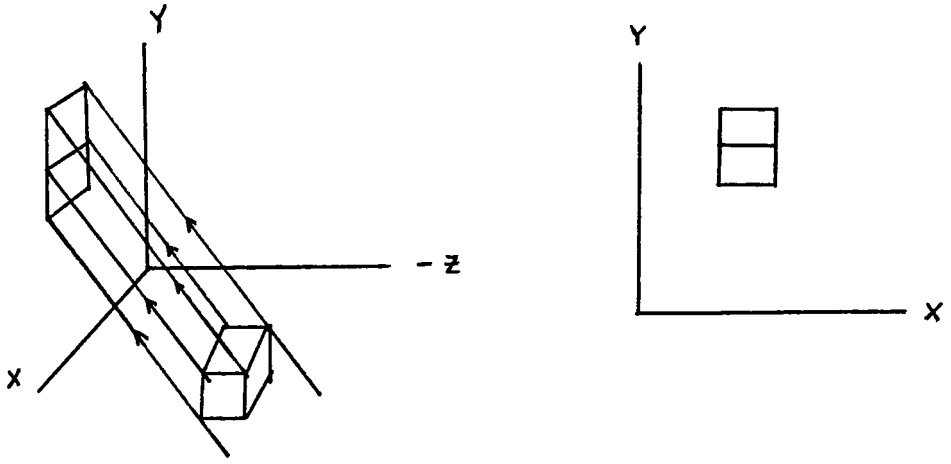


Figure 3.10 Parallel projection onto XY-plane.

A parallel projection is formed by projecting points on the object surface along parallel lines onto a viewing plane (see figure 3.10). If the direction of projection is given by a vector (XP, YP, ZP), the projection onto XY-plane is

$$X2 = X1 - Z1\ (XP/ZP)$$
$$Y2 = Y1 - Z1\ (YP/ZP)$$
$$Z2 = 0$$

Perspective projection changes the sizes of objects so that the further away an object is from the viewer, the smaller it appears. The projection lines instead of being

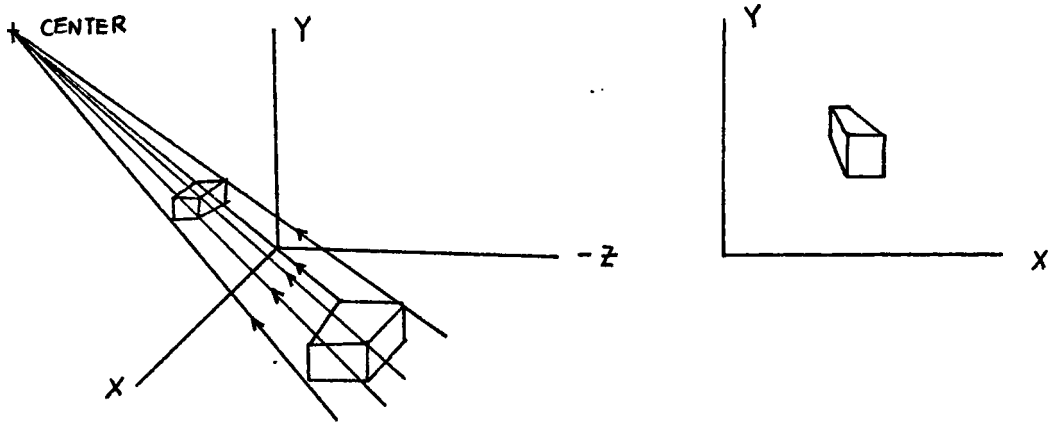


Figure 3.11 Perspective projection

parallel will converge to a single point known as center of projection. The intersections of these lines with the view plane becomes the projected image (see figure 3.11). With

the center of projection at (XC, YC, ZC), the perspective projection of a point (X, Y, Z) on the object onto the XY-plane is

$$X2 = (XC\ Z1 - X1\ ZC)/(Z1 - ZC)$$
$$Y2 = (YC\ Z1 - Y1\ ZC)/(Y1 - ZC)$$
$$Z2 = 0$$

3.4.3 VIEWING PARAMETERS

In this system, the view plane is treated as a variable element just like the film in a camera which can be positioned around in space. There are viewing parameters that can change the projection onto the view plane.

a. reference point (XR,YR,ZR) : is the center of attention.
b. view plane normal (VX,VY,VZ) : is the perpendicular direction to the view plane.
c. view distance : is the distance from the view reference point to the view plane.
d. view up direction (UX,UY,UZ) : is the upward orientation of the view plane.

These parameters allow the user to select how the object is

to be displayed. The commands allowing these settings are

```
SET_VIEW_REFERENCE_POINT(X,Y,Z)
SET_VIEW_PLANE_NORMAL(NX,NY,NZ)
SET_VIEW_DISTANCE(D)
SET_VIEW_UP(UX,UY,UZ)
```

The definitions of projection are specified by

```
SET_PARALLEL(VX,VY,VZ)
SET_PERSPECTIVE(XC,YC,ZC)
```

In order to project the object model onto the view plane correctly, a transformation from the object

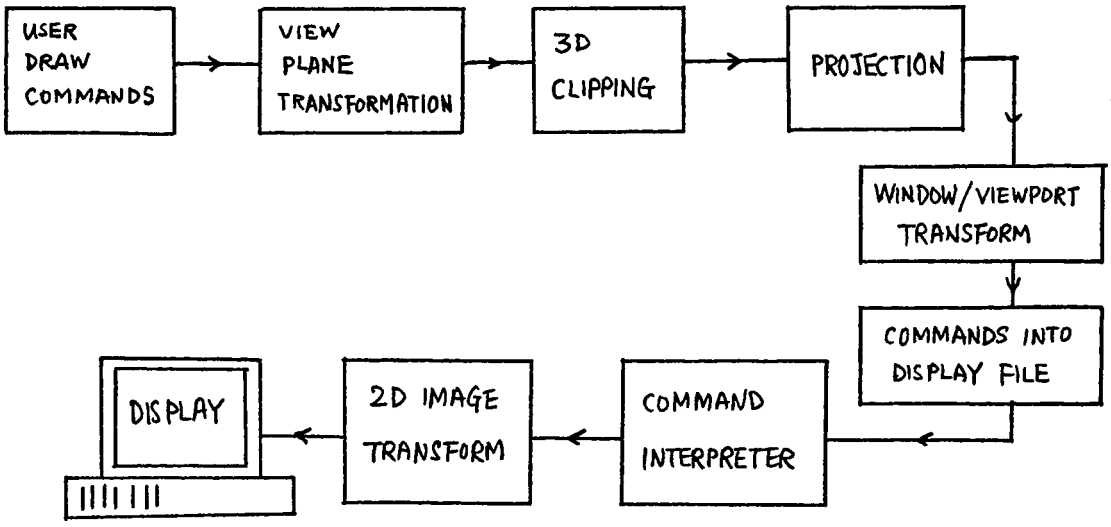


Figure 3.13 System with three-dimensional viewing operations

coordinates to the view plane coordinates is performed first. The process includes translation and three rotations about X-, Y- and Z- axes. Once this is done, the specified projection is carried out. Figure 3.13 illustrates the implementation of the three-dimensional viewing operations.

3.4.4 THREE-DIMENSIONAL CLIPPING

The projected image may produce more detail lines than necessary. For example, objects behind the viewport can appear on the screen. Sometimes objects may exceed the prescribed limits of the viewport specified. These effects can be eliminated by defining a clipping plane to clip away the undesirable portion before projection has taken place.

The testing point in viewer's coordinates is checked as a viewing volume which defines the space bounded by the projecting rays and two clipping planes - front and back (see figure 3.14). In this system, the user has an option to do three-dimensional clipping by simply executing the following commands :

```
SET_VIEW_DEPTH(front_plane_dist, back_plane_dist)
SET_FRONT_PLANE_CLIPPING(on_off)
SET_BACK_PLANE_CLIPPING(on_off)
```

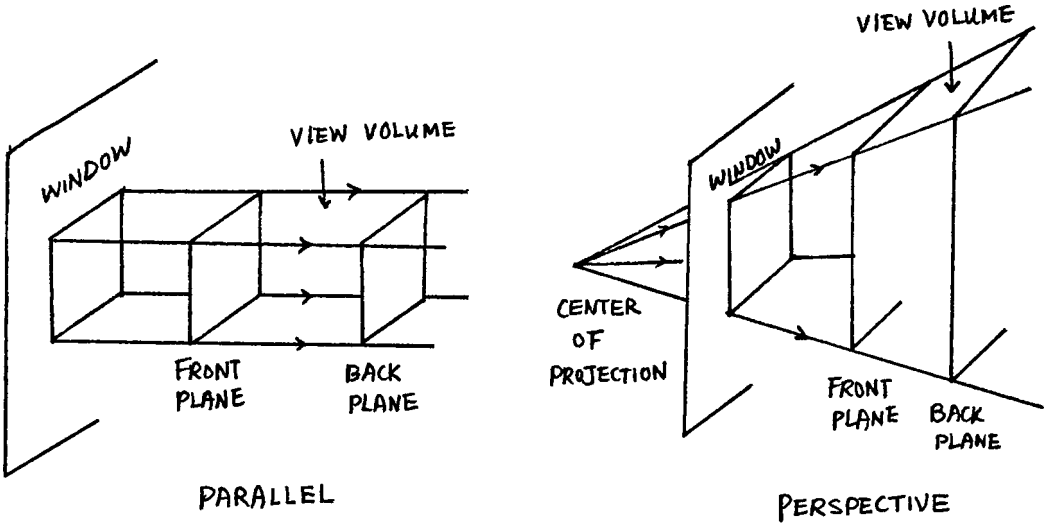


Figure 3.14 Viewing volume in parallel and perspective projections

Here three-dimensional clipping requires two additonal steps, clipping back and front. The whole process is placed before the projection. This eliminates a lot of undesired points and thus saves computation. Given a window size (WXH,WXL,WYH,WYL), the four clipping planes in parallel projection with direction (VXP,VYP,VZP) are :

a. top : $Y = S1\ Z + WYH$

b. bottom : $Y = S2\ Z + WYL$

c. right : $X = S3\ Z + WXH$

d. left : $X = S4\ Z + WXL$

where

$S1 = S2 = VYP/VZP$; $S3 = S4 = VXP/VZP$.

For the perspective case with projection center at (XC,YC,ZC), the corresponding planes take the same form of equations with different plane slopes as

S1 = (YC - WYH)/ZC ; S2 = (YC - WYL)/ZC

S3 = (XC - WXH)/ZC ; S4 = (XC - WXL)/ZC.

For a point (X1,Y1,Z1) to be visible within the viewing volume, it must satisfy the following conditions :

Y1 <= S1 Z1 + WYH	below top plane
Y1 >= S2 Z1 + WYL	above bottom plane
X1 <= S3 Z1 + WXH	left of right plane
X1 >= S4 Z1 + WXL	right of left plane
FRONT_Z <= Z1 <= BACK_Z	between front & back planes.

The clipping of a line is performed by calculating the intersecting point of the line with the plane.

CHAPTER 4

HIDDEN SURFACE AND HIDDEN LINE REMOVAL

A major consideration in the generation of realistic scenes is the identification and removal of the parts of the picture definition, either line or surface, that are not visible from a chosen position. The discussion here is limited to plane polygons and lines. Both involve the determination of depth and visibility of hidden geometry.

4.1 BACK-PLANE REMOVAL

A surface (plane polygon with straight edges) is said to have two faces, front and back. By adopting a convention such that a visible front face is drawn clockwise on the

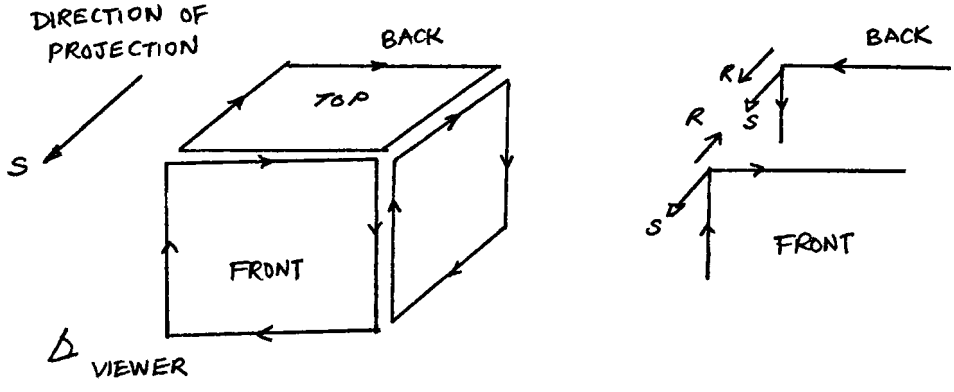


Figure 4.1 Front and back face convention.

viewer side, the cross-product of two adjacent edges and the direction of projection determine the visibility of the face. If R is the resulting cross-product and S the direction of projection, a positive dot-product of R and S will indicate a back face, otherwise a front face to the viewer (see figure 4.1).

Care is taken to ensure the two vectors formed from two adjacent sides meet at a convex vertex and do not coincide with each other. The back-face check algorithm is to be done after the clipping is performed. Front face is saved while back face is discarded. When multiple objects exist together, a front face of one object can be obscured by another front face of another object. To display these multiple front faces properly requires the knowledge of depth information about them. A simple technique called painter's algorithm handles the display of these faces as if they were being painted onto the screen one over the other in the order of their distance from the viewer. Nearer faces are painted on top of more distant ones partially or totally. Therefore before the display, the faces are sorted in decreasing depth order.

In order to facilitate the geometrical sorting, polygon face is decomposed into triangles. Buffers are

needed to keep track of the changes in polygon properties. They are edge colors, fill style, depth and projected coordinates, and drawing commands. The painting function is applied within a given segment. Objects in other segments have no effect on the current one. Therefore, the hidden surface algorithm is done just before the segment is closed.

4.2 DECOMPOSITION OF POLYGON

By starting from the leftmost vertex Pi of a polygon to ensure a convex vertex, and combining the points preceding and following it (Pi-1 and Pi+1), a trial triangle is created. The rest of vertices are tested if any leftmost point lies inside this triangle. If none, Pi-1, Pi and Pi+1 form

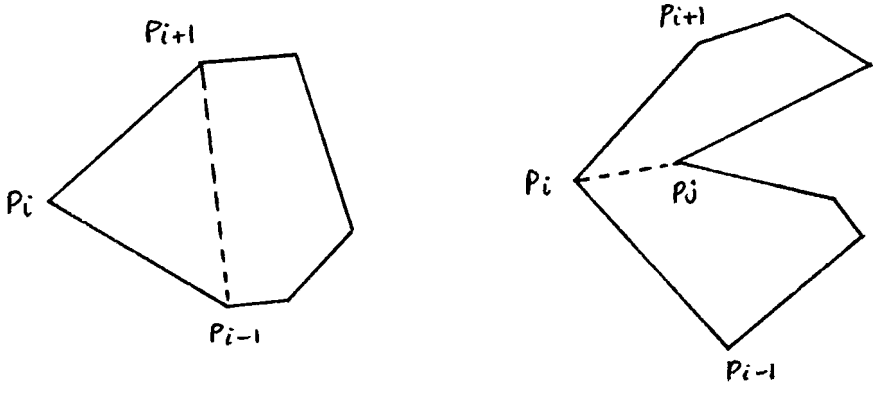


Figure 4.2 Decomposition of polygon.

the triangle. The triangle is copied to a buffer. If there is an interior point Pj, the polygon is split into two subpolygons sharing the splitting side Pi Pj (see figure 4.2). The process is repeated until the polygon buffer is emptied.

4.3 GEOMETRICAL SORTING OF TRIANGLES

The first step in sorting is to establish the depth order of all triangles in a list. From the top, a triangle is compared with all others down in the list according to their depths. By depth, it means the Z coordinate of a projected point common to both triangles. The front triangle Fi having shorter depth is inserted into a list called INFRONT of the back triangle Bj. A back count of Fi is incremented to indicate one more triangle is behind it. Once all have been compared, the order is constructed from back to front.

Initially, the back count list is searched for zero in which the triangle has nothing behind it. These triangles are put into a TO_BE_DONE list. There must be at least 1 such triangle in the set. By taking out a triangle P from the bottom of this stack, the back count for each of its corresponding front triangles is decremented by 1. If one

count reaches zero, this front triangle is added to the TO_BE_DONE list. When all its front triangles have been examined, triangle P is entered into the display file. Figure 4.3 shows an example.

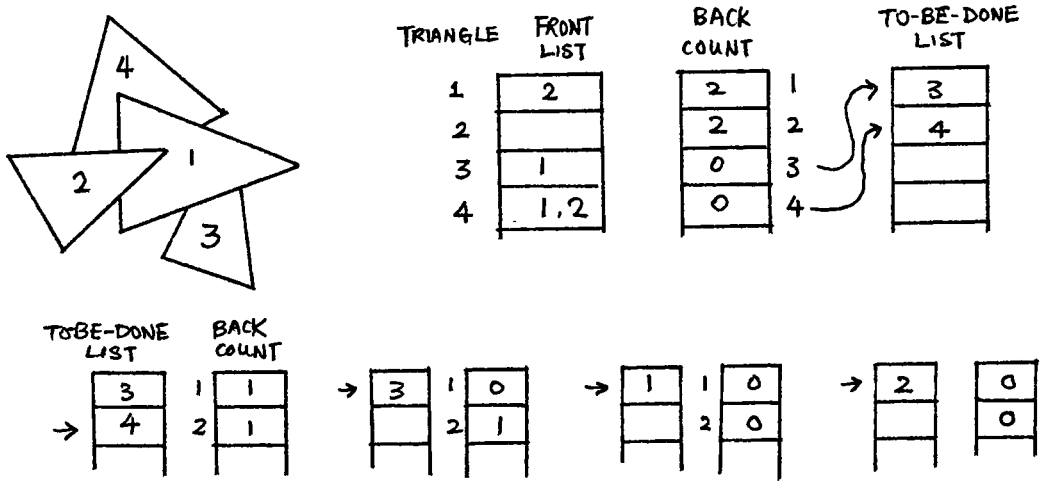


Figure 4.3 Sorting of triangles.

4.4 COMPARING DEPTHS OF TWO TRIANGLES

There are several passes before a depth conclusion can be reached if one test fails after another. The tests for overlapping are listed in order of increasing difficulty.

a. XY MINIMAX TEST

A rectangular box over the vertices of a triangle is compared with the other rectangle over

the other triangle. The rectangle sides are the maximum and minimum of X and Y coordinates of the three vertices. If X minimum of one box is less than the X maximum of the other box, the boxes overlap and need further tests.

b. Z MINIMAX TEST

This test determines which triangle is in front of which. If the smallest Z value for one triangle is larger than the largest Z value for the other triangle, then the first triangle lies in front.

c. OVERLAPPING EDGE TEST

If a projected side of one triangle intersects one of the projected sides of the other triangle, they must overlap. At the point of intersection, their Z values determine the depth order of the two triangles.

d. CONTAINMENT TEST

If three vertices of one triangle lie inside the other triangle, the first one is contained by the second. A point is said to be inside a triangle if it

lies to the right of each triangle side assuming a clockwise drawn triangle. Mathematically, the condition is

$$(X - X1)(Y2 - Y1) - (X2 - X1)(Y - Y1) < 0$$

where (X,Y) is the point and (X1,Y1) and (X2,Y2) represent the end points of a triangle side. Once a point is found inside the triangle, it can be used to establish the depth order. The value (X,Y) of this point is substituted into the plane equation formed by the triangle to solve for Z value. This Z is compared with the Z of the point.

The sequence of these tests is summarized by figure 4.4

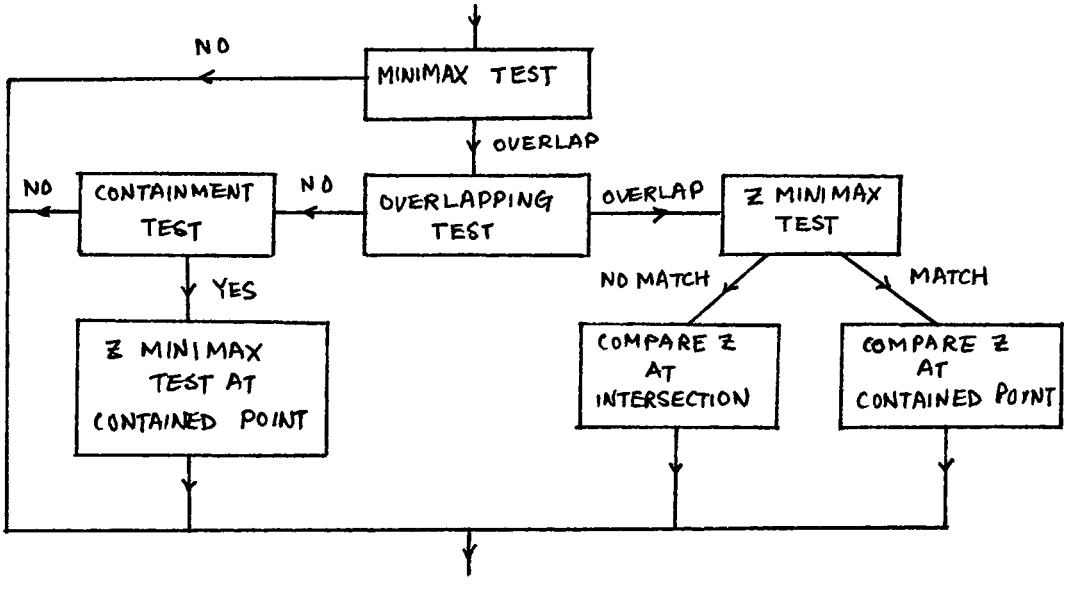


Figure 4.4 Comparison of two triangles.

4.5 HIDDEN LINE REMOVAL

The hidden face algorithm discussed does not work for line oriented (unfilled) polygons. For hidden line removal, the obscured portions of line segments will be replaced by MOVE commands. The approach is to compare each side of a triangle against all of the triangles which lie in front of it to see if it is partially or totally obscured. The three sides are saved initially on a stack. Three cases can occur: 1) a line segment is completely inside the triangle; 2) one end is inside and the other is outside; 3) both ends are outside with middle lying inside the triangle generating two intersecting points. The invisible part of the line will need no further comparison while the visible part needs further comparison against the remainder of the front triangles.

When a line segment is to be drawn from the outside in of a concealing triangle, the LINE command is replaced by a new LINE command drawn to the intersecting point. If it is an inside out line, the LINE command will be replaced first by a MOVE command to the intersecting point and then a LINE command drawn to the outside point. If both end points of the line segment lie outside the triangle yielding two distinct intersecting points P and Q, the new commands are

a LINE command drawn to first point P, a MOVE command to the second point Q, and a LINE command to the last end point down the line. A complete concealed line is replaced by a MOVE command.

When all sides are processed and the stack is empty, the resulting triangle is entered into the display file. The implementation including hidden surface and line removal is illustrated in figure 4.5.

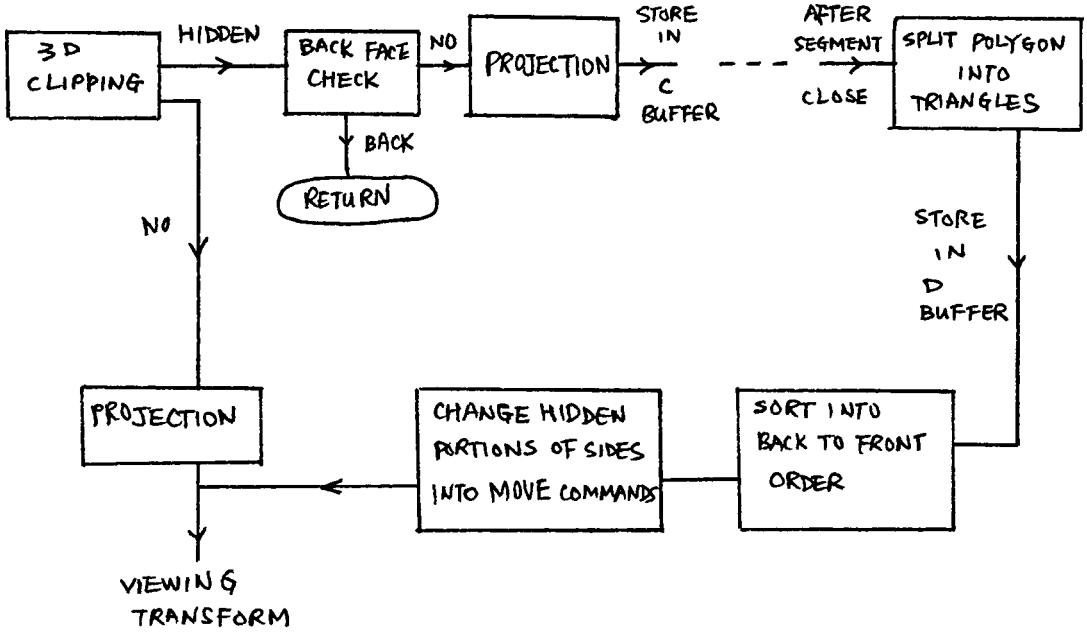


Figure 4.5 Implementation of hidden surface and line removal.

CHAPTER 5

SHADING

5.1 INTRODUCTION

Shading three-dimensional objects can give further realism to the image. A mathematical model is used to describe the light sources which illuminate objects. There are two kinds of reflection that change the shading effect, diffuse reflection and specular reflection. If the light energy emitted from the light source is reflected uniformly in all directions, it is called diffuse reflection. Specular reflection occurs at certain viewing angles and produces a spot of reflected light that is the same color as the incident light. A shiny surface reflects all incident light and has a narrow reflection range. A dull surface has a wider reflection range. The intensity to the viewer decreases as the viewing angle falls off the range (see figure 5.1).

5.2 SHADING PARAMETERS

Physical parameters needed in computing the shade of a surface are background intensity B, surface reflectivity R

which is the fraction of light reflected by the surface, portion of light that goes into specular reflection SP which happens at certain viewing angles and the location of the light source. When given these values, the following are defined:

a. amount of incident light source P

$$P = 1 - B$$

b. amount of specular reflection S

$$S = (1 - B)(1 - R)\ SP$$

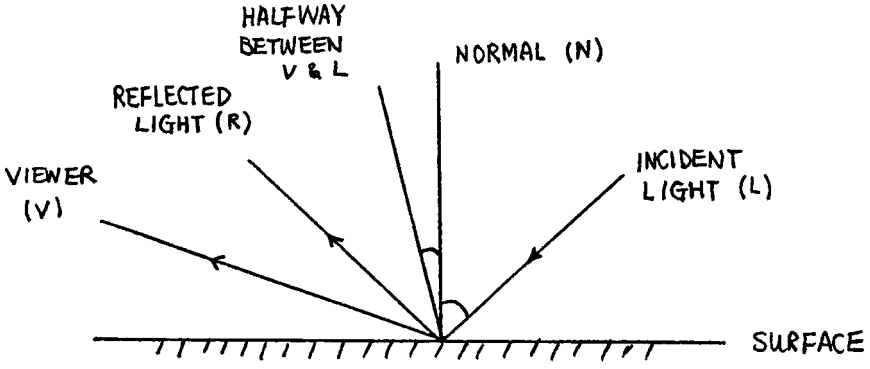


Figure 5.1 Relation between reflections and viewing angle.

Lambert's Law states that the reflection of light from a surface varies as the cosine of the angle between the normal to the surface and the direction of reflected ray, and also that illumination from a point source decreases by the square of the distance between it and the object being illuminated. If L, N, and H are the unit vectors shown in

figure 5.1 with H being half way between the light ray and the viewing direction, by applying the law, a shading model is expressed as follows:

$$SHADE = BR + \frac{P\ R\ cosI + S\ (cosN)\ **\ A}{1 + D}$$

where cosI = L . N, cosN = N . H, D = distance between the object surface and the light source, and the value of SHADE is between 0 and 1. CosN is raised to power A called glossiness in order to produce the specular reflection effect more prominently so that it is 1 within the range and 0 for out of range.

5.3 IMPLEMENTATION

The shading algorithm is inserted at the end of the back-face check procedure. If the face is a back face, shading is not necessary else the surface color is changed by shading. After shading, the face is passed to hidden surface and line removal process. Shading parameters can be set using the following routines:

```
SET_LIGHT(LX,LY,LZ, brightness, background)
SET_OBJECT_SHADE(reflectivity, specular, gloss)
SET_SHADING(on_off)
```

In this system, brightness is between 1 and 2, background 0 to 1, reflectivity 0 to 1, specular reflection 0 to 1, and glossiness 40 to 60. Initially, shading is turned off.

CHAPTER 6

CURVES

The easiest way to approximate a curve on the screen is by a number of small straight-line segments. To draw a curve based on given sample points requires finding a polynomial function that passes these points. A polynomial can be expressed in parametric form as

$$X = Fx(u)$$
$$Y = Fy(u)$$
$$Z = Fz(u)$$

6.1 BLENDING FUNCTION

If the polynomial passes through n sample points, the function can be represented by

$$Fx(u) = SUM(Xi\ Bi(u))$$
$$Fy(u) = SUM(Yi\ Bi(u))$$
$$Fz(u) = SUM(Zi\ Bi(u))$$

where i = 1 .. n. Bi(u) are called blending functions. Each is between 0 and 1 for some u. A function Bi(u) is chosen such that it passes through four given sample points. It is set equal to 1 at some u and 0 for other u. Using the Lagrange polynomial,

$$Bi(u) = \frac{(u+1)\ u\ (u-1)\ \ldots(u-(i-3))(u-(i-1))\ldots(u-(i-2))}{(i-1)(i-2)(i-3)\ldots(1)(-1)\ldots(i-n)}$$

the blending functions over four control points are :

$$B1(u) = \frac{u\ (u-1)\ (u-2)}{-1\ (-2)\ (-3)}$$

$$B2(u) = \frac{(u+1)(u-1)(u-2)}{-1\ (-2)\ (1)}$$

$$B3(u) = \frac{(u+1)\ u\ (u-2)}{(2)\ (1)\ (-1)}$$

$$B4(u) = \frac{(u+1)\ u\ (u-1)}{(3)\ (2)\ (1)}$$

and so the curve in parametric form is

$$X = X1\ B1(u) + X2\ B2(u) + X3\ B3(u) + X4\ B4(u)$$

$$Y = Y1\ B1(u) + Y2\ B2(u) + Y3\ B3(u) + Y4\ B4(u)$$

$$Z = Z1\ B1(u) + Z2\ B2(u) + Z3\ B3(u) + Z4\ B4(u)$$

When given a set of sample points, the curve between two successive points, i and i+1, is approximated by a specified number of line segments with the blending function computed over the four sample points i-1, i, i+1 and i+2. The entire curve is approximated by repeating this process. The larger the number of line segments per section, the smoother the curve drawn on the screen is. As an example, given that X = (1,3,4,7) and Y = (1,5,3,6), to generate three segments over the interval u = (0,1)

requires the calculation of Bi(u) with u = (0, 1/3, 2/3,1). As a result, two more points are generated at u = 1/3 and 2/3.

	u = 0	1/3	2/3	1
X =	3	3.296	3.593	4
Y =	5	4.457	3.654	3

Once these new points are created, they are entered as LINE commands. The routines to generate such a curve are

```
START_CURVE(XA, YA, ZA, COLORA)
CURVE_ABS_3(X,Y,Z,COLOR)
END_CURVE(X,Y,Z,COLOR)
```

XA, YA, ZA, COLORA are 4-element arrays containing first 4 sample points. X, Y, Z, COLOR represent the new sample point for the curve.

These blending functions can be applied to drawing smooth polygons. Because the number of sides will be increased during interpolation, a triangle could be smoothed to 10 small line segments per side while a 15-sided polygon could only be smoothed by 2 segments per side. The routine to do this is

```
SMOOTH_POLY_ABS_3(AX,AY,AZ,ACOL,N).
```

6.2 SPLINE CURVE

The blending functions given above have some drawbacks. First, the sum is equal to 1 only at integer values of u. Hence, flat behavior cannot be obtained when needed. Second, the slopes at the boudnary point between two sections are not the same, therefore creating corner at this point. Finally, control of the curve by a sample point ripples in and out as it moves along in u.

A set of functions that guarantees the curve smoothly follows the control points without discontinuity is called B spline function. The composite function is a polynomial of degree one less than the number of control points used. It is described as

$$P(U) = \sum_{i=0}^{n} P_i \, B_{i,k}(U)$$

where $B_{i,k}$ are the blending functions of degree k - 1. The parameter u varies from 0 to n - k + 2. The functions $B_{i,k}$ are defined recursively as

$$B_{i,1}(U) = 1 \quad \text{if } Ti <= Ti+1$$
$$= 0 \quad \text{otherwise}$$

$$B_{i,k}(U) = \frac{(U - Ti)}{Ti+k-1 - Ti} * B_{i,k-1}(U) + \frac{Ti+k - U}{Ti+k - Ti} * B_{i+1,k-1}(u)$$

If any term becomes 0/0, it is set to 0 by convention. The values of T define the subinterval of U. They are chosen, for i from 0 to n + k, as

$$T_i = 0 \quad \text{if } i < k$$
$$= i - k + 1 \quad \text{if } k \leq i \leq n$$
$$= n - k + 2 \quad \text{if } i > n.$$

For example when k = 3, n = 4 then T values are T0 to T7 of 0, 0, 0, 1, 2, 3, 3, 3. Cubic B spline is obtained with k = 4.

Sharp corners can be created with B-spline functions. It is done by controlling the curve over several identical sample points. Figure 6.2 illustrates the difference between interpolation smoothing and the B-spline smoothing.

by interpolation by B spline

Fig 6.1 Interpolation smoothing and B-spline smoothing

Routines that draw a B-spline curve and a smooth polygon are

```
SET_B_SPLINE(no_of_lines_per_section)
START_B_SPLINE(AX,AY,AZ,COLORS)
END_B_SPLINE(X1,Y1,Z1,X2,Y2,Z2,COLOR)
B_SPLINE_POLY_ABS(AX,AY,AZ,M COLORS)
```

where AX,AY,AX,COLORS are arrays and X1,Y1,Z1,X2,Y2,Z2 are the last two sample points.

CHAPTER 7

STANDARD GRAPHICS ALGORITHMS ANALYSIS

Some common standard graphics algorithms are analyzed to give readers an insight how they work differently from those discussed in Harrington's book. These algorithms are polygon filling methods, clipping methods, hidden surface removal methods and hidden line methods.

7.1 POLYGON FILLING METHODS

The method used by Harrington is called scan-conversion filling. This algorithm is based on the geometrical information in which only vertices are known. In case of a frame buffer is used, the polygon boundaries are represented by pixels. Several methods based on pixels are known. Two of these are flooding filling algorithm and boundary-fill algorithm.

In flood filling, a seed inside the polygon is initiated. Its surrounding 8 pixels are inspected to determine whether the boundry has been reached. The process is repeated until all pixels inside the region have been inspected.

In boundary filling, a starting seed inside the polygon is also required. The method inspects each pixel to the left and right of the seed. When the left- and rightmost boundary pixels are hit, a run or line of pixels is drawn. Then each pixel above and below the line just drawn is examined. Again when the boundary pixels are reached, a line is drawn. The process is continued until all pixels inspected are shown to be boundary pixels.

These two algorithms are useful in interactive sketching and painting packages. Using a graphics tablet or other interactive devices, a user sketches a figure outline, picks an interior point and selects a color or pattern from a color palette. The system then paints the figure interior.

7.2 CLIPPING ALGORITHMS

The clipping of a line segment by a rectangular window can be done by checking the end points of the line against eight regions around the window as shown in figure 7.1. Each region is coded by 4 bits. Each bit position indicates the point is in that relative position. The assignment is bit 1 as left, bit 2 as right, bit 3 as below and bit 4 as top. If a point is within the window, the region code is

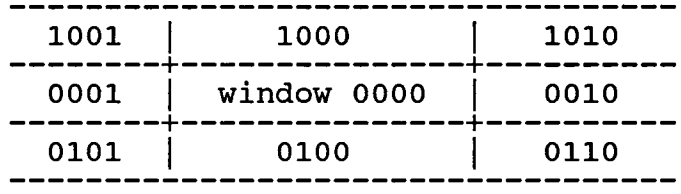


Figure 7.1. Eight regions of a window.

0000. A point in the lower left corner has the region code 0101. Bit values of the point are determined by comparing endpoint coordinate value (X,Y) to the window boudnaries. Bit 1 is set to 1 if X < WXmin (window minimum X value). The other three bit values can be determined in similar fashion. Once the region codes for all endpoints have been established, lines which are completely inside and which are outside the window can be found. Line with both endpoints of 0000 code is completely contained inside the window. If the line has a 1 in the same bit position in the region codes for each endpoint, it is completely outside the window and is rejected. All these operations can be done by XORing the bits. Further computation for intersecting point with the window boundries is needed if the line is partially inside the window or cannot be identified completely. An endpoint is checked against the window boundaries in the order left, right, bottom and top.

This method finds the clipping endpoints very rapidly

but also rejects even more rapidly any line that is clearly invisible. This makes it a very good algorithm for clipping pictures that are much larger than the screen.

Another technique for locating window intersections without the direct computation of the line-equation is a binary search procedure. The goal is to find the visible point on the line segment P0P1 that is farthest from P0. The steps in the process are as follows. Initial testing of lines is carried out using region codes. Undetermined case will be examined by studying the line's midpoint. Each half of the line can be tested for total acceptance or rejection. If half of the line can be accepted or rejected, then the other half is processed in the same way. This continues until an intersection point is reached. If one half of the line cannot be trivially accepted or rejected, each half of it is processed until either the line is totally rejected or a visible section is found.

This method can be hardware implemented because midpoint calculation is equivalent to an addition and a left shift operation, and hence provide fast line clipping than a software type.

7.3 HIDDEN SURFACE REMOVAL

Besides the Painter's algorithm, there are several commonly used approaches.

7.3.1 DEPTH-BUFFER METHOD

Basically, it tests the surface visibility one point at a time. For each pixel position of a projected surface on the view plane, the surface with the smallest Z coordinate at the location is visible. Two buffers are required. One stores the Z values for each X and Y. Another stores the intensity or color for each position. The depth at points over the surface is calculated from the plane equation. Initially, the depth buffer and refresh buffer are set to 1 for maximum normalized depth and background intensity. For each position on each surface, if the calculated depth Z is less than the value of the same position stored in the depth buffer, then new Z value is stored and intensity at this point is put into the buffer. The polygons are processed by scan line one at a time. For any scan line, X coordinates across the line differ by 1 and Y values between line differ 1. If the initial Z is determined by the plane equation $Z = -(AX + BY + D)/C$ at (X,Y) the next iteration at (X+1,Y) is $Z' = Z - A/C$ and

move to next line Z'= Z + B/C at (X, Y-1).

This method requires no geometrical sorting of surfaces. However, it does require a very larger buffer; e.g., a resolution of 1024 by 1024 would require a million.

7.3.2 SCAN LINE METHOD

This method is similar to the scan-line-fill polygon algorithm. The edges of polygons are sorted in order of increasing X. A flag is defined for each surface. It is set on when the scan line is processed from left to right. When the rightmost boundary of the surface is reached, the surface flag is set to off. When the scan line is over the overlapping region of two surfaces, both surface flags are on. For single flag on, no depth calculations are necessary and the intensity information for that surface is entered. When multiple flags are on, depth calculation are performed. The color of the surface with the smallest Z is loaded into the intensity buffer.

7.3.3 QUAD-TREE SUBDIVISION METHOD

This method is applied by successively dividing the view plane into smaller rectangles until in each rectangle,

the projected polygon is found visible or until the screen area is a single pixel. Tests to determine the visibility of a single polygon within a specified rectangle are made by comparing surfaces to the boundary of the rectangle. Four possible catagories can occur :

1. Surrounding polygon which completely encloses the rectangle.
2. Intersecting polygon which intersects the rectangle.
3. Contained polygon which is completely inside the rectangle.
4. Disjoint polygon which is completely outside the rectangle.

No further subdivisions of a specified rectangle are needed if one of the following conditions is true :

1. All polygons are outside the rectange.
   Action : the rectangle is colored to background color.
2. Only one polygon which is completely inside, intersecting or surrounding the rectangle.
   Action : area covering the polygon is colored with the rest in background color.
3. A surrounding polygon is closer to the viewer than all other polygons within the boundary.
   Action : color the area with the color of this

polygon.

4. Subdivision reaches a pixel.

   Action : the Z-coordinates at this point of all visible polygons is computed. The pixel is set to the color of the polygon with the smallest Z value.

If none of these conditions has occurred, the screen area is subdivided into fourths and the process is repeated for each of these quadrants.

7.3.4 OCTREE METHOD

It is used for the viewing volume which is similar to the viewing rectangle in the quad-tree method. The modelling process is to subdivide the viewing cube into eight suboctants. These subcubes are tested against the object to be examined to determine whether a) they lie entirely inside the object - called FULL cubes, b) they lie entirely outside the object - called EMPTY cubes, or c) they lie partially inside and partially outside the object - PARTIAL cubes. Only partial subcubes are further subdivided into sub-subcubes and are tested again. This process of successive subdividing and testing continues until a cube size is reached that is of the desired resolution fineness.

Such a model can be represented by an 8-ary tree structure in which each node represents a cube. The terminal nodes correspond to the FULL or EMPTY subcubes while non-terminal nodes correspond to the PARTIAL subcubes.

The octants when created are labelled as 0 to 7 such that the highest number indicates the octant is the most visible one to the viewer (see figure 7.2). Therefore nothing in octants 0 through 5 can obscure anything in octants 6 and 7. This ordered-priority convention is used as a basis for hidden surface elimination.

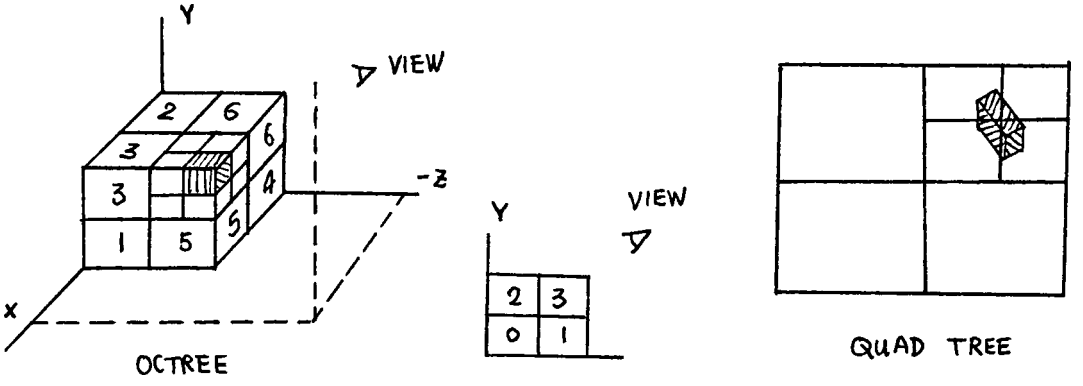


Figure 7.2 Octree method.

The elimination is accomplished first by setting up a corresponding quad-tree that mapps the visible octants. All FULL and PARTIAL octants are projected onto this quad-tree using a recursive front-to-back traversal order (i.e. 7-to-

0 order). Obviously, surfaces of front octants are visible to the viewer while any surfaces toward the rear of the front octants or in the back octants may be hidden by the front surfaces.

If a quad-tree location has no value, this location is then assigned to the color of the projected visible octant being examined. If the traversal order shows an octant to be completely obscured, it needs no further processing and its subsequent subtrees as well. The final result of this quad-tree representation for the visible surfaces is loaded into the frame buffer.

The effectiveness of a hidden-surface method depends on the characteristics of a particular application. As a general rule, the Painter's algorithm is a lightly effective approach for scenes with only a few surfaces. These scenes usually have few overlapping surfaces. The scan-line method also performs well when a scene contains a small number of surfaces. Either the scan-line method or the Painter's algorithm can be used effectively for scenes with several thousand faces. For over a few thousand surfaces, the depth-buffer or octree method performs best. The depth-buffer method has a nearly constant processing time, independent of the number of surfaces in a scene.

However it requires more memory than most methods. Octree or quad-tree method may be preferred for scenes with many surfaces as well. Either one requires no sorting or intersection calculations and occupies less memory. Octree representation can be useful for obtaining cross-sectional slices of solids.

7.4 HIDDEN-LINE REMOVAL METHODS

Some hidden-surface methods can be adapted to hidden-line removal. By using a back-face method, all back surfaces of an object can be identified. Only the boundaries of the visible surfaces are displayed. With the Painter's algorithm surfaces can be painted into the frame buffer so that surface interiors are in the background color, while boundaries are in the foreground color. By processing the surfaces from back to front, hidden lines are erased by the nearer surfaces. A quad-tree subdivision can be adapted by displaying only the boundaries of visible surfaces. Scan-line methods can be used to display ivisible lines by setting points along the scan line that coincide with boudnaries of visible surfaces.

CHAPTER 8

3D SURFACE RECONSTRUCTION FROM PLANAR CONTOURS

8.1 INTRODUCTION

In many scientific and technical applications, a three-dimensional solid must be reconstructed from serial sections, either to aid in the comprehension of the object's structure or to facilitate its automatic manipulation and analysis. The structures, for example, in biological research, medical diagnosis, and automobile design are very often so detailed and the interaction with them so extensive that automation of some kind by computer is almost a necessity.

In order to define a three-dimensional structure effectively to the computer, it has to be reconstructed from a sequence of two dimensional images. One way to obtain a set of two-dimensional images is by means of a laser range finder which measures the three-dimensional coordinates of a point on the surface of the object. The cross-sectional images are then used to reconstruct the three-dimensional surfaces. The construction is done by means of triangulation. There are already different methods

to deal with triangulation. Most of them use heuristic and interpolation approaches. The method to be discussed here will solve the problem by using graph theory and the properties of convex hull.

8.2 AN EXAMPLE OF A MEASURING SYSTEM

The system which is used to collect the three-dimensional coordination data of an object consists of a laser range finder, camera system, a complete controlled table and a microprocessor system. The sensor provides the Z coordinate of a point on the surface of an object as a function of the X coordinate. The laser beam creates a small spot on the surface. The spot is then picked up as an image on each camera detector. The position of the image and the geometrical parameters of the cameras X and Z coordinates of the point can be computed by the microprocessor. The automated vertical motion and the rotation of the platform, by varying the X and Y coordinates, will make the uniform collection of the three-dimensional coordinates possible. The use of two cameras each associated with a microprocessor is to produce a better accuracy. The connection to a computer is done through the second microprocessor. The system is shown in figure 8.1.

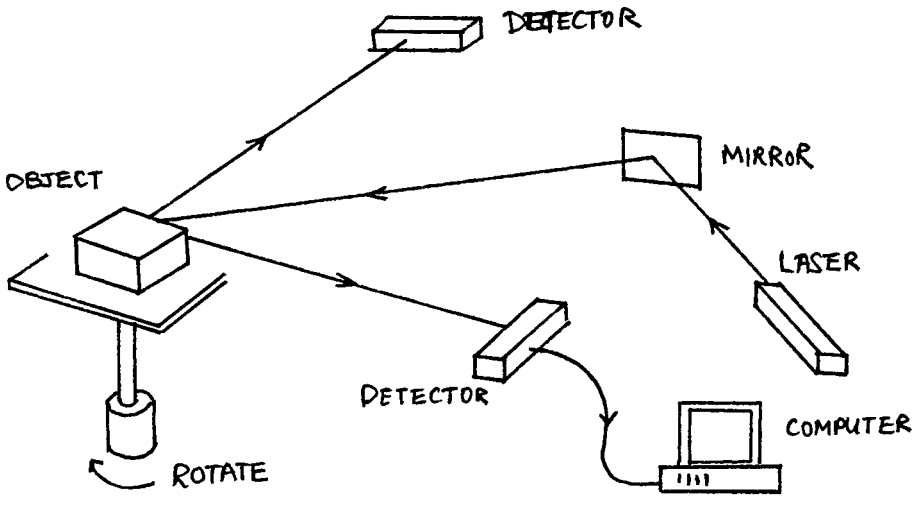


Figure 8.1 A measuring system.

8.3 TWO-DIMENSIONAL REPRESENTATION OF AN OBJECT

Once the three-dimensional coordinate data is collected, the object can be displayed with a finite number of specified paralllel planes. Each of these planes intersect the object with the measured coordinates. Obviously, each set of coordinates is assumed to be a simple closed curve called a contour. The curve segment between two consecutive points is approximated by a straight line. The analysis here is based on the fact that only a single contour per cross-section is allowed (see figure 8.2).

The contour representation of the three-dimensional

object does not have the real human perception of the object's surface. In order to construct the surface over a set of cross-sectional contours, a piecewise planar approximation to the original surface is to be done by means of triangulation between contours as shown in figure 8.2. The result is a closed band of non-intersecting, non-overlapping triangular tiles between two slices.

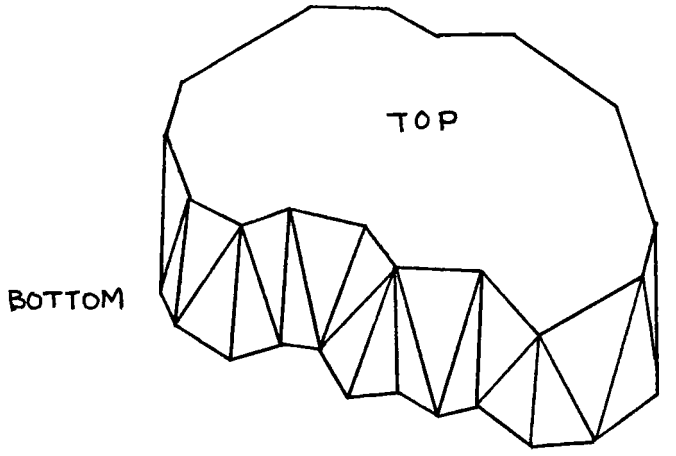


Figure 8.2 Triangulation between planar contours.

8.4 TRIANGULATION METHOD

To create a surface between two successive contours, one would construct tiles between the contours. One way of doing this is by means of triangles such that the vertices are the contour points. The vertices of each triangle are taken - two from one contour and one from the other. The

surface is complete when one edge of the last triangle meets the very first one created. The property of such triangulation is that no two triangles will intersect or overlap each other. In other words, one edge of a triangle is also the edge of the next adjacent one.

8.4.1 CRITERIA OF CONSTRUCTION

There are several ways to construct a triangle. Some popular criteria are :

1. Find a triangle having a minimum surface area between three source points.
2. For a pair of triangle vertices A and B on one contour, find the minimum cost path from A to B through the third point on the other contour.
3. Find the nearest neighboring points that will form a triangle touching the two contours.

Papers concerning these methods can be found in the works of Yuval, Shamos, and Johnson. The next section will introduce another approach to the triangulation.

8.4.2 SHORTER DIAGONAL METHOD

The principle of constructing a triangle from three points taken from two successive contours is by considering

four points - two consecutive points on one contour and two consecutive ones on the other. In other words, they form a quadrangle in space. Let these four points be a and b on the top contour and p and q on the bottom as shown in figure 8.3. Assuming the construction is in anticlockwise sequence and started off from the closest

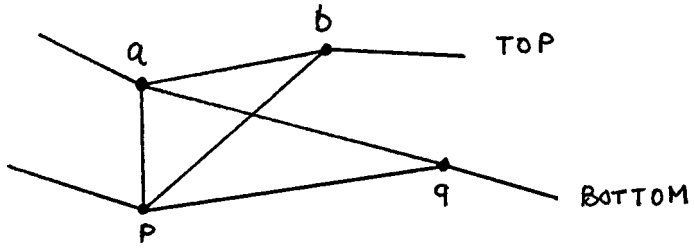


Figure 8.3 Shorter diagonal method.

pair of points a0 and p0, the segment ap is then a side of a triangle. A point either b or q will be chosen to complete such triangle. The shorter diagonal of the two, aq or pb segment is selected as the criterion. The process moves on to the next two pairs, for example in figure 8.3, bc and pq with bp as the base of the next triangle.

The calculation in finding the shorter diagonal is simple. Let the spacing between the contours be in Z-direction and the coordinates of a point p be denoted by (Xp,Yp,Zp). Then the diagonals are

$$(aq)^2 = (Xa - Xq)^2 + (Ya - Yq)^2 + (Za - Zq)^2$$

$$(bp)^2 = (Xb - Xp)^2 + (Yb - Yp)^2 + (Zb - Zp)^2$$

To compare the two diagonals, we take the difference

$$diff = (aq)^2 - (pb)^2$$

$$= (Xa - Xq)^2 - (Xb - Xp)^2 + (Ya - Yq)^2 - (Yb - Yp)^2 + (Za - Zq)^2 - (Zb - Zp)^2$$

From the fact that these points are lying on the two adjacent parallel planes, hence

$$Za - Zq = Zb - Zp$$

and so

$$diff = (Xa - Xq)^2 - (Xb - Xp)^2 + (Ya - Yq)^2 - (Yb - Yp)^2$$

If diff is negative, diagonal aq is shorter, otherwise diagonal pb is shorter if diff is non-zero.

8.4.3 PROPERTIES OF TRIANGULATION

There are several interesting points about this method. First, the computation does not depend on the third coordinate (the spacing between two contours). From the diff expression, spacing is cancelled out. This certainly saves a lot of computer time in three-dimensional system and pattern recognition. Second, it requires no

interpolation between points. The joining of points will automatically take care of the uneven distribution of points on two contours. As an example in figure 8.4, the number of points on the top is larger than that on the bottom. It is clear that the left half points on the top contour will be joined to the left bottom point while the rest of the right half points on the top are joined to the bottom right point. This effect is extremely desirable because it is direct and straight forward. It needs no interpolation, heuristic, or approximation.

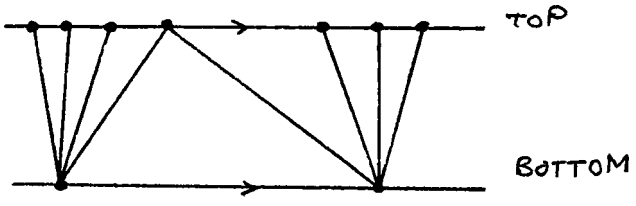


Figure 8.4 Uneven distribution case.

8.44 NUMBER OF TRIANGLES

Number of triangles created can be computed if number of points on each contour is given. If on two successive contours the number of points are A and B, then the number of triangles formed without overlapping is obviously A + B. So for three cross-sections having A, B and C, the total number of triangles will be

$$(A + B) + (B + C).$$

If there are n such cross-sections, and the last one has N points, the total will be

$$\begin{aligned} \text{total} &= (A+B) + (B+C) + (C+D) + \ldots \qquad n-1 \text{ terms} \\ &= A + (2B + 2C + \ldots) + N \\ &= 2(A+B+\ldots+N) - (A+N) \\ &= 2 * \sum_{i=1}^{n}(Xi) - (X1 + Xn) \end{aligned}$$

For an object having 20 cross-sections each with an average of 25 points, the total number of trianlges created will be

$$\begin{aligned} \text{total} &= 2 * (20 * 25) - 2 * 25 \\ &= 950. \end{aligned}$$

Certainly, in order to construct a smoother surface, more cross-sections are needed and hence the number of triangles will increase proportionally. For every extra section of X points, there is an increase of 2X triangles.

8.4.5 ALGORITHM

Let the two successive contours be Ci and Ci+1.

1. Find the rightmost (largest X coordinate) point Pi on Ci and call it P0.
2. Find the rightmost point Qi which is the closest to Pi and call it Q0 on Ci+1.

3. Pick Pi+1 and Qi+1 and find the shorter diagonal between diagonals D1 = Pi Qi+1 and D2 = Pi+1 Qi.
4. If D1 is shorter, form triangle Pi Qi+1 Qi and set

   Qi = Qi+1

   or else form triangle Pi Pi+1 Qi, and set

   Pi = Pi+1.
5. If Pi or Qi, whichever reaches the starting point P0 or Q0 first, create the rest of the triangles by joining the rest of the unlinked points to this P0 or Q0.
6. Otherwise, repeat 3 to 4.

8.4.6 LIMITATION OF ALGORITHM

The shorter diagonal algorithm works correctly if two contours are more or less similar in shape and size to each other. In other words, the profile variation between the two is not very great. One could think of the algorithm as the delinearization of the two closed curves and the joining of the points on two parallel lines based on the said criteria ideally.

There are two extreme cases that will produce an unacceptable effect to the viewer. This happens because the two contours have a large variation. Figure 8.5 illustrates the two cases. In the first one, points Qi, Qi+1, ... on

contour 2 are incorrectly linked to the concavity points of contour 1. The second one is due to large variation in size between the two contours. This problem can be solved. The smaller contour can be imagined as a big one but shrunk. By expanding its size close to the big one such that both are inscribed by the same rectangle, triangulation process can be applied as usual. Case one needs special treatment and is discussed next.

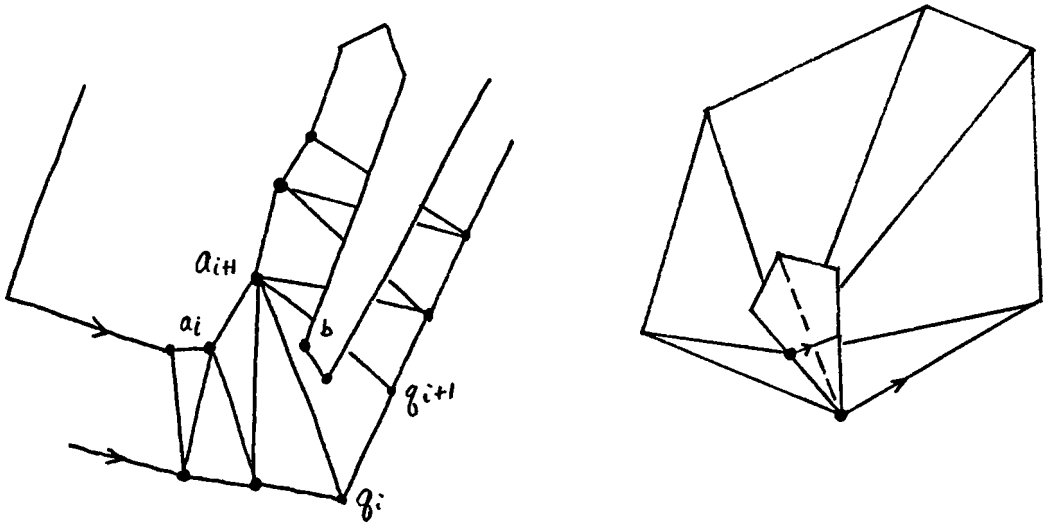


Figure 8.5 Extreme cases.

8.5 GRAPHICAL REPRESENTATION OF TRIANGULATION

Let two successive contours be defined by P and Q and the points on each of them be P0, P1, P2, ... Pn-1 and Q0, Q1, Q2, ... Qn-1 respectively. Each triangle is constructed by a set of three distinct points either of the form (Pi,

Qj, Pi+1) or the form (Pi, Qj, Qj+1) with increasing index orientation. Segment Pi Qj is always on the left of (Pi, Qj, Pi+1) or (Pi, Qi, Qj+1). It is obvious that an acceptable triangulation set is said to be satisfied if the following two conditions are met :

1. Each contour segment appears in exactly one triangle in the set.
2. If Pi Qi appears as the left edge of a triangle t, it will appear as the right edge of the triangle t-1 in the set.

Additional criteria will be used to construct the most appropiate surface as suggested by the shorter diagonal algorithm.

Points on two successive contours can be represented by a two-dimensional directed graph G with row i standing for top contour P and column j the bottom contour Q. The last row and column will repeat the starting points of each contour. The intersection called vertex Vij in the graph maps to the edge joining the two contours. An arc then denotes the orientation of a triangle. So there is a one-to-one correspondence between the triangle set and the graph. As an example, the triangles (5,4,5) and (5,5,6) in figure 8.6 are mapped into the graph as arcs (row5, col4) to (row5, col5) and (row5, col5) to (row6, col5). The

orientation in the graph in from left to right for row and top to bottom for column.

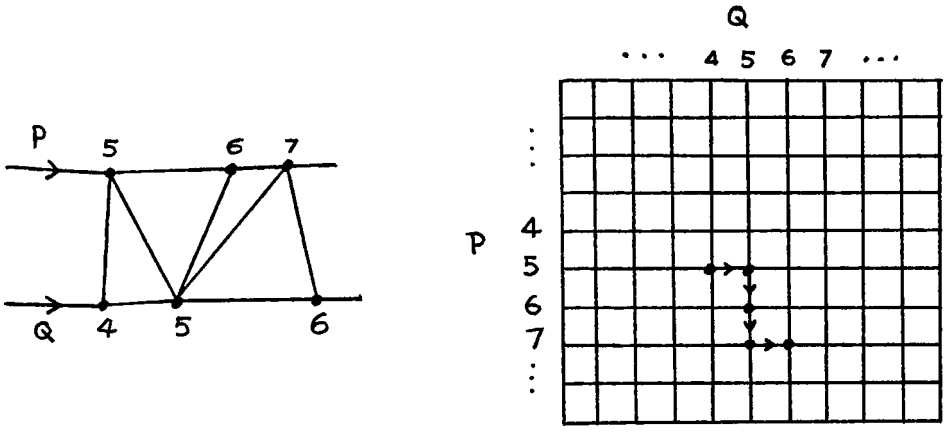


Figure 8.6 Mapping of triangles.

The reconstruction of triangles from the graph is simple just by reading off the marked arcs. Arc I in the graph for instance has end coordinates (row5, col4) and (row5, col5). Therefore the triangle is (P5, Q4, Q5).

Any set of triangles can be described as a subgraph of the entire directed graph G. A subgraph is called an acceptable subgraph which corresponds to an acceptable surface. If S is the acceptable subgraph, the previously stated conditions can be redefined as

1. For every row i, i = 0, 1, ... m-1, 0, there is exactly one vertical arc Pi Pi+1 in S between the

rows i and i+1 ;

and for every column j, j = 0, 1, ... n-1, 0, there is also exactly one horizontal arc Qj Qj+1.

2. If a vertex is marked, it must be shared by two links (incoming and outgoing).

A directed graph is weakly connected if and only if it is connected with arcs of no unique direction. There are two Lemmas related to this directed graph.

Lemma 1. An acceptable subgraph S of G is weakly connected.

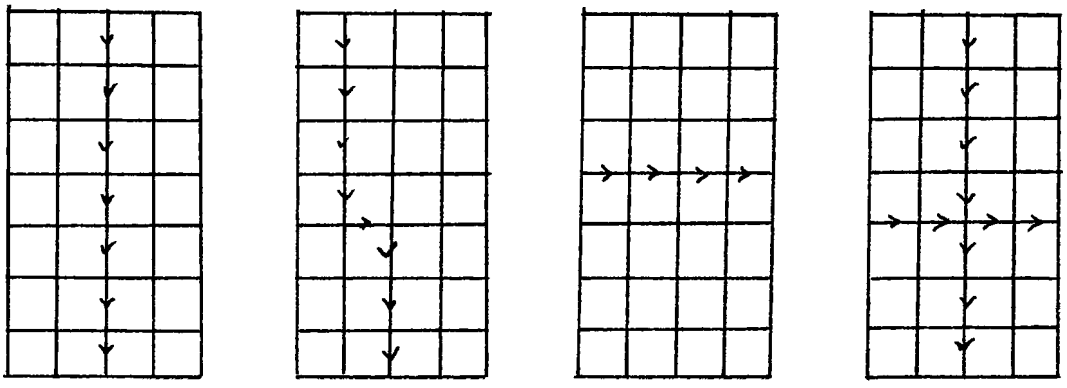


Figure 8.7 Possible subgraphs.

Lemma 2. If vertex of a subgraph S such that the number of incoming arcs plus the number of outgoing arcs is greater than or equal to 3, then at vertex Vij, the number of

incoming arcs = number of outgoing arcs = 2, and for every other vertex Vst of S the number of incoming arcs = number of outgoing arcs = 1.

Figure 8.7 illustrates some possible subgraphs.

Proof of Lemma 1 :

Let S be an acceptable subgraph which contains arcs and vertices at least two weak components, one S0, the other S - S0. For simplicity there is at least one horizontal arc in S but not S0. Therefore, we claim that Vij is in S0 and S0 does not contain a horizontal arc between columns j-1 and j. Let e denote an element of a set and /e denote not belonging to the set.

arc V_{kj} $V_{k+1\ j}$ e S0 for k = 0, 1,.. m-1, 0

but V_{ij} e S

therefore

incoming(Vij) > 0

with arc $V_{i\ j-1}$ V_{ij} /e S0 by assumption

therefore

$V_{i-1\ j}$ V_{ij} e S0

and $V_{i-1\ j}$ e S0

By similar argument,

$$\text{incoming}(V_{i-1\ j}) > 0$$

and arc $V_{i-1\ j-1}$ $V_{i-1\ j}$ /e S

so $V_{i-2\ j}$ $V_{i-1\ j}$ e S0

Continuously proving in this fashion, for all k = 0, 1, ... i, ... m-1, 0, we will have

arc $V_{k\ j}$ $V_{k+1\ j}$ e S0

As we claim earlier S has a horizontal arc between any two adjacent columns for some k, that is

arc $V_{k\ j-1}$ $V_{k\ j}$ e S

but not S0. This implies

Vkj e S, and Vkj e S0.

By the fact that a vertex Vkj in S0 indicates its incoming arc must be in S0. Hence

$V_{k\ j-1}$ $V_{k\ j}$ e S0

Early we assumed S0 contains no arc between column j-1 and j, it follows by contradiction that

$$S = S0$$

and so S is weakly connected.

Proof of Lemma 2 :

Let the three arcs of Vij be $V_{i\,j-1}\ V_{ij}$, $V_{ij}\ V_{i\,j+1}$, and $V_{i-1\,j}\ V_{ij}$. Assume that S does not contain any other horizontal arcs between columns j-1 and j, and between columns j and j+1. Using the proof in lemma 1, then all vertical arcs

$$V_{kj}\ V_{k+1\,j} \quad e \quad S \quad \text{for } k = 0, 1, \ldots i, m-1, 0$$

Therefore,

arcs $V_{i-1\,j}\ V_{ij}$ e S

$V_{ij}\ V_{i+1\,j}$ e S

and by condition 1, S then contains no other vertical arcs between rows i-1 and i and between the rows i and i+1. Since S is weakly connected,

$V_{ik}\ V_{i\,k+1}$ e S between any two adjacent columns with k = 0, 1, ..., j, ... n-1, 0

therefore $V_{i\,j-1}\ V_{ij}$, $V_{ij}\ V_{i\,j+1}$, $V_{i-1\,j}\ V_{ij}$, $V_{ij}\ V_{i+1,j}$

all four are connected through vertex Vij. In other terms,

any other vertex Vik for some k, can have only arcs $V_{i\,k-1}$ V_{ik}, V_{ik} $V_{i\,k+1}$ and the vertex Vkj with arcs $V_{k-1\,j}$ V_{kj}, V_{kj} $V_{k+1\,j}$.

Obviously, triangles created between two planar contours will never cross each other. In terms of the arcs in the directed graph, we claim that the graph is traversed by a closed walk in which every arc occurs exactly once. That is, for every vertex Vij in S the number of incoming arcs = the number of outcoming arcs. The above two lemmas then lead to the following theorem.

Theorem :

A subgraph S of G represents an acceptable surface if and only if S contains exactly one horizontal arc between any two adjacent columns, and exactly one vertical arc between any two adjacent rows and it is a closed graph.

If IN = number of incoming arcs and OUT for outgoing arcs at vertex Vij, there are two acceptable cases :

1) IN = OUT = 1 for every vertex Vij.
2) IN = OUT = 2 for one vertex Vab and
   IN = OUT = 1 for every other vertex Vij,
   i <> a, j <> b in S.

These two are illustrated in figure 8.8.

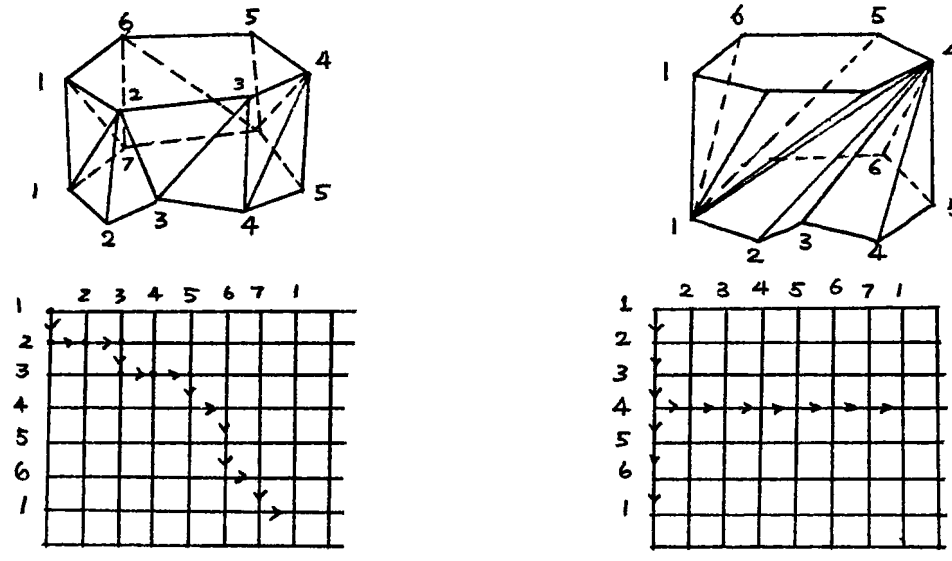


Figure 8.8 Two acceptable subgraphs.

As there are many acceptable surfaces (m+n)!/(m! n!) where m, n are the number of vertices of two planar contours, a criterion must be chosen to obtain a good acceptable surface. In this discussion, only case 1 is an acceptable construction. Case 2 happens as discussed in section 8.4.6. The shorter diagonal method presented earlier is used to find the appropiate path in the graph.

After the path is created, the directed graph theorem is then used to detect disqualified triangulation between two contours. A good triangulation is one whose every vertex in the graph is visited no more than once.

8.6 CONVERSION OF CONCAVITY TO CONVEXITY

One way of keeping case 2 from happening when applying the shorter diagonal method is to detect the concavity of a contour or polygon. If with two polygons, one has extreme concavity and the other has convexity over the same vicinity of the first one, case 2 connection will result; so such concavity has to be dealt with locally.

The convex hull of a finite planar set is defined as the minimum area convex set containing the original set. The two properties of a convex polygon are given here to help the discussion that follows.

a. All interior angles are less than 180 degrees.
b. If the polygon is drawn counter-clockwise, each vertex must be lying on the left side of a preceding edge.

Property b is used to detect a special situation during the conversion process.

Two vertices of either convex or concave polygon P are said to be visible if the line segment joining them lies in P. P is said to be weakly visible from edge UV if for each vertex Z belonging to P, there exists a vertex W belonging to UV such that Z and W are visible. A polygon is edge-

visible if there exists at least one edge of P from which P is weakly visible.

8.6.1 CONVEX HULL ALGORITHM

The main function of this algorithm is to discard a concave point j. Its neighboring points j-1 and j+1 will then form a new edge. The vertices are processed counter-clockwise starting with the rightmost vertex - one with the largest X value. By applying the property a, three consecutive points i, i+1, i+2 are tested for the condition

$$S1 = (X_{i+1} - X_i)(Y_{i+2} - Y_{i+1}) - (Y_{i+1} - Y_i)(X_{i+2} - X_{i+1}) > 0$$

to be satisfied. If S1 < 0, that means vertex i+1 is a concave point lying to the left of the line segment joining i and i+2 and is discarded. A new edge joining i and i+2 is created. In order to ensure that vertex i is still convex with respect to vertices i-1 and i+2, the process is backtracked to i-1 (see figure 8.9). If the condition is satisfied, vertex i+1 is said to be on the right of the diagonal joining vertices i and i+2. The tested points are said to be temporarily in proper order. A further test on point i+2 is required to maintain the property b.

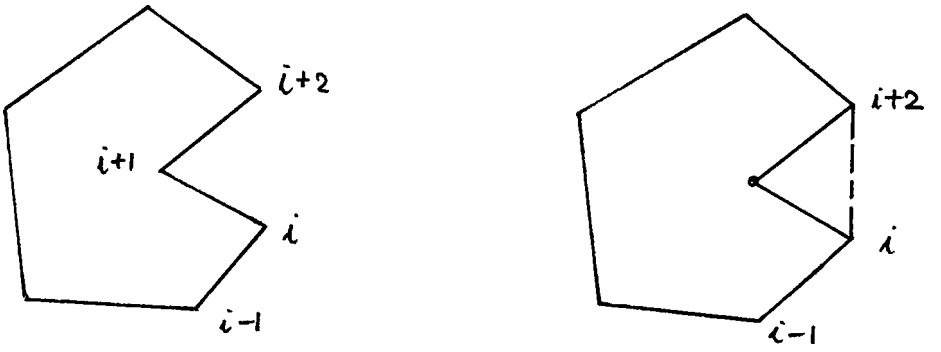


Figure 8.9 Eliminating a concave point.

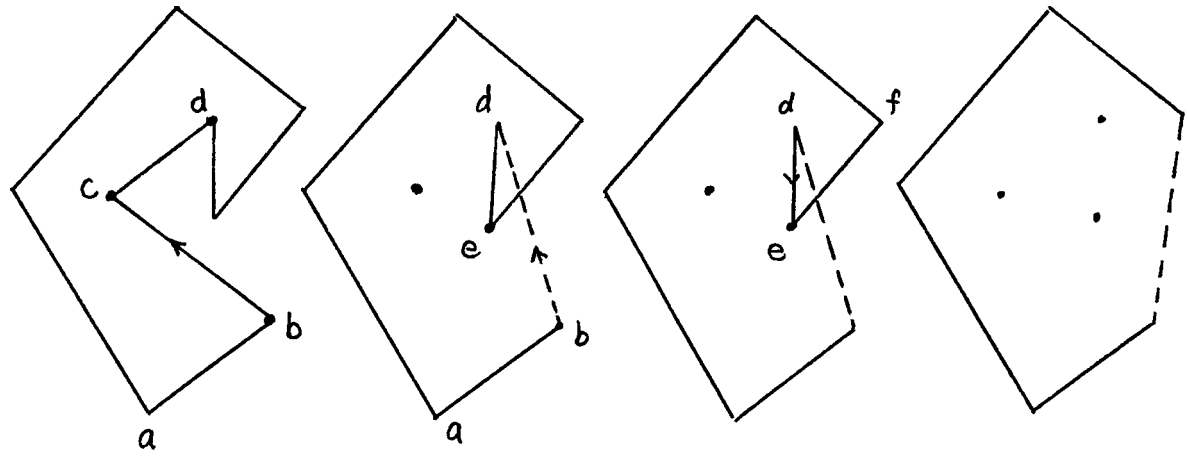
A new edge which results from eliminating a concave point may cross an existing edge. That is whenever a reflex vertex is discarded, a weakly external visible polygon is replaced by another weakly external visible polygon with one less vertex as shown in figure 8.10. The second test condition remains the same, but with changes in vertices i, i+2 and i-1.

$$S2 = (X_{i+2} - X_i)(Y_{i-1} - Y_{i+2}) - (Y_{i+2} - Y_i)(X_{i-1} - X_{i+2})$$

$$> 0$$

If the condition S2 is not satisfied, the polygon is weakly visible with i, i+1 as the visible edge. Both points i and i+1 are discarded from consideration and the process is repeated again by backtracking. When three points survive

from the tests S1 and S2, the index is incremented by one until point i+1 reaches the beginning rightmost vertex.



a.	b.	c.	d.
test bcd result drop c	test bde result OK test bea result OK	test def result OK test dfb result drop de	

Figure 8.10 Crossover condition.

8.6.2 BACKTRACKING

Backtracking can be done without building a recursive function. A simple link list is used to save the point i at i+1 when points i, i+1 and i+2 satisfy the conditions S1

and S2. So at i+1, it remembers the previous qualified point i. If three points do not meet the criteria, backtracking one point will become

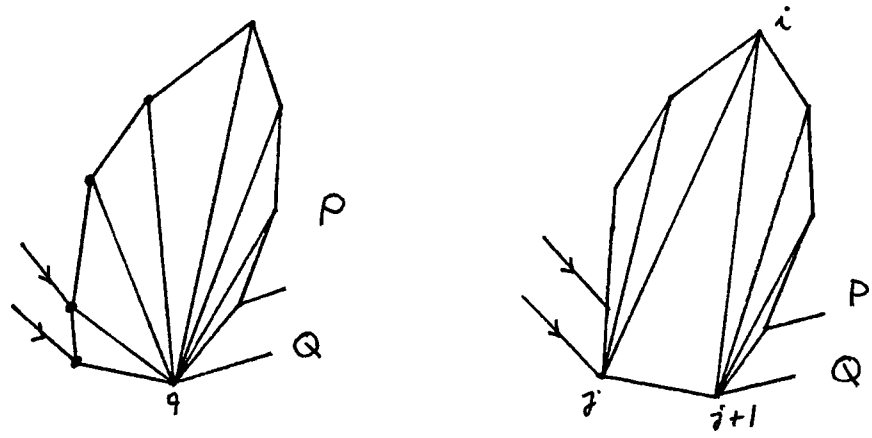
```
TEMP = i
i = LIST(i)
i2 = i1
i1 = TEMP .
```

At the end of process, the list will give all the convex points.

8.6.3 RECONSTRUCTION FROM LOCAL CONCAVITY

It is the concavity that gives rise to an unacceptable triangulation. When there exists a concave region on one polygon P with two end convex points a and b embracing it, a search for two corresponding end convex points p and q on the other polygon Q in the neighborhood of the concavity is carried out. Such a local situation is shown in figure 8.11. The shorter diagonal method applied to this local region will not produce a good surface. All points on polygon P will converge to the point q on polygon Q. In order to produce a better approximation of the surface when one region has more points than the other, the points must be distributed evenly over both regions. Therefore a point

i on the larger region is compared with two points j, j+1 on the smaller region. If it is nearer to j+1, two lines



a. shorter diagonal b. even distribution

Figure 8.11 Local surface reconstruction.

are formed i j and i j+1; that is two triangles are created. Otherwise only line i j is joined forming one triangle. The process continues to the next point i+1 until all points in this local area are joined.

8.7 TRIANGULATION IMPLEMENTATION

For a given set of polygons, two will be processed at a time. Each is converted to a convex hull with the concave points marked if they exist. The two convex hulls are joined together using the shorter diagonal algorithm.

At the same time, the directed graph of this connection is constructed. The directed graph is represented by a two dimensional matrix with rows representing the top polygon and columns representing the bottom one. A check for concavity on each polygon is made. If it exists, the links on the directed graph over this region must be broken. Local surface reconstruction is performed by even distribution method and the broken arcs are reconnected. In the end, the triangles are recreated from the directed graph by following through the arcs (figure 8.12).

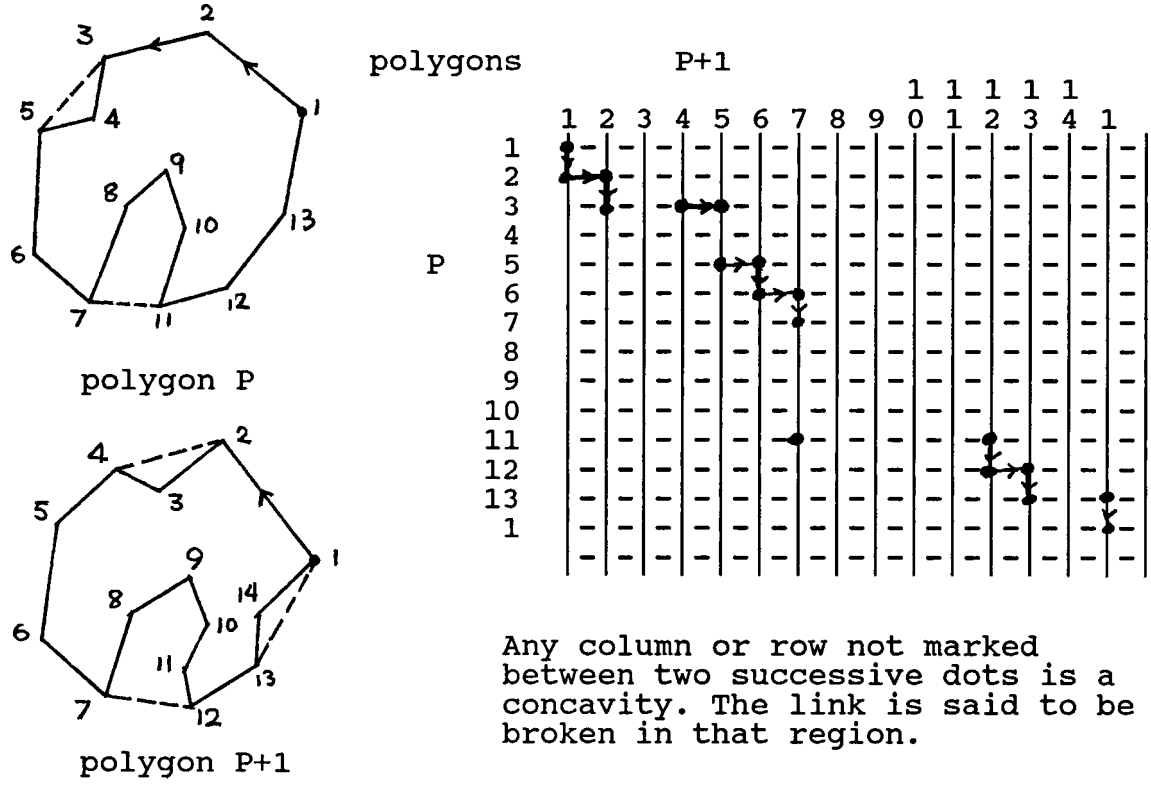


Figure 8.12 Triangulation between 2 polygons.

Another variation to this implementation is first to join the two polygons by the shorter diagonal method regardless of the existence of concavity. Simultaneously, the directed graph is constructed. Second, the graph is examined to detect unacceptable triangulation where IN = OUT = 2 occurs. If such vertex exists on the graph, the two polygons are converted to convex hulls, and the above scenerio is applied. Otherwise, the process is moved onto the next two polygons.

The triangulation result is output to a formatted file so that it can be displayed by the graphics system.

8.8 EXPERIMENTATION USING THE GRAPHICS PACKAGE

The experimentation here is the three-dimensional surface reconstruction of the cross-section of a human heart. Coordinate points were obtained from a sequence of PET images. Limitations of the implemented graphics system do exist. The most important drawback of the system is its number of allowable sides for a polygon. It limits the user to having 31 sides. The experiment has polygons over 31 sides. This problem is solved by making use of the existing routine which splits the polygon into triangles when dealing with hidden surfaces and lines. Only the display of

the topmost and the very bottom slices will use such transformation. The surfaces between layers are represented by triangular tiles which are the result of the triangulation process discussed in previous sections. Each layer is colored differently in order to show concavity clearly. Another limitation of the system is its view plane being fixed to the X-Y plane where projected image is displayed. That is the object must be located in the region bounded by X >= 0, Y >= 0 and Z <= 0. If the viewer wants the back view of the object using the view point transformation, the result is a blank screen. To move the view point requires readjustments of the center of projection, the view plane normal and the view up direction, making the user input difficult. To this end, the experiment is performed by allowing the user to rotate about three axes parallel to X-, Y- and Z-axis and through the center of gravity of the object in any order and for whatever number of times. Windowing function is also implemented in the experiment to allow the user to zoom a particular region of the image on the screen. Results are shown in appendix B.

CHAPTER 9

CORRECTIONS OF PROGRAMMING ERRORS

The implemented graphics system is based on Steven Harrington's book - Computer Graphics, A Programming Approach, 1983 edition. Errors are found during testing. They are either typographical or logical. The corrected versions of these routines are listed with comments. The system is implemented in VAX FORTRAN 77.

```
* ALGOR 3.11
  SUBROUTINE        INCLUDE(END_EDGE,LAST_EDGE,SCAN)

  INTEGER*2         END_EDGE,LAST_EDGE,SCAN
  REAL              YMAX(31),XA(31),DX(31)
  COMMON            /C_YMAX/YMAX,/C_XA/XA,/C_DX/DX
  COMMON            /C_SCAN_DEC/SCAN_DECREMENT

  DO WHILE ((END_EDGE.LE.LAST_EDGE).AND.(YMAX(END_EDGE)
           .GE.SCAN))

     XA(END_EDGE)=XA(END_EDGE) + DX(END_EDGE)*
           (SCAN_DECREMENT + SCAN - YMAX(END_EDGE))

     DX(END_EDGE) = DX(END_EDGE)*(-1.0 * SCAN_DECREMENT)
     END_EDGE = END_EDGE + 1
  ENDDO

 RETURN
 END

 Comment : typo error.
```

```
* ALGOR 6.8
* routine for clipping against the lower boundary

SUBROUTINE          CLIP_BOTTOM_M912(OP,X,Y,COLOR)

INTEGER*2 OP,COLOR,COLS(4)
REAL                X,Y,XB
REAL                WXL,WYL,WXH,WYH
REAL                XS(4),YS(4)

COMMON              /W_CUR/WXL,WYL,WXH,WYH
COMMON              /L_PTS/XS,YS, /L_COLS/COLS

IF ((Y.GE.WYL).AND.(YS(3).LT.WYL)) THEN
  XB = (X-XS(3)) * (WYL-Y)/(Y-YS(3)) + X
*          adjust the x-coordinate
  IF (XB.LT.WXL) XB = WXL
  IF (XB.GT.WXH) XB = WXH
  CALL CLIP_TOP(1,XB,WYL,COLOR)
ENDIF
IF ((Y.LE.WYL).AND.(YS(3).GT.WYL)) THEN
  XB = (X-XS(3)) * (WYL-Y)/(Y-YS(3)) + X
*          adjust the x-coordinate
  IF (XB.LT.WXL) XB = WXL
  IF (XB.GT.WXH) XB = WXH
  IF (OP.LT.32) THEN
       CALL CLIP_TOP(OP,XB,WYL,COLOR)
  ELSE
       CALL CLIP_TOP(1, XB,WYL,COLOR)
  ENDIF
 ENDIF

XS(3) = X
YS(3) = Y
COLS(3) = COLOR

IF (Y.GE.WYL) THEN
   CALL CLIP_TOP(OP,X,Y,COLOR)
ENDIF

RETURN
END
```

Comment : The correction is to adjust the computed value of X if it is still beyond the window boundary.

```
* ALGOR 6.9
* routine for clipping against the upper boundary

SUBROUTINE          CLIP_TOP_M913(OP,X,Y,COLOR)

INTEGER*2 OP,COLOR,COLS(4)
REAL                X,Y,XTOP
REAL                WXL,WYL,WXH,WYH
REAL                XS(4),YS(4)

COMMON              /W_CUR/WXL,WYL,WXH,WYH
COMMON              /L_PTS/XS,YS, /L_COLS/COLS


 IF ((Y.LE.WYH).AND.(YS(4).GT.WYH)) THEN
      XTOP = (X-XS(4)) * (WYH - Y)/(Y - YS(4)) + X
 *         adjust the x-coordinate
      IF (XTOP.LT.WXL) XTOP = WXL
      IF (XTOP.GT.WXH) XTOP = WXH
      CALL SAVE_CLIPPED_POINT(1,XTOP,WYH,COLOR)
 ENDIF

 IF ((Y.GE.WYH).AND.(YS(4).LT.WYH)) THEN
      XTOP = (X-XS(4)) * (WYH - Y)/(Y - YS(4)) + X
 *         adjust the x-coordinate
      IF (XTOP.LT.WXL) XTOP = WXL
      IF (XTOP.GT.WXH) XTOP = WXH
      IF (OP.LT.32) THEN
         CALL SAVE_CLIPPED_POINT(OP,XTOP,WYH,COLOR)
      ELSE
         CALL SAVE_CLIPPED_POINT(1, XTOP,WYH,COLOR)
      ENDIF
 ENDIF

  XS(4) = X
  YS(4) = Y
  COLS(4) = COLOR

  IF (Y.LE.WYH) THEN
     CALL SAVE_CLIPPED_POINT(OP,X,Y,COLOR)
 ENDIF

 RETURN
 END
```

Comment : Again the computed value for X should be adjusted if X exceeds the window boundary.

```
* ALGOR 8.10
SUBROUTINE        ROTATE_Y_3(S,C)

INTEGER*2         I
REAL              S,C, TMP
REAL              TMATRIX(4,3)
COMMON            /C_TMX/TMATRIX

DO I = 1, 4
  TMP = TMATRIX(I,1)*C + TMATRIX(I,3)*S
  TMATRIX(I,3) = -TMATRIX(I,1)*S + TMATRIX(I,3)*C
  TMATRIX(I,1) = TMP
ENDDO

RETURN
END
```

```
* ALGOR 8.20
SUBROUTINE        MAKE_VPLANE_TRANSFORM

REAL              XR,YR,ZR
REAL              DXN,DYN,DZN
REAL              DXUP,DYUP,DZUP
REAL              TMATRIX(4,3)
LOGICAL           PERSPECTIVE_FLAG
REAL              VIEW_DISTANCE

REAL              V,XUP_VP, YUP_VP, RUP, ROUNDOFF

COMMON            /C_REF/XR,YR,ZR
COMMON            /C_NORM/DXN,DYN,DZN
COMMON            /C_VUP/DXUP,DYUP,DZUP
COMMON            /C_TMX/TMATRIX
COMMON            /C_PPFLAG/PERSPECTIVE_FLAG
COMMON            /C_VDIST/VIEW_DISTANCE

ROUNDOFF = 0.001

* Start with the identity matrix
CALL NEW_TRANSFORMATION_3

* Translate so that view plane center is new origin
CALL TRANSLATE_3(-(XR + DXN * VIEW_DISTANCE),
*      -(YR + DYN * VIEW_DISTANCE),
*      -(ZR + DZN * VIEW_DISTANCE))

* Rotate so that view plane normal is z axis
V = SQRT(DYN*DYN + DZN*DZN)
IF (V.GT.ROUNDOFF) THEN
   CALL ROTATE_X_3(-DYN/V,-DZN/V)
ENDIF
CALL ROTATE_Y_3(DXN,V)

* Determine the view-up direction in new coordinates
XUP_VP = DXUP * TMATRIX(1,1) + DYUP * TMATRIX(2,1) +
*         DZUP * TMATRIX(3,1)
YUP_VP = DXUP * TMATRIX(1,2) + DYUP * TMATRIX(2,2) +
*         DZUP * TMATRIX(3,2)

* Determine rotation needed to make view-up vertical
RUP = SQRT(XUP_VP*XUP_VP +YUP_VP*YUP_VP)
IF (RUP.LT.ROUNDOFF) THEN
   PRINT *,' ** ERROR: SET_VIEW_UP ALONG '
*   VIEWPLANE NORMAL **'
   PRINT *,' ** ALGOR 8.20 **'
   STOP
```

```
ENDIF
CALL ROTATE_Z_3(XUP_VP/RUP, YUP_VP/RUP)
IF (PERSPECTIVE_FLAG.EQ. .TRUE.) THEN
   CALL MAKE_PERSPECT_TRANSFORM
ELSE
   CALL MAKE_PARALLEL_TRANSFORM
ENDIF

RETURN
END
```

Comment : NEW_TRANSFORM_3 is changed to

NEW_TRANSFORMATION_3

```
* ALGOR 9.12
* extension of ALGOR 6.8

SUBROUTINE        CLIP_BOTTOM(OP,X,Y,Z,COLOR)

INTEGER*2         OP,COLOR,COLS(6)
REAL              X,Y,Z
REAL              WXL,WYL,WXH,WYH
REAL              XLM,XHM,YLM,YHM
REAL              XS(6),YS(6),ZS(6)
REAL              OLD_POINT_TEST(6)
REAL              NEW_POINT_TEST,X_CLIP,Y_CLIP,Z_CLIP

COMMON            /W_CUR/WXL,WYL,WXH,WYH
COMMON            /W_SLOP/XLM,YLM, XHM,YHM
COMMON            /L_PTS/XS,YS
COMMON            /L_PTZ/ZS, /L_COLS/COLS
COMMON            /C_OLDPT/OLD_POINT_TEST


NEW_POINT_TEST = YLM * Z + WYL

IF ( ( (Y .GE. NEW_POINT_TEST) .AND.
*        (YS(3) .LT. OLD_POINT_TEST(3)) ) .OR.
*        ( (Y .LE. NEW_POINT_TEST) .AND.
*        (YS(3) .GT. OLD_POINT_TEST(3)) ) )  THEN
     Z_CLIP = CLIPPED_Z(Y,Z,YS(3),ZS(3),YLM,WYL)
     Y_CLIP = YLM * Z_CLIP + WYL
     X_CLIP = CLIPPED_X_OR_Y(Y,X,Z,YS(3),XS(3),ZS(3),
*                Y_CLIP, Z_CLIP)

* fix bug in text book
     IF (X_CLIP.LT.WXL) X_CLIP = WXL
     IF (X_CLIP.GT.WXH) X_CLIP = WXH

     IF ((YS(3) .LT. OLD_POINT_TEST(3))
*       .OR. (OP.GT.31)) THEN
          CALL CLIP_TOP(1,X_CLIP,Y_CLIP,Z_CLIP,COLOR)
     ELSE
          CALL CLIP_TOP(OP,X_CLIP,Y_CLIP,Z_CLIP,COLOR)
     ENDIF
ENDIF

XS(3) = X
YS(3) = Y
ZS(3) = Z
COLS(3) = COLOR
OLD_POINT_TEST(3) = NEW_POINT_TEST
```

```
IF (Y.GE. NEW_POINT_TEST) THEN
 CALL CLIP_TOP(OP,X,Y,Z,COLOR)
ENDIF

RETURN
END
```

Comment : X value is readjusted not to exceed the window boundary.

```
* ALGOR 9.13
* extension of ALGOR 6.9

SUBROUTINE        CLIP_TOP(OP,X,Y,Z,COLOR)

INTEGER*2         OP,COLOR,COLS(6)
REAL              X,Y,Z
REAL              WXL,WYL,WXH,WYH
REAL              XLM,XHM,YLM,YHM
REAL              XS(6),YS(6),ZS(6)
REAL              OLD_POINT_TEST(6)
REAL              NEW_POINT_TEST,X_CLIP,Y_CLIP,Z_CLIP

COMMON            /W_CUR/WXL,WYL,WXH,WYH
COMMON            /W_SLOP/XLM,YLM, XHM,YHM
COMMON            /L_PTS/XS,YS
COMMON            /L_PTZ/ZS, /L_COLS/COLS
COMMON            /C_OLDPT/OLD_POINT_TEST


NEW_POINT_TEST = YHM * Z + WYH

IF ( ( (Y .LE. NEW_POINT_TEST) .AND.
*        (YS(4) .GT. OLD_POINT_TEST(4) ) ) .OR.
*        ( (Y .GE. NEW_POINT_TEST) .AND.
*        (YS(4) .LT. OLD_POINT_TEST(4)) ) )  THEN
     Z_CLIP = CLIPPED_Z(Y,Z,YS(4),ZS(4),YHM,WYH)
     Y_CLIP = YHM * Z_CLIP + WYH
     X_CLIP = CLIPPED_X_OR_Y(Y,X,Z,YS(4),XS(4),ZS(4),
*                Y_CLIP, Z_CLIP)

*   fix bug in text book
     IF (X_CLIP.LT.WXL) X_CLIP = WXL
     IF (X_CLIP.GT.WXH) X_CLIP = WXH

     IF ((YS(4).GT.OLD_POINT_TEST(4)) .OR.
*         (OP.GT.31)) THEN
         CALL CLIP_BACK(1,X_CLIP,Y_CLIP,Z_CLIP,COLOR)
     ELSE
         CALL CLIP_BACK(OP,X_CLIP,Y_CLIP,Z_CLIP,COLOR)
     ENDIF
ENDIF

XS(4) = X
YS(4) = Y
ZS(4) = Z
COLS(4) = COLOR
OLD_POINT_TEST(4) = NEW_POINT_TEST
```

```
IF (Y.LE. NEW_POINT_TEST) THEN
   CALL CLIP_BACK(OP,X,Y,Z,COLOR)
ENDIF


RETURN
END
```

Comment : Same argument as in algorithm 9.12.

```
* ALGOR 10.21
* Minimax test of two triangles

LOGICAL  FUNCTION        MINIMAX(L1,L2)

INTEGER*2          L1,L2

INTEGER*2          IDB(4096),COLD(4096)
REAL               XDB(4096),YDB(4096),ZDB(4096)

REAL               ROUNDOFF

COMMON             /C_DBUF/IDB,XDB,YDB,ZDB,COLD


ROUNDOFF = 0.0001

T1 = MAX(XDB(L1),MAX(XDB(L1-1),XDB(L1-2)))
T2 = MIN(XDB(L2),MIN(XDB(L2-1),XDB(L2-2)))
T3 = MAX(YDB(L1),MAX(YDB(L1-1),YDB(L1-2)))
T4 = MIN(YDB(L2),MIN(YDB(L2-1),YDB(L2-2)))


T5 = MAX(XDB(L2),MAX(XDB(L2-1),XDB(L2-2)))
T6 = MIN(XDB(L1),MIN(XDB(L1-1),XDB(L1-2)))
T7 = MAX(YDB(L2),MAX(YDB(L2-1),YDB(L2-2)))
T8 = MIN(YDB(L1),MIN(YDB(L1-1),YDB(L1-2)))

MINIMAX = (((T1-T2).LT.ROUNDOFF).OR.
*               ((T3-T4).LT.ROUNDOFF).OR.
*         ((T5-T6).LT.ROUNDOFF).OR.
*         ((T7-T8).LT.ROUNDOFF))


RETURN
END
```

Comment : YD(L1) of line 5 in the text should read YD(L2).

```
* ALGOR 10.29
* compares two triangles for depth order

FUNCTION TRIANGLE_COMPARE(I0,J0)

INTEGER*2         I0,J0
INTEGER*2         L1,I1,I2


TRIANGLE_COMPARE = 0

IF (MINIMAX(I0,J0).EQ. .TRUE.) THEN
   RETURN
ENDIF


L1 = Z_MINIMAX(I0,J0)

I1 = I0 - 2
I2 = I0 -1

DO WHILE ((TRIANGLE_COMPARE .EQ.0).AND. (I1 .LE. I0))
    TRIANGLE_COMPARE = COMPARE_SIDES(I1,I2,I0,J0,L1)

    I1 = I1 + 1

    IF (I1.EQ.I0) THEN
       I2 = I0 - 2
    ELSE
       I2 = I2 + 1
    ENDIF

ENDDO


IF (TRIANGLE_COMPARE.EQ.0) THEN
       TRIANGLE_COMPARE = COMPARE_CONTAINED(I0,J0,L1)
ENDIF

RETURN
END
```

Comment : Line TRIANGLE_COMPARE <> 0 in the text should read TRIANGLE_COMPARE = 0.

```
* ALGOR 10.31
* compare all triangles to determine their depth order

SUBROUTINE        COMPARE_ALL_TRIANGLES(NUMBER_OF_TRIANGLES)

INTEGER*2         NUMNBER_OF_TRIANGLES
INTEGER*2         I,K,L, NXT

INTEGER*2         INFRONT(4096), INBACK(4096)
INTEGER*2         INLIST(4096),INLINK(4096)

COMMON            /C_INF/INFRONT, /C_INB/INBACK
COMMON            /C_NXT/NXT
COMMON            /C_HEAP/INLIST,INLINK

NXT = 1

DO I = 1, NUMBER_OF_TRIANGLES
    INFRONT(I) = 0
    INBACK(I) = 0
ENDDO

DO I = 1, NUMBER_OF_TRIANGLES - 1

DO K  = I+1, NUMBER_OF_TRIANGLES

    L = TRIANGLE_COMPARE(3*I, 3*K)
    IF (L.GT.0) THEN
        CALL ADD_TO_LISTS(K,I)  !2ND INFRONT
    ENDIF

    IF (L.LT.0) THEN
        CALL ADD_TO_LISTS(I,K)  !1ST INFRONT
    ENDIF

ENDDO

ENDDO


RETURN
END
```

Comment : Inequality comparisons are interchanged.

```
* ALGOR 10.46
* Case of both endpoints visible

SUBROUTINE        CHOP_OUT_IN_OUT(INFR,BFREE,IDX)

INTEGER*2         INFR,BFREE,IDX
INTEGER*2         BCOL,OPB
REAL              LXA,LYA,LXB,LYB

REAL              X,Y,U,V
LOGICAL           CROSS
REAL              T,ROUNDOFF2

COMMON            /C_LDIV/LXA,LYA,LXB,LYB
COMMON            /C_BCOL/BCOL
COMMON            /C_OPB/OPB

ROUNDOFF2 = 0.0001

CALL LEFT_IN_FRONT(INFR)
CALL INTERSECTION_PAIR(X,Y,U,V,CROSS,IDX)

IF (CROSS.EQ. .FALSE.)  THEN
    RETURN
ENDIF

IF ((SIGNOF(LXA - LXB) .NE. SIGNOF(X - U)) .OR.
*       (SIGNOF(LYA - LYB) .NE. SIGNOF(Y - V))) THEN
   T = U
   U = X
   X = T
   T = V
   V = Y
   Y = T
ENDIF

IF ((ABS(X - LXA) + ABS(Y - LYA)).LT.ROUNDOFF2) THEN
 IF ((ABS(U-LXB) + ABS(V-LYB)).LT.ROUNDOFF2) THEN
    CALL CHANGE_OP_CODE(1)
 ELSE
    CALL PUT_IN_B(1,U,V,0.0,BFREE,BCOL)
    BFREE = BFREE + 1
    LXA = U
    LYA = V
 ENDIF
ELSE
 TRIAN_INFR = INFR
 IF ((ABS(U-LXB) + ABS(V-LYB)).LT.ROUNDOFF2) THEN
    CALL CHANGE_OP_CODE(1)
```

```
*    CALL PUT_IN_C(2,X,Y,TRIAN_INFR,BCOL)
*    set to opcode of point B
     CALL PUT_IN_C(OPB,X,Y,TRIAN_INFR,BCOL)

 ELSE
     CALL PUT_IN_C(1,U,V,TRIAN_INFR,BCOL)
*    CALL PUT_IN_C(2,X,Y,TRIAN_INFR,BCOL )
*    set to opcode of point B
     CALL PUT_IN_C(OPB,X,Y,TRIAN_INFR,BCOL)

 ENDIF
ENDIF


RETURN
END
```

Comment : Wrong logical indication of NO_CROSS in the text.

```
* ALGOR 10.48
* Routine to remove an instruction from the C-buffer stack
* and save it in the B buffer

SUBROUTINE        SIDE_IS_DONE(BFREE,OP)

INTEGER*2         BFREE,OP
INTEGER*2         BCOL
REAL              LXA,LYA,LXB,LYB
INTEGER*2         CFREE,I

INTEGER*2         IBB(4096),COLB(4096)
REAL              XBB(4096),YBB(4096),ZBB(4096)

COMMON            /C_LDIV/LXA,LYA,LXB,LYB
COMMON            /C_CFREE/CFREE
COMMON            /C_BCOL/BCOL
COMMON            /C_BBUF/IBB,XBB,YBB,ZBB,COLB

CFREE = CFREE - 1

*         adjust the endpoint
*         check conflict opcode of a point
IF (OP .EQ. 2) THEN
   I = BFREE -1
   IF ( (LXB.EQ.XBB(I)) .AND. (LYB.EQ.YBB(I)).AND.
*        (IBB(I) .EQ.1) ) THEN
       OP = 1
   ENDIF
ENDIF

CALL PUT_IN_B(OP,LXB,LYB,0,BFREE,BCOL)
BFREE = BFREE + 1
LXA = LXB
LYA = LYB

RETURN
END
```

Comment : This is to adjust the end point command coding to prevent it from creating a dot.

```
* ALGOR 10.51A
* similar to algor 10.26 INSIDE
* to test if a point is inside a triangle formed by
* 3 vertices X1,Y1,X2,Y2,X3,Y3


LOGICAL FUNCTION INSIDE_TRIANGLE(X,Y,X1,Y1,X2,Y2,X3,Y3)

REAL              X,Y,X1,Y1,X2,Y2,X3,Y3
REAL              XMAX,XMIN,YMAX,YMIN
REAL              ROUNDOFF

ROUNDOFF = 0.0001

XMAX = MAX(X1,MAX(X2,X3))
XMIN = MIN(X1,MIN(X2,X3))
YMAX = MAX(Y1,MAX(Y2,Y3))
YMIN = MIN(Y1,MIN(Y2,Y3))

INSIDE_TRIANGLE = .FALSE.

IF (((X - XMAX).GT.ROUNDOFF).OR.((XMIN - X) .GT. ROUNDOFF)
*          .OR.((Y - YMAX).GT.ROUNDOFF).OR.((YMIN - Y)
*          .GT. ROUNDOFF) )  THEN
     RETURN
ENDIF

IF ( HALF_PLANE(X,Y,     X1,Y1, X2,Y2) .NE.
*    HALF_PLANE(X3,Y3,   X1,Y1, X2,Y2) )       RETURN

IF ( HALF_PLANE(X,Y,     X2,Y2, X3,Y3) .NE.
*    HALF_PLANE(X1,Y1,   X2,Y2, X3,Y3) )       RETURN

     INSIDE_TRIANGLE = HALF_PLANE(X,Y,X3,Y3,X1,Y1) .EQ.
*          HALF_PLANE(X2,Y2, X3,Y3,X1,Y1)

RETURN
END
```

Comment : Extra subroutine is created for use in algorithm 10.51.

```
* ALGOR 10.51
* find two vertices on which to divide the polygon

FUNCTION SPLIT_VERTEX(L,LA,LB,M,N)

INTEGER*2         L, LA, LB, M, N
INTEGER*2         IBB(4096),COLB(4096)
REAL              XBB(4096),YBB(4096),ZBB(4096)
REAL              X,Y,X1,Y1,X2,Y2,X3,Y3
INTEGER*2         LPU,LPL, K

COMMON            /C_BBUF/IBB,XBB,YBB,ZBB,COLB

IF (YBB(LB).GT.YBB(LA)) THEN
   LPU = LB
   LPL = LA
ELSE
   LPU = LA
   LPL = LB
ENDIF

IF ( XBB(LB).GT. XBB(LA)) THEN
   SPLIT_VERTEX = LB
ELSE
   SPLIT_VERTEX = LA
ENDIF

X1 = XBB(L)
Y1 = YBB(L)
X2 = XBB(LPU)
Y2 = YBB(LPU)
X3 = XBB(LPL)
Y3 = YBB(LPL)
DO K = M, N
     X = XBB(K)
     Y = YBB(K)
     IF (INSIDE_TRIANGLE(X,Y,X1,Y1,X2,Y2,X3,Y3)
*       .EQ. .TRUE.) THEN
        SPLIT_VERTEX = K
     ENDIF
ENDDO
XS = XBB(SPLIT_VERTEX)
YS = YBB(SPLIT_VERTEX)

RETURN
END
```

Comment : Minmax test in the text is too primitive.

```
* ALGOR 10.52
* splits the polygon into two polygons

SUBROUTINE        DOSPLIT(M,M1,N1,N)

INTEGER*2         M,N,M1,N1

INTEGER*2         IBB(4096),COLB(4096)
REAL              XBB(4096),YBB(4096),ZBB(4096)

INTEGER*2         K,K1,K2,J

COMMON            /C_BBUF/IBB,XBB,YBB,ZBB,COLB


K = N + 3 - M

DO J = M1, N1
    CALL SHIFT_BUFFER(J,J+K)
ENDDO

DO J = N,N1, -1
    CALL SHIFT_BUFFER(J,J+2)
ENDDO

K = N1 - M1 + 1
DO J = M1, M, -1
   CALL SHIFT_BUFFER(J,J+K)
ENDDO

K1 = N + 3 - M
K2 = M - M1
DO J = M1, N1
   CALL SHIFT_BUFFER(J+K1, J+K2)
ENDDO

IBB(M) = 1
IBB(N1+2) = 1


RETURN
END
```

Comment : Wrong index M1 in the text.

```
* ALGOR 10.53
* replacement for algor 10.17

SUBROUTINE        POLYGON_SPLIT(SIDES,DFREE,KNT)

INTEGER*2         SIDES,DFREE,KNT
INTEGER*2         SOURCE_POLY(4096)

INTEGER*2         POLYGON_END(4096),DO_NEXT,M,N,JP,L
INTEGER*2         LA,LB,LS,M1,N1,ILEFT

INTEGER*2         IBB(4096),COLB(4096)
REAL              XBB(4096),YBB(4096),ZBB(4096)

INTEGER*2         I

COMMON            /C_BBUF/IBB,XBB,YBB,ZBB,COLB
COMMON            /C_SOURCE/SOURCE_POLY


DO_NEXT = 2
POLYGON_END(1) = 0
POLYGON_END(2) = SIDES

DO WHILE (DO_NEXT.GT.1)

     M = POLYGON_END(DO_NEXT - 1) + 1
    N = POLYGON_END(DO_NEXT)

    IF ( (N-M).EQ.2) THEN
*         triangle case
       L = M+1

       IF (IN_A_LINE(N).EQ. .FALSE.) THEN
         JP = INT((DFREE + 2.0)/3.0)
         SOURCE_POLY(JP) = KNT
         CALL PUT_IN_D(M,N,DFREE)
       ELSE
*      in case of a straight line, enter it as a line
*      the 3 points directly into display file
          CALL VIEWING_TRANSFORM(1,XBB(N),YBB(N),COLB(N))
          DO I = M,N
             CALL VIEWING_TRANSFORM(IBB(I),XBB(I),
*                        YBB(I),COLB(I))
          ENDDO
        ENDIF
         DO_NEXT = DO_NEXT - 1
     ELSE
```

```
*       polygon case
*       the following code replace LEFT_MOST(M,N)
        L = M
        DO ILEFT = M+1, N
        IF (XBB(ILEFT).LT.XBB(L)) THEN
         L = ILEFT
         ENDIF
        ENDDO

        IF (L.EQ.N) THEN
         LA = M
        ELSE
         LA = L + 1
        ENDIF
        IF (L.EQ.M) THEN
         LB = N
        ELSE
         LB = L - 1
        ENDIF
        LS = SPLIT_VERTEX(L,LA,LB,M,N)


        IF ((LS.EQ.LA).OR.(LS.EQ.LB))    THEN
           M1 = MIN(LA,LB)
           N1 = MAX(LA,LB)
        ELSE
           M1 = MIN(L,LS)
           N1 = MAX(L,LS)
        ENDIF
        CALL DOSPLIT(M,M1,N1,N)
        POLYGON_END(DO_NEXT) = M + N1 - M1
        DO_NEXT = DO_NEXT + 1
        POLYGON_END(DO_NEXT) = N + 2
      ENDIF

 ENDDO

 RETURN
 END
```

Comment : If the straight line formed by 3 points is ignored, the polygon when displayed becomes broken. Therefore a provision for straight case is needed.

CHAPTER 10

CONCLUSION

The CORE protocol system implemented and applied to the experimenation of 3D surface reconstruction by triangulation was found quite effective. The speed of the system is slowed down when the number of the surfaces is increased. It is due to the fact that there is a huge amount of computation by the Painter's algorithm in hidden-surface and hidden-line removal process. A better throughput can be improved if some other algorithm, for example quad-tree method, is used.

The triangulation method used here is found satisfactory. The quality of triangulated surface representation is very acceptable. When the number of slices is increased, the size of the display file has to be increased tremendously. One triangle display will occupy 4 elements in the display file. In other words, the increase of size is four fold.

In any event the system is built very portable and simple to use. Most importantly, it has provided all the standard features that a graphics system should provide. In

the future, the author hopes add-on enhancement can be implemented by the users. Results of the experimentation are given in appendix B.

REFERENCES

[1] Boissonnat, J. D., "Representation of Objects by Triangulating Points in 3-D Space," IEEE Proceeding, 6th International Conference on Pattern Recognition, Vol. 2, 1982, pp. 830 - 832.

[2] Boissonnat, J. D. and Faugeras, P. D., "Triangulation of 3-D Objects," Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vol. 2, 1981, pp 658 - 660.

[3] Faugeras, P. D. and Pauchon, E., "Measuring the Shape of 3-D Objects," IEEE Computer Vision and Pattern Recognition, 1983, pp. 2 - 7.

[4] Fuchs, H., Kedem, Z. M., and Uselton, S. P., "Optimal Surface Reconstruction from Planar Contours," Communications of ACM, Vol. 20, Oct., 1977, pp. 693 - 702.

[5] Graham, R. L. and Yao, F. F., "Finding the Convex Hull of a Simple Polygon," Journal of Algorithms, Vol. 4, 1983, pp. 324 - 331.

[6] Harrington, S., "Computer Graphics - A Programming Approach," McGraw-Hill, 1983.

[7] Hearn, D. and Baker, M. P., "Computer Graphics," Prentic-Hall, 1986.

[8] Johnson, D. B., "Efficient Algorithms for Shortest Paths in Sparse Networks," Journal of ACM, Vol. 24, Jan., 1977, pp. 1 - 13.

[9] Newman, W. and Sproull, R. F., "Principle of Interactive Computer Graphics," McGraw-Hill, 1979.

[10] Pavlidis, T., "Algorithms for Graphics and Image Processing," Computer Science Press, 1982.

[11] Pieroni, G. G. and Freeman, H., "Computer Architecture for Spatially Distributed Data," Spinger-Verlag, 1985.

[12] Plastock, R. A. and Kalley, G., "Computer Graphics," McGraw-Hill, 1986.

[13] Shamos, M. I. and Hoey, D., "Closest-Point Problems," 6th Annual IEEE Symposium on Foundations of Computer Science, Oct., 1975, pp. 151 - 162.

[14] Toussaint, G. T., "Pattern Recognition and Geometrical Complexity," IEEE Proceedings, 5th International Conference on Pattern Recognition, Vol. 1, 1980, pp. 1324 - 1347.

[15] Toussaint, G. T. and Avis, D., "On a Convex Hull Algorithm for Polygons and Its Application to Triangulation Problems," IEEE Proceedings, Pattern Recognition, Vol. 15, 1982, pp. 23 - 28.

[16] Yuval, G., "Finding Near Neighbours in K-Dimensional Space," Information Processing Letters, Vol. 3, No. 4, March 1975, pp. 113 - 114.

APPENDIX A

USER CALLABLE SUBROUTINES

In this Appendix, a list of user available subroutines is given below. These subroutines are written in VAX FORTRAN 77. Argument names prefixed by character I are in INTEGER*2, otherwise they are in REAL. Prefix IA, or A is to denote an array argument either in INTEGER*2 or REAL. ONOFF stands for LOGICAL value, either .TRUE. or .FALSE.

```
ERASE
MOVE_ABS_2(X,Y,ICOLOR)
LINE_ABS_2(X,Y,ICOLOR)
MOVE_REL_2(DX,DY,ICOLOR)
LINE_REL_2(DX,DY,ICOLOR)
NEW_FRAME
MAKE_PICTURE_CURRENT
SET_CHARUP(DX,DY)
SET_CHARSPACE(SPACING)
TEXT(STRING,IACOLORS)
     STRING : CHARACTER*1  STRING(80)
     IACOLORS : INTEGER*2  IACOLORS(80)
SET_LINE_STYLE(ISTYLE)
POLYGON_ABS_2(AX,AY,I,IACOLORS)       dimension 128
```

```
POLYGON_REL_2(AX,AY,I,IACOLORS)       dimension 128
SET_FILL(ICOLOR)
TRANSLATE(TX,TY)
SCALE(SX,SY)
ROTATE_ANGLE(DEGREE)
CREATE_SEGMENT(INAME)
DELETE_SEGMENT(INAME)
DELETE_ALL_SEGMENTS
RENAME_SEGMENT(IOLDNAME, INEWNAME)
SET_VISIBILITY(INAME,ONOFF)
SET_IMAGE_TRANSLATION(INAME,TX,TY)
SET_VIEWPORT(XL,XH,YL,YH)
SET_WINDOW(XL,XH,YL,YH)
MOVE_ABS_3(X,Y,Z)
MOVE_REL_3(DX,DY,DZ)
LINE_ABS_3(X,Y,Z,ICOLOR)
LINE_REL_3(X,Y,Z,ICOLOR)
POLYGON_ABS_3(AX,AY,AZ,I,IACOLORS)   dimension 128
POLYGON_REL_3(AX,AY,AZ,I,IACOLORS)   dimension 128
SET_VIEW_REFERENCE_POINT(X,Y,Z)
SET_VIEW_PLANE_NORMAL(DX,DY,DZ)
SET_VIEW_DISTANCE(D)
SET_VIEW_UP(DX,DY,DZ)
SET_PARALLEL(DX,DY,DZ)
SET_PERSPECTIVE(XC,YC,ZC)
```

```
SET_VIEW_DEPTH(FRONT_DISTANCE,BACK_DISTANCE)
SET_FRONT_PLANE_CLIPPING(ONOFF)
SET_BACK_PLANE_CLIPPING(ONOFF)
SET_HIDDEN_LINE_REMOVAL(ONOFF)
SET_SHADING(ONOFF)
SET_OBJECT_SHADE(REFLECTIVITY,SHINE,GLOSS)
SET_LIGHT(X,Y,Z,BRIGHTNESS,BACKGROUND)
SET_SMOOTH(ILINES_PER_SEGMENT)
START_CURVE(AX,AY,AZ,IACOLORS)     dimension 4
CURVE_ABS_3(X,Y,Z,ICOLOR)
END_CURVE(X,Y,Z,ICOLOR)
SMOOTH_POLY_ABS_3(AX,AY,AZ,I,IACOLORS)  dimension 128
```

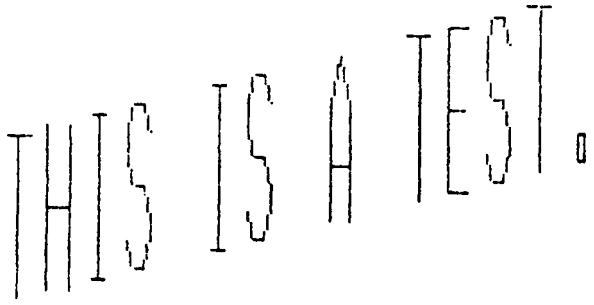
APPENDIX B

DIFFERENT GRAPHICS OUTPUTS

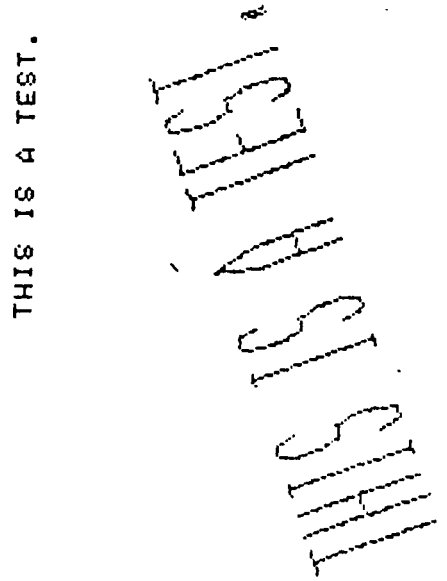
The system has been implemented on different graphics output devices. They are Lexidata, Tektronix, Printronix, regular line printer and CRT terminal. The results of the experimentation on these devices are included for reference.

T H I S I S A T E S T .

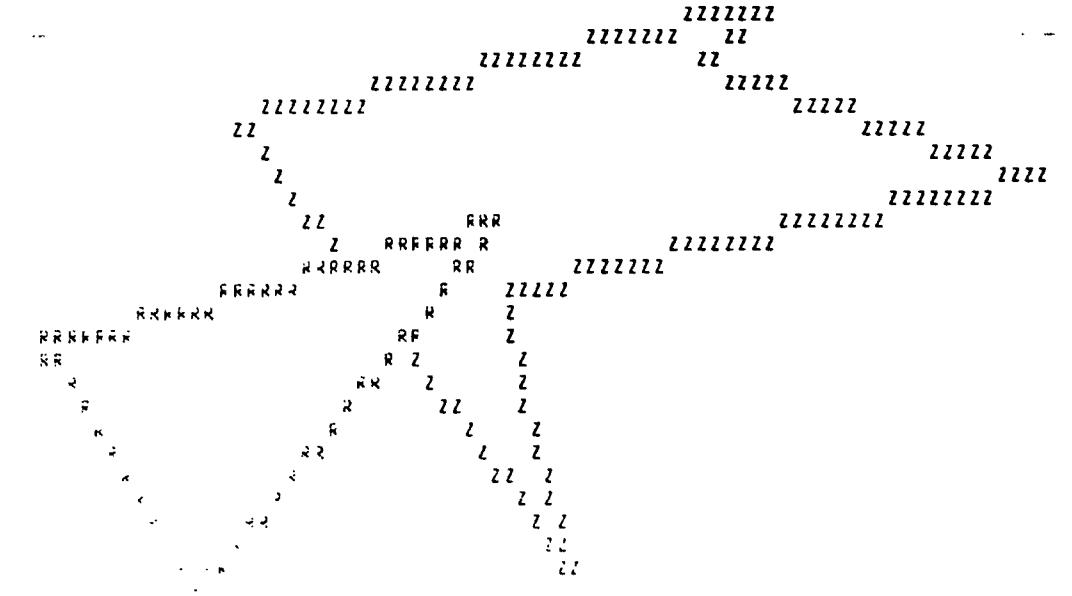
THIS IS A TEST.



THIS IS A TEST.



PRINTRONIX output



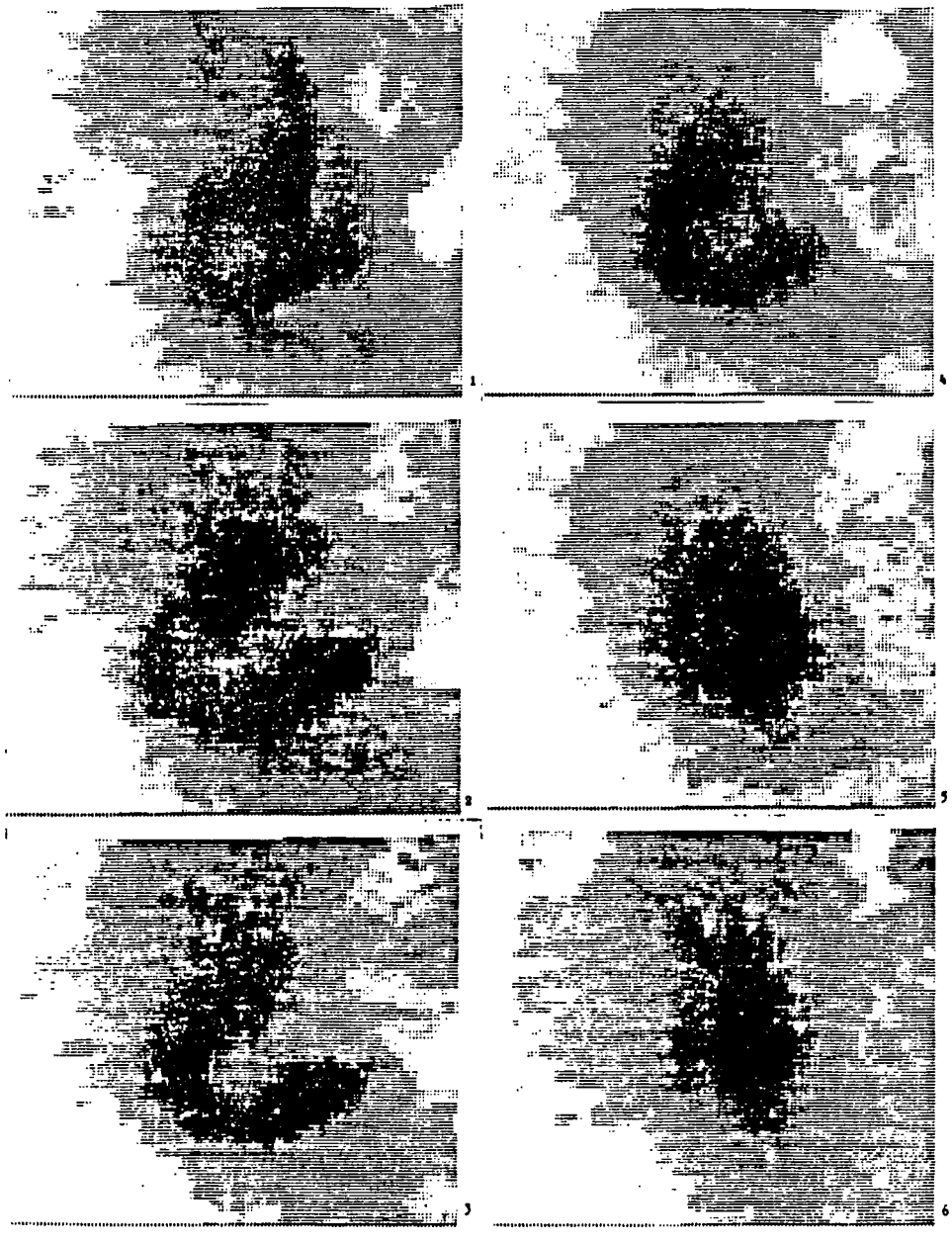
HIDDEN LINE REMOVAL

Line printer output

MY FAIR LADY

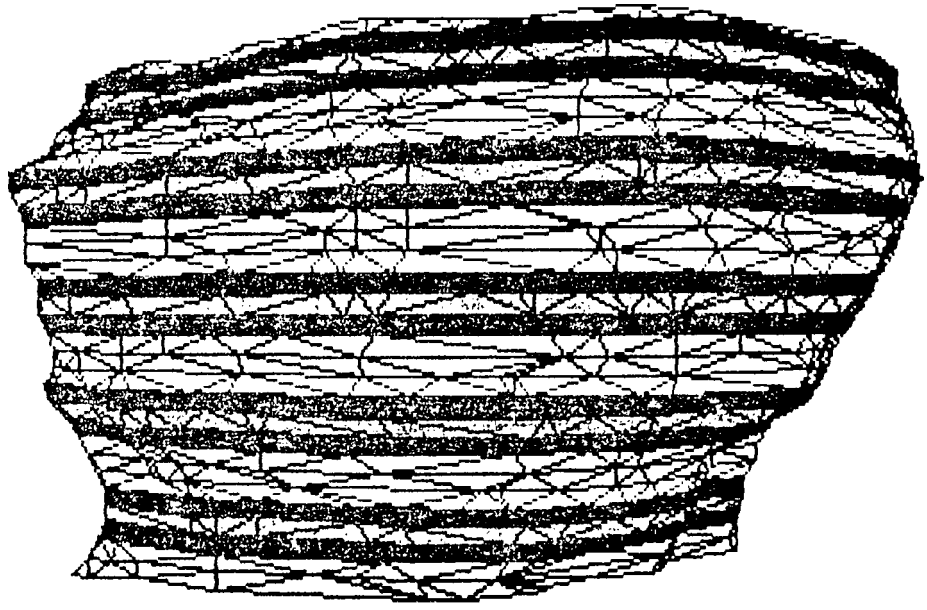
TEST FOR B_SPLINE CURVE

Dot Matrix Printer : PRINTRONIX output



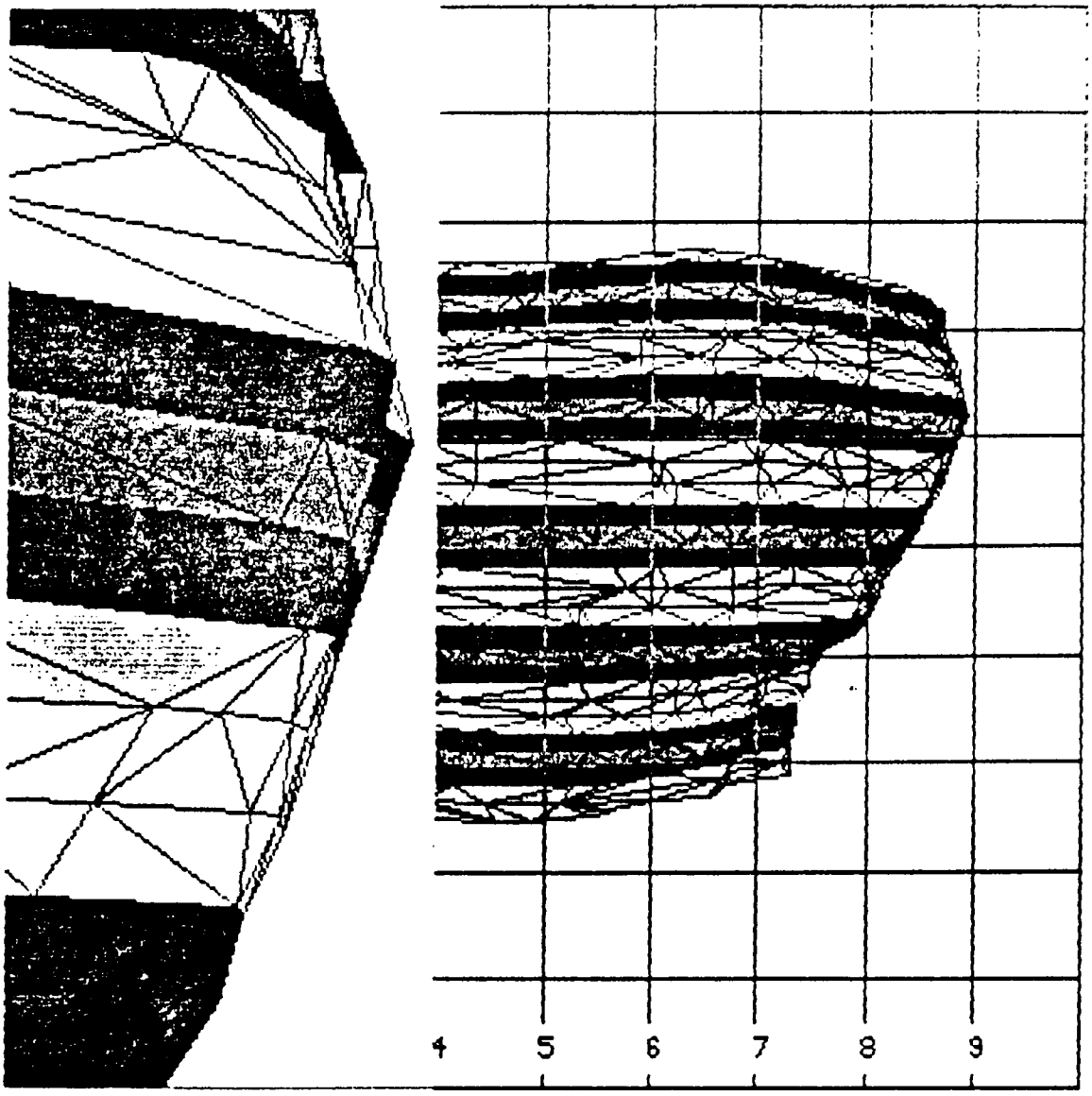
Sequence of PET images

Data set 1



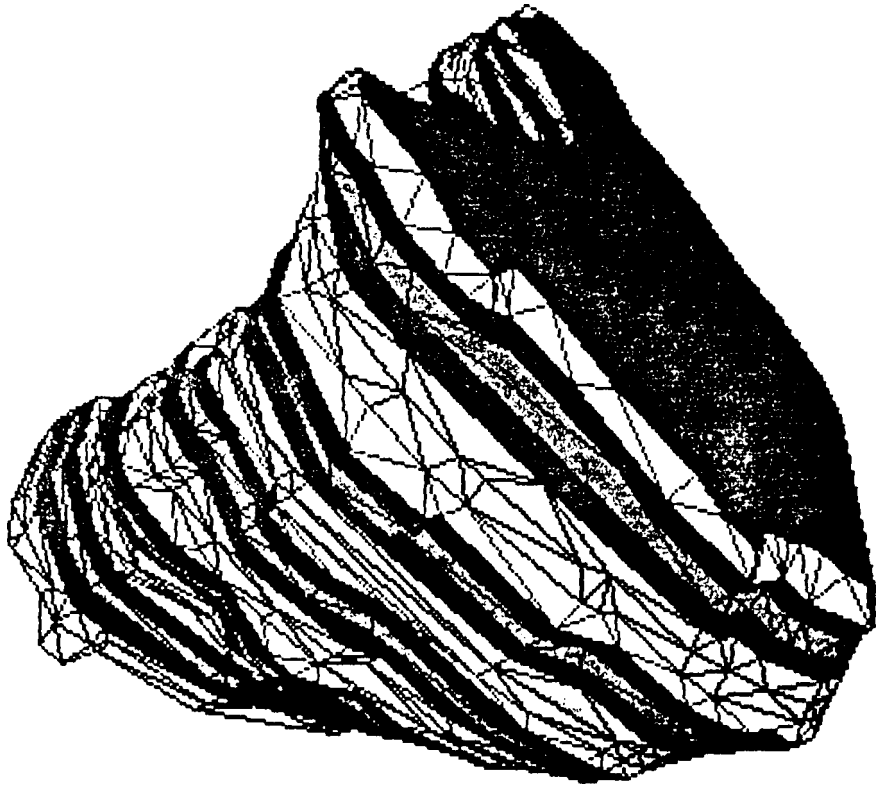
Scale(3.,3.) Perspective(.5,.5,.5) Rotation(0,0,0)

TEKTRONIX output



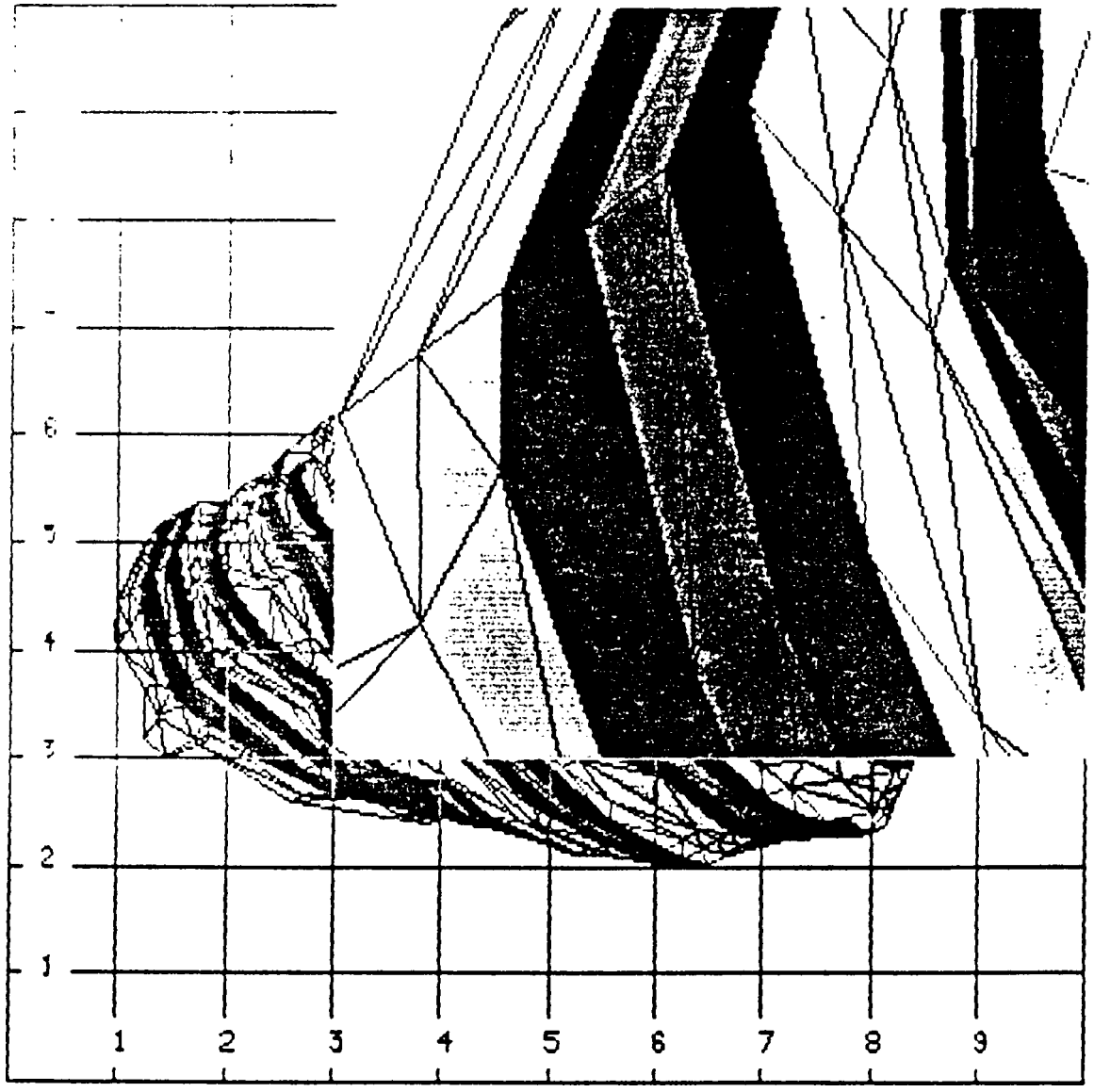
window(x1,x2,y1,y2)= (.8,.9,.5,.7) viewport(.0,.4,.0,1.)

TEKTRONIX output



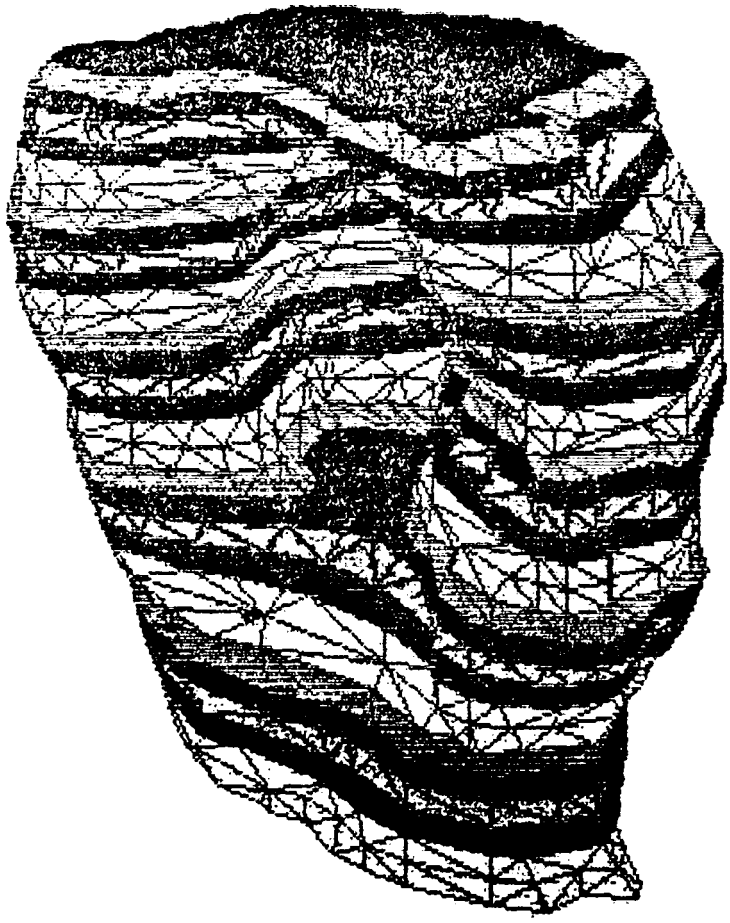
scale(3.,3.) perspective(.5,.5,.5) rotation(60,60,0)

TEKTRONIX output



window(x1,x2,y1,y2)= (.1,.2,.4,.5) viewport(.3,1.,.3,1.)

TEKTRONIX output

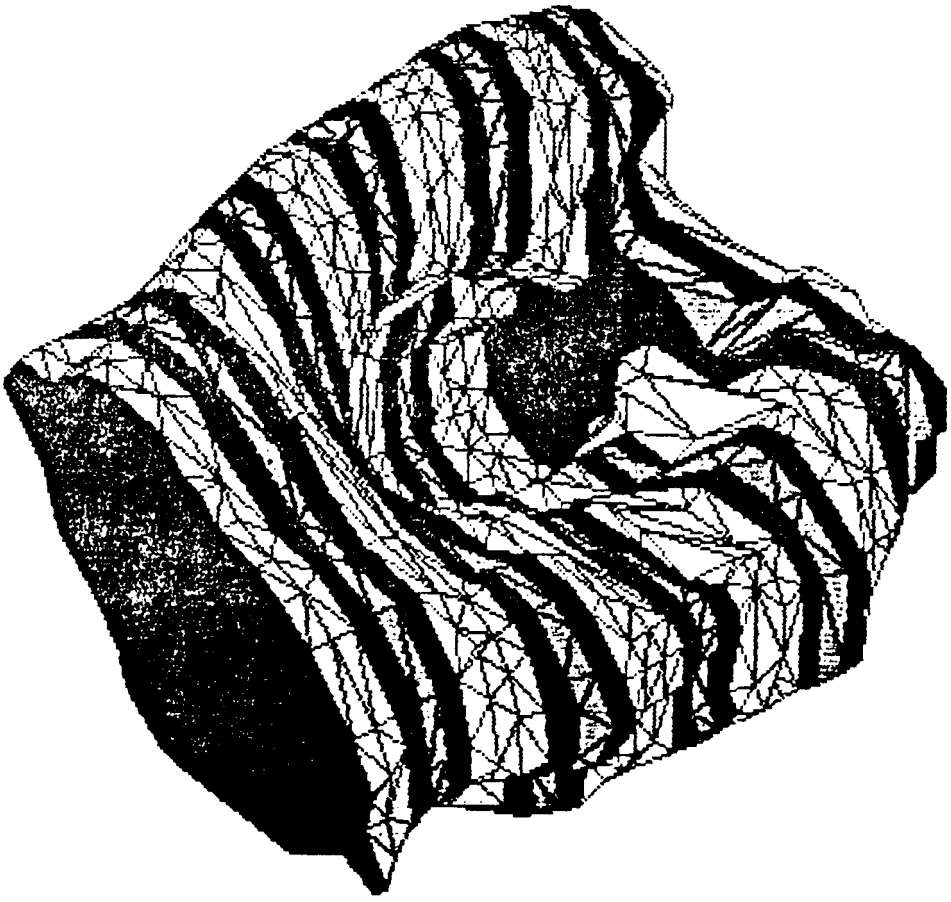


scale(2.5,2.5) perspective(.5,1.,1.) rotation(0,110,0)

TEKTRONIX output

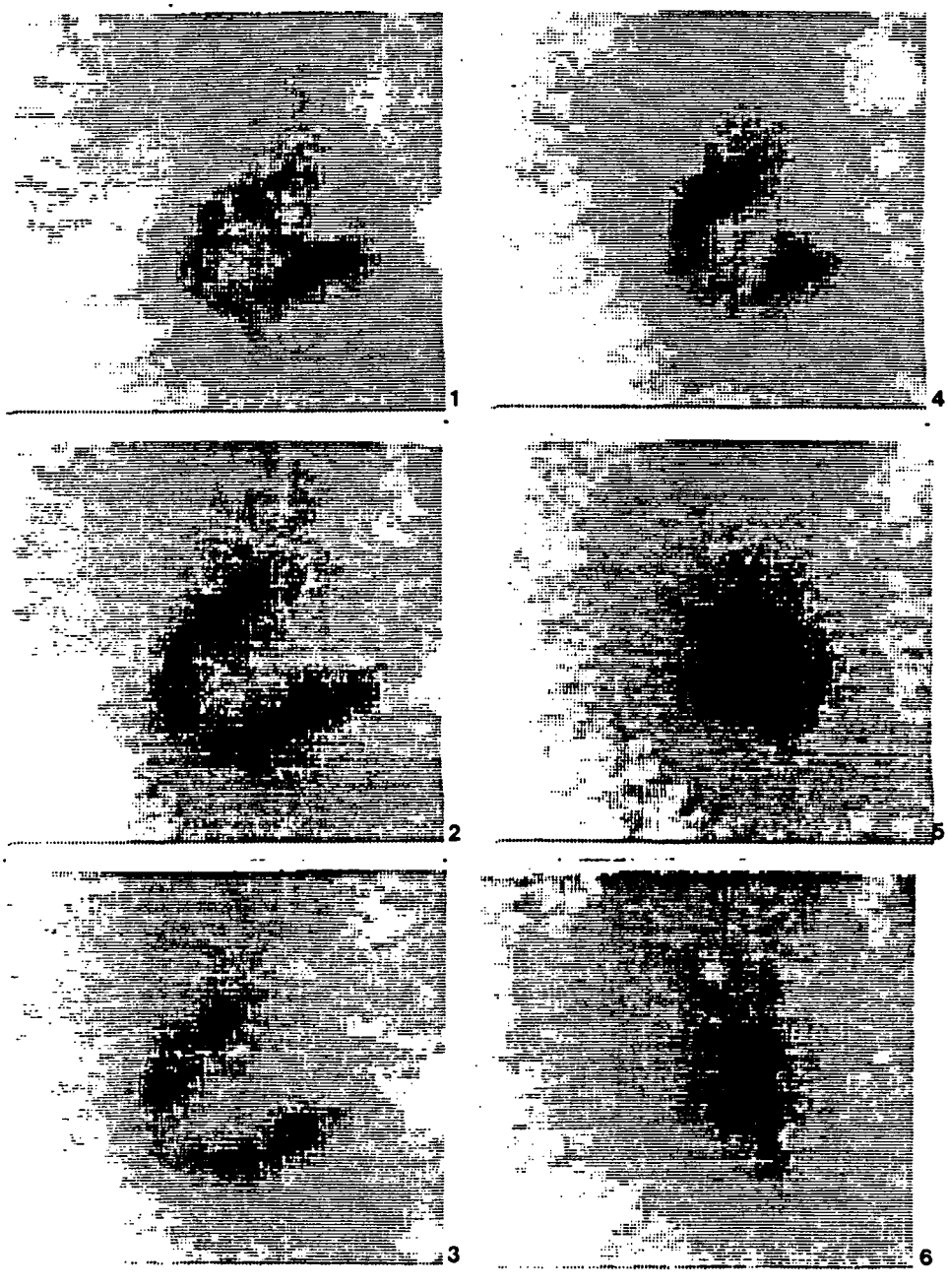
scale(2.5,2.5) perspective(.5,1,1.) rotation(-30,110,-30)

TEKTRONIX output



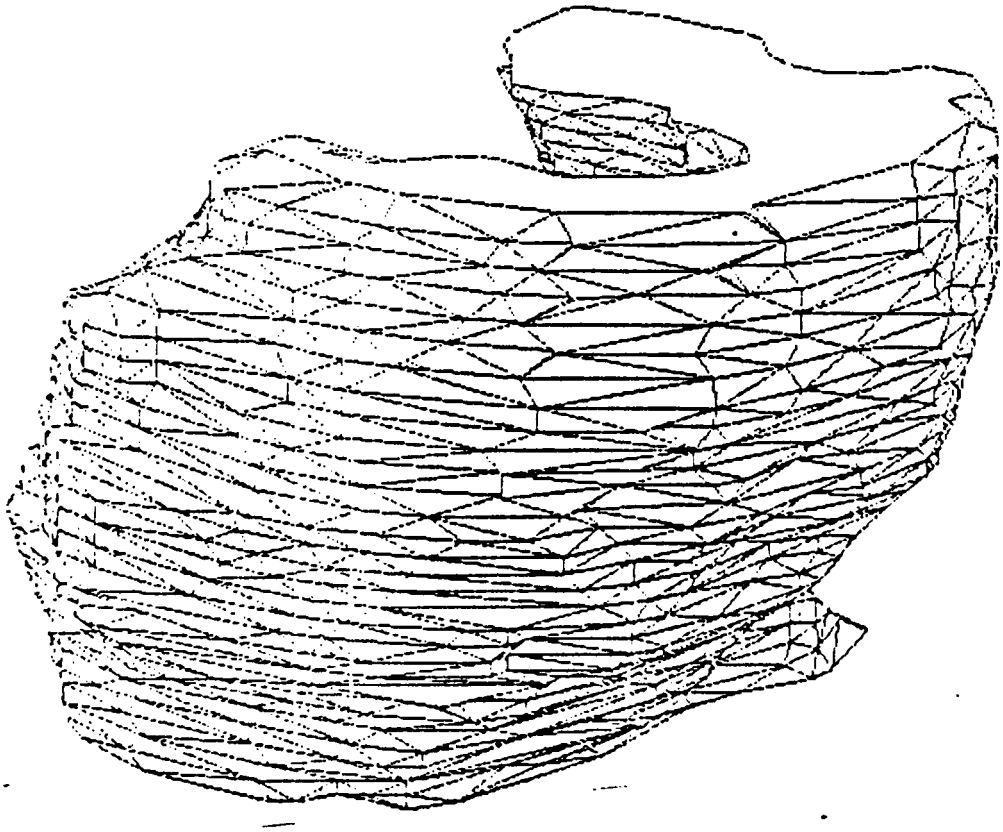
scale(2.5,2.5) perspective(.5,1.,1.) rotation(-30,110,-75)

TEKTRONIX output



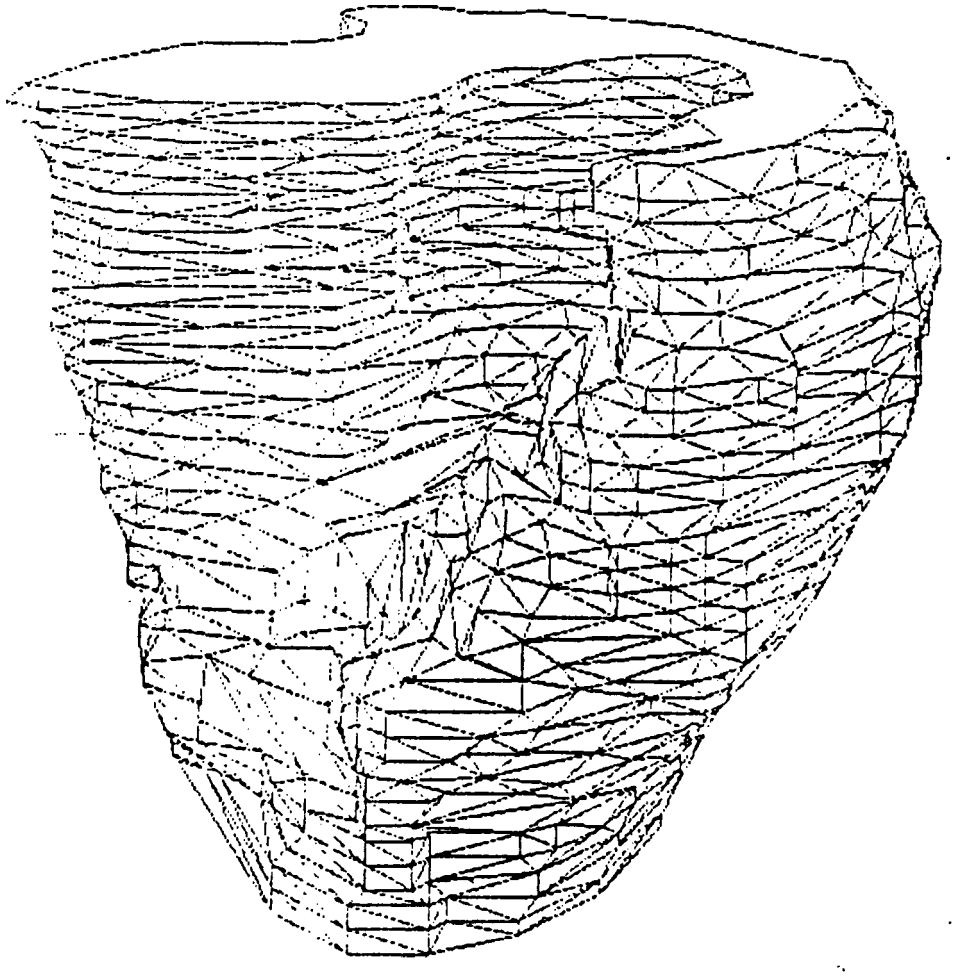
Sequence of PET images

Data set 2



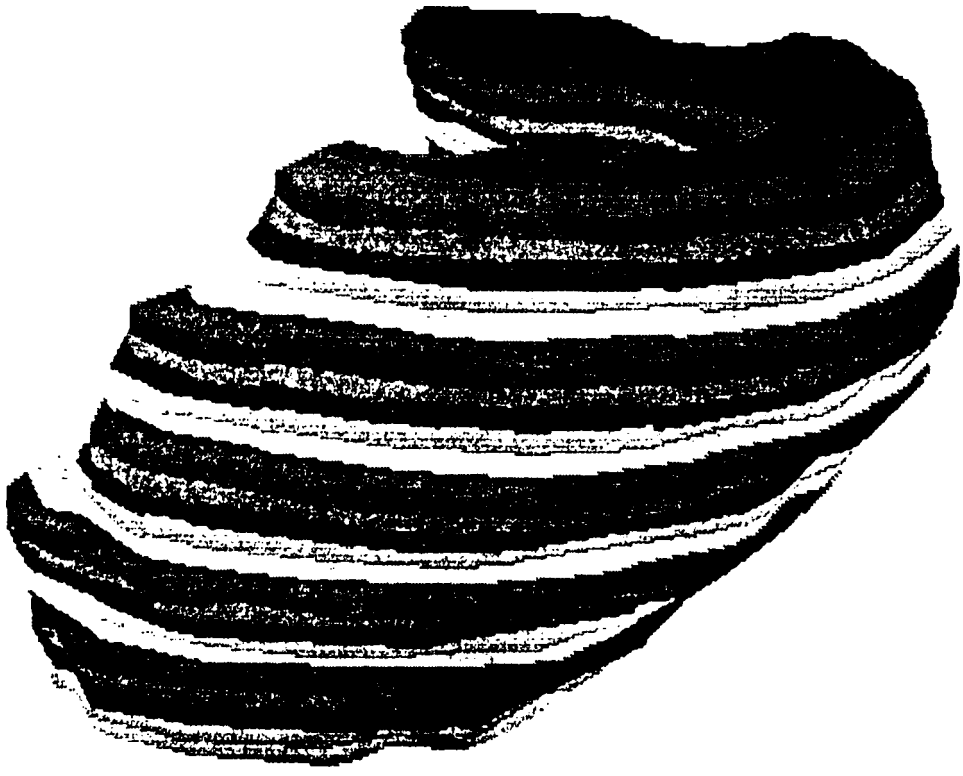
scale(3.,3.) perspective(.5,1.,1.) rotation(0.,0.,0.)

PRINTRONIX output



scale(3.,3.) perspective(.5,1.,1.) rotation(0,100,0)

PRINTRONIX output

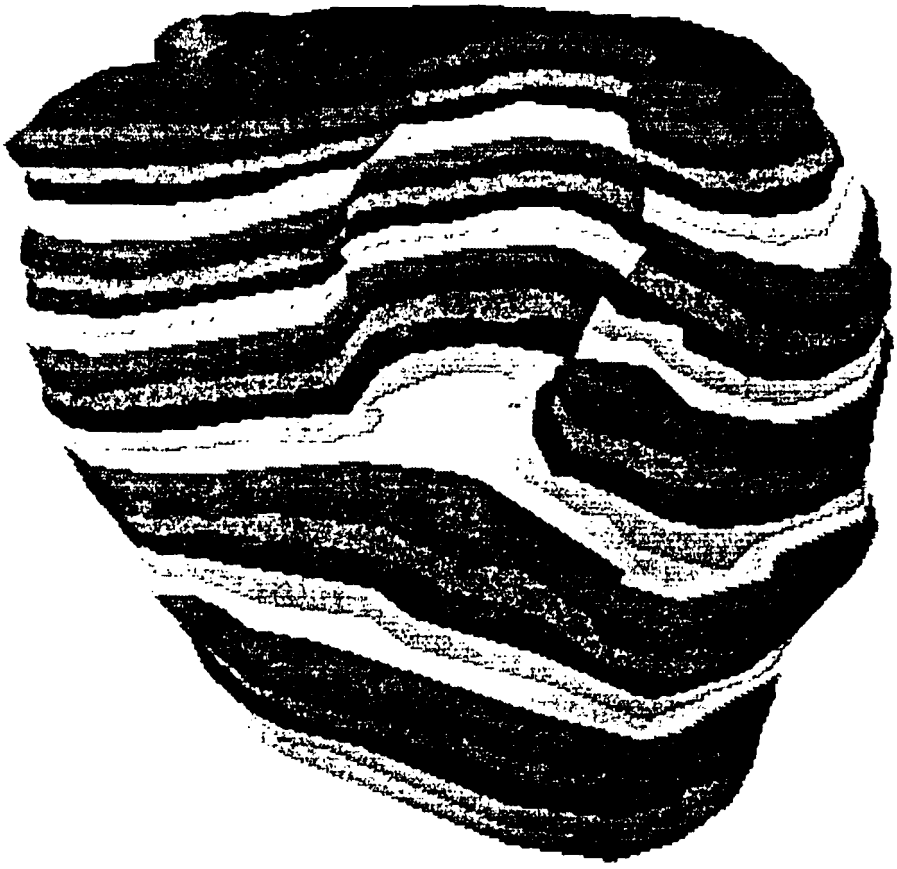


scale(2.5,2.5) perspective(.5,1.,1.) rotation(o,20,0)

TEKTRONIX output

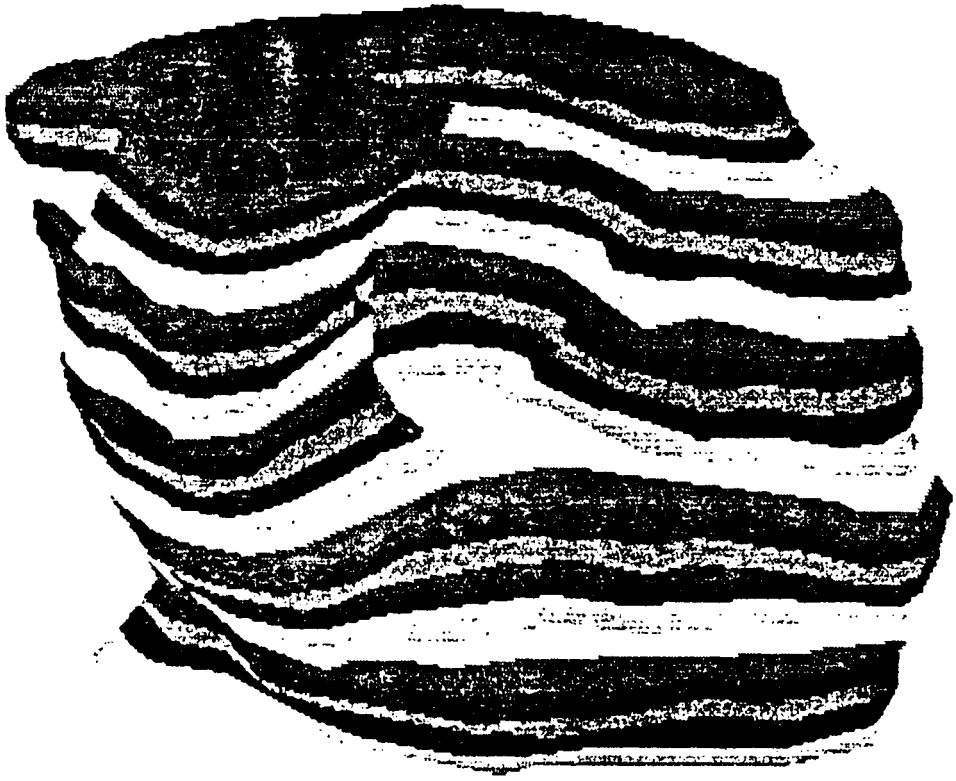
scale(2.5,2.5) perspective(.5,1.,1.) rotation(0,80,0)

TEKTRONIX output



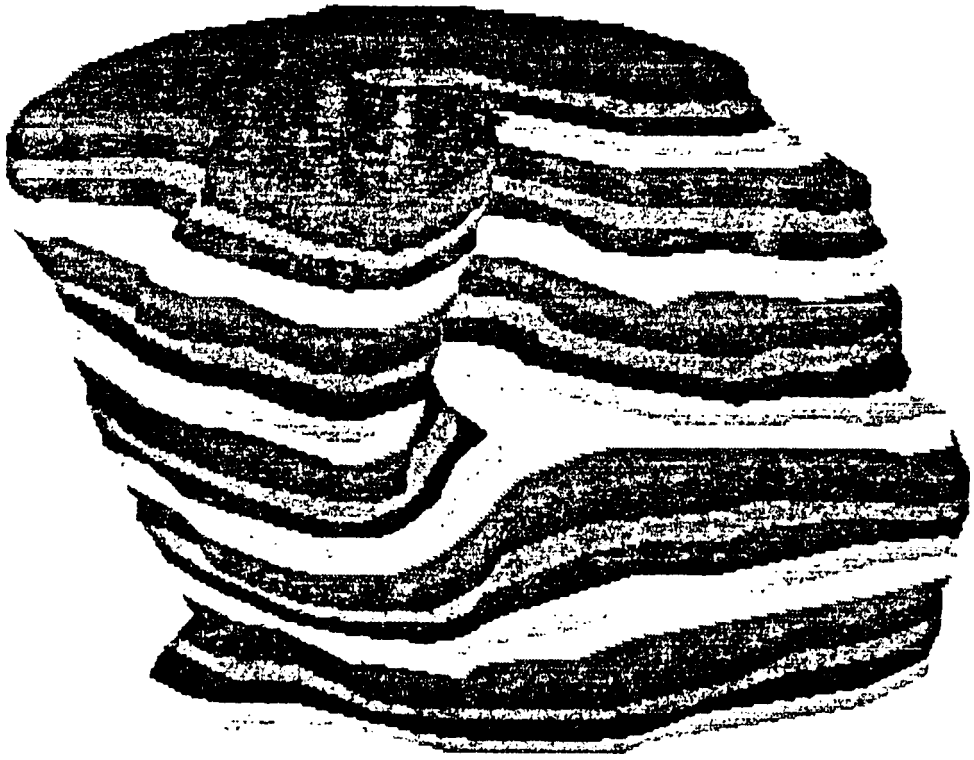
scale(2.5,2.5) perspective(.5,1.,1.) rotation(0,120,0)

TEKTRONIX output



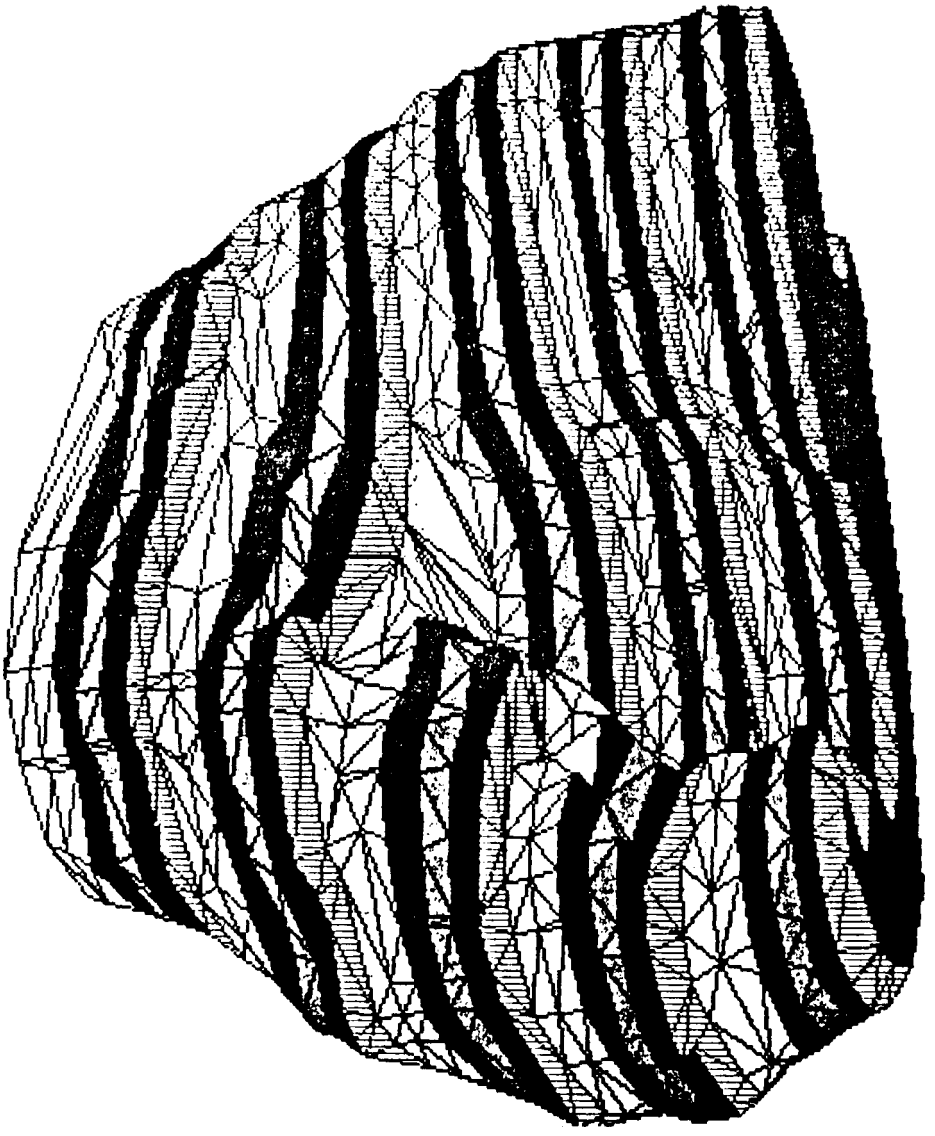
scale(2.5,2.5) perspective(.5,1.,1.) rotation(0,160,0)

TEKTRONIX output



scale(2.5,2.5) perspective(.5,1.,1.) rotation(0,180,0)

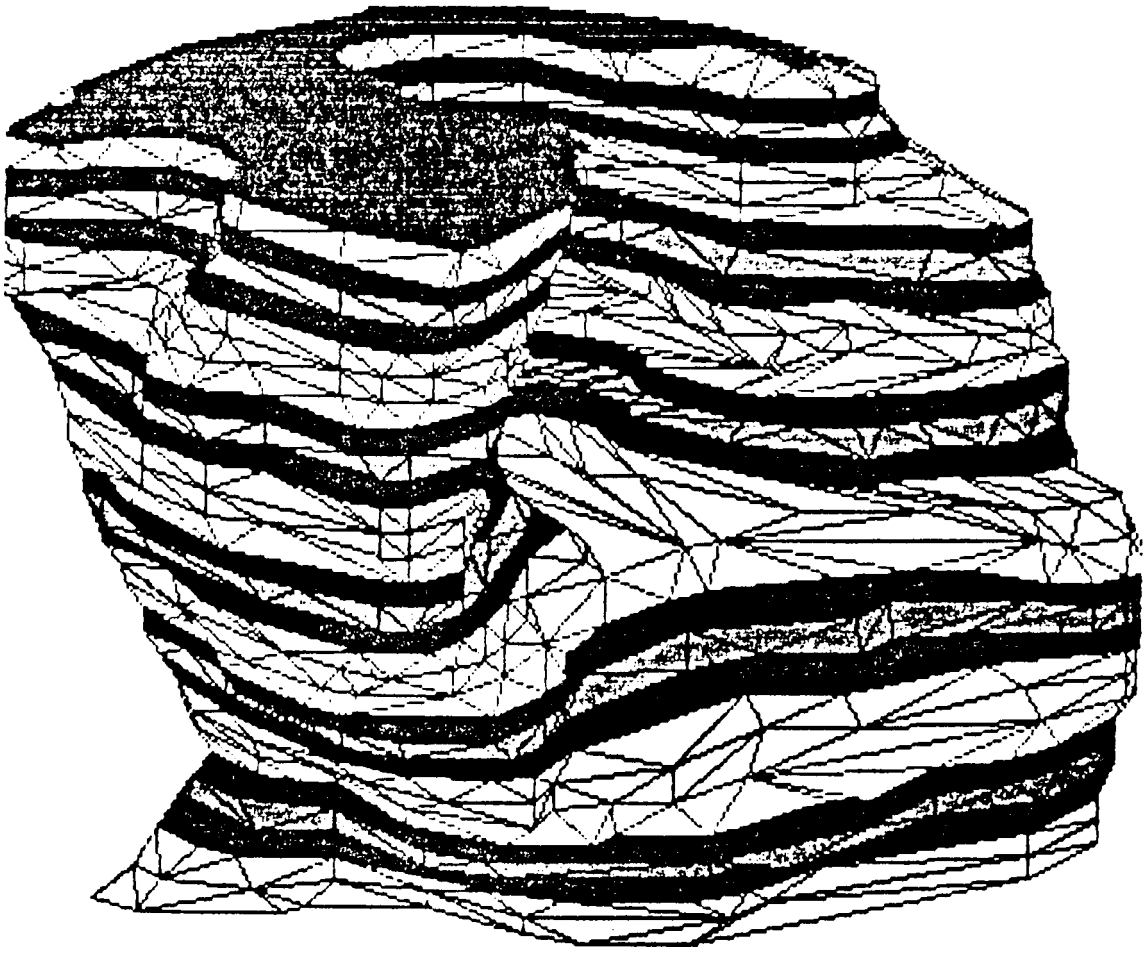
TEKTRONIX output



(with triangles shown)

scale(3.,3.) perspective(.5,1.,1.) rotation(80,90,0)

TEKTRONIX output



(with triangles shown)

scale(3.,3.) perspective(.5,1.,1.) rotation(0,180,0)

TEKTRONIX OUTPUT