May 29, 1973

To the Graduate Council:

I am submitting herewith a thesis written by Larry J. Minyard entitled "Minicomputer Implementation of a Functional Graphics Package." I recommend that it be accepted for nine quarter hours of credit in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

R C Gonzalez
Major Professor

We have read this thesis and recommend its acceptance:

Robert E Bakenheimer

Robert J Plemmons

Accepted for the Council:

Hilton A. Smith
Vice Chancellor for
Graduate Studies and Research

MINICOMPUTER IMPLEMENTATION OF A FUNCTIONAL GRAPHICS PACKAGE

A Thesis

Presented to

the Graduate Council of

The University of Tennessee

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

Larry J. Minyard

August 1973

ACKNOWLEDGEMENT

The author wishes to take this opportunity to express his appreciation to Dr. Rafael C. Gonzalez for his suggestion of the topic and his assistance in the clarification of the problems involved. The author would also like to express thanks to Mr. D. D. Thompson for his aid in the debugging of portions of the program. Finally, the author would like to express his special gratitude to his parents, Mr. and Mrs. Ben J. Minyard, and to his wife, Alice, without whose understanding and patient encouragement this graduate program would never have begun.

1104108

ABSTRACT

This thesis describes the development and implementation of a partially interactive, real-time functional graphics package for a minicomputer. The package was developed as a part of an image processing system involving the storage and analysis of digital pictures in two- or three-dimensional functional representations.

The development of the package involved the development and implementation of algorithms to process and display the digital images as functional graphs. The most important of these algorithms was the hidden-point algorithm devised to solve the hidden line problem for the limited set of functions to be displayed. This algorithm proved to be very effective and extremely short, a feature which was paramount because of the limited memory normally available with minicomputer systems.

The algorithms used to generate the functional graphs and some example output are described along with potential uses of the program.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION AND PROBLEM FORMULATION	1
II.	THE GEOMETRY OF GRAPHIC FUNCTIONS	8
III.	DEVELOPMENT OF THE GRAPHICS PACKAGE	18
IV.	USE AND DISPLAY CONSIDERATIONS	27
V.	CONCLUSIONS	34
BIBLIOGRAPHY		35
APPENDIXES		37
Appendix A		38
Appendix B		46
Appendix C		70
VITA		78

LIST OF FIGURES

FIGURE		PAGE
1.	The Image Processing System	6
2.	A Perspective Planar Projection	9
3.	The Hidden Line Conjecture	15
4.	Line Drawings	16
5.	The Hidden Line Array	16
6.	The Rotational Coordinates	21
7.	Summary Flow Chart	24
8.	The Quadrant Orientation	28
9.	Flow Chart	39
10.	The Function in Quadrant One	72
11.	The Function Raised for Viewing	73
12.	The Function in Quadrant Four	74
13.	A System Photograph	76
14.	The Photograph Processed and Displayed	77

CHAPTER I

INTRODUCTION AND PROBLEM FORMULATION

Whoever said "A picture is worth a thousand words" grossly underestimated the power of the pictorial medium. In many cases, a pictorial representation can be invaluable in terms of the information presented to the viewer. Often a pictorial, or graphical, representation can present intuitive or heuristic information which no amount of numerical description could portray. More and more, as our technological society becomes aware of this fact, pictorial or graphical forms are often replacing numbers or the written word. Even the familiar NO PARKING sign has been replaced by a simple, yet completely descriptive, graphical symbol.

Nowhere is the need for graphical communication more obvious than in the interaction between the computer and the man who uses it. The growing amount of data processed every day by computing systems around the world is enough to keep the population of a fair-sized state busy day and night just reading it, not to mention trying to apply it to the problems at hand. This staggering amount of written information, when coupled with the relatively slow rates of even modern peripheral equipment presents the computer scientist with a two-fold problem. First, conventional output devices simply cannot keep up with their associated computing machinery. And second, even at the output rates of these devices, the user cannot possibly digest and apply all the written

information which he is being presented. In recent years it has become a pressing problem to find some method of displaying some of this information in a more efficient manner.

This problem is particularly important to the growing number of minicomputer users. Until very recently, these people have been tied, due to economic considerations, to the slow and cumbersome teletypewriter as their primary means of communications with their computers. It is to this audience that the bulk of this thesis will be dedicated.

The problem of graphical display of information has been a major area of study for the past several years for those in the related disciplines of Computer Science and Electrical Engineering. Different workers have approached the problem from several different directions and have developed systems which involve many different types of hardware. Some have used hybrid analog-digital systems to display their output data. A good example of this approach to graphical display is the system developed by NASA for use in their space vehicle simulator. The simulator is an attempt to simulate the view an astronaut would have when looking out the window of a vehicle during such manuevers as take-off, landing, or docking with another vehicle.

Most users, however, have concentrated on developing all-digital display systems. This type of system has been developed for a number of varied applications. These include several systems for use in the area of computer-aided design. Large scale interactive graphic systems are currently in use by a number of companies involved in the design of such varied products as aircraft, integrated electronics, and computers

themselves. Another area of application of graphic systems is in the area of control of large scale systems of almost any type. These would include traffic control, factory management, and warehouse inventory systems.

No matter which problem a graphic system is applied to, there are certain common problems which must be solved in order to make the system efficient and simple to use. In large scale multipurpose systems, the worst of these problems fall under the categories of control programming (the control necessary to drive the interactive hardware being used), and graphic language programming (the software needed to communicate graphically represented information to the computer). The second of these problem areas is most common to all work, and less installation limited; therefore it has received the most attention.

Two basic approaches have been taken to solve the graphic language problem. The first is to try to adapt an existing higher level language to handle the graphical data presented to the computer by the interactive terminal. This approach has obvious limitations due to the complex data structures which must be developed in order to manipulate the graphical information. The second method is to develop a new language to handle the graphical data more efficiently. This approach has been used very satisfactorily in a number of instances. However, the time and difficulty involved in developing a special purpose language is often prohibitive. Only in those systems involving large-scale or long time span problems can this approach be justified.

The third area of common problems is in applications programming. This area, due to the scope of a particular problem, is in the somewhat

paradoxical situation of having shown the most useful and spectacular results, and yet having only scratched the surface of the problems which will be encountered. Each system which has been developed has solved certain common problems in a manner right for its own equipment and application, but the results are not in general directly applicable to the solution of the same problem on most other systems. This causes a great deal of inefficient duplication of effort among those engaged in the study of graphical display.

A large amount of the study done in computer graphics has been in the area of graphical display of three-dimensional functional representations of one- and two-dimensional functions. A variety of ways in which graphic output can be used to simulate the appearance of three-dimensional functions or objects with a two-dimensional picture have been developed. Up until now, however, the problem has almost always been considered to be in the realm of large, batch type, computing systems. The programs involved are normally written in a higher level language such as FORTRAN or PL/1 with little regard to storage requirements, and few attempts have been made to approach a real-time environment.

This thesis has approached the problem with the hopes of achieving a real-time, partially interactive, system involving a minicomputer with limited (16k words) storage capacity. This approach has necessitated a major change of outlook with respect to the primary problems involved in the display of digital functions, and has required that new, simplified versions of the existing algorithms be developed to solve those problems. The major portion of this thesis was involved with the development and

implementation of these new algorithms to achieve an efficient operational system.

The work reported in this thesis represents a subsystem of a larger project involved with the study of digital image processing. The system, as it now stands, contains a PDP-11-20 digital computer, a television camera system for use as an input to the computer and a PEP-400 storage device interfacing the computer to a color television monitor. The pictures taken by the camera are sampled by a conversion unit and the grey-level equivalent of the video signal is stored on magnetic tape. (See Figure 1.) The object of this thesis was to process the digitized photographs from the magnetic tape by converting the grey-level representation of the picture into a three-dimensional graphical form.

Chapter II will describe the problems involved in the geometry of graphic functions and the solutions to these problems developed for this system. In particular, the algorithms used to rotate the functional space with respect to the viewing plane, and the algorithm used to solve the hidden line problem will be presented.

Chapter III will describe the development of the graphics package as a comprehensive subsystem.

Chapter IV will discuss the use of the program and the abilities and limitations inherent in the program. Also, some consideration will be given to problems to which the program might be applied.

Chapter V will discuss the conclusions of the author and some suggestions for future additions to the program and the system which might increase its general usefulness.

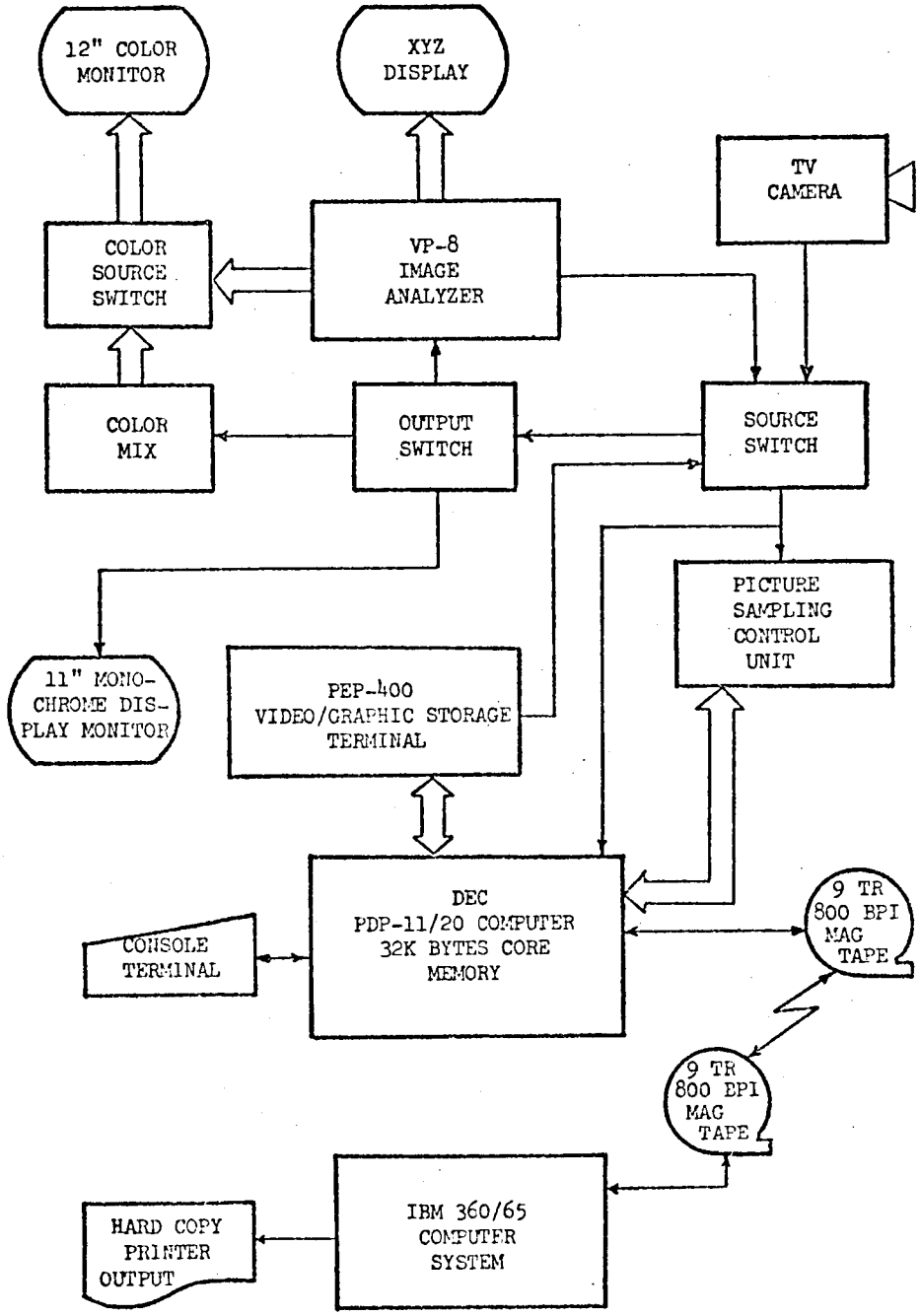

Figure 1. The Image Processing System

Included as appendices will be a flow chart and listing of the program and a test case for study as an example of how the program might be applied to a particular problem.

CHAPTER II

THE GEOMETRY OF GRAPHIC FUNCTIONS

Chapter I introduced the idea of graphical displays and described some of the problems involved in the use of a graphical output medium. It was stated that a large amount of the work in recent years has been done in the area of graphical display of three-dimensional objects and functions by attempting to simulate their appearance with two-dimensional functional representations. This chapter will describe some of the problems encountered in the attempt to simulate these three-dimensional figures in the two-dimensional world of the graphical device.

The task at hand is to simulate, in a three-dimensional field, the effect of the third dimension (i.e., distance from the point of view) which occurs when viewing a three-dimensional figure. The two-dimensional field may be paper or some electronic viewing surface, such as the screen of a cathode ray tube. The problem may be approached as one of calculating the projection of each point on the surface of a figure onto a plane, which can be observed by the user. This approach makes the problem of display basically one of descriptive geometry.

The geometry of graphic functions involves basically two problems, rotation and projection. Rotation is not an absolutely necessary feature but it is valuable in the sense that it allows the viewing of an object from more than one position. Projection, on the other hand, is the basic task of any graphic system.

A basic course in every engineering curriculum is Engineering Drawing or Graphics. One of the earliest problems studied in these courses is that of a perspective drawing, that is, representing depth perception in a two-dimensional drawing. The problem of projection in computer graphics is basically the same as that involved in a simple perspective drawing. Figure 2 shows a perspective planar projection.

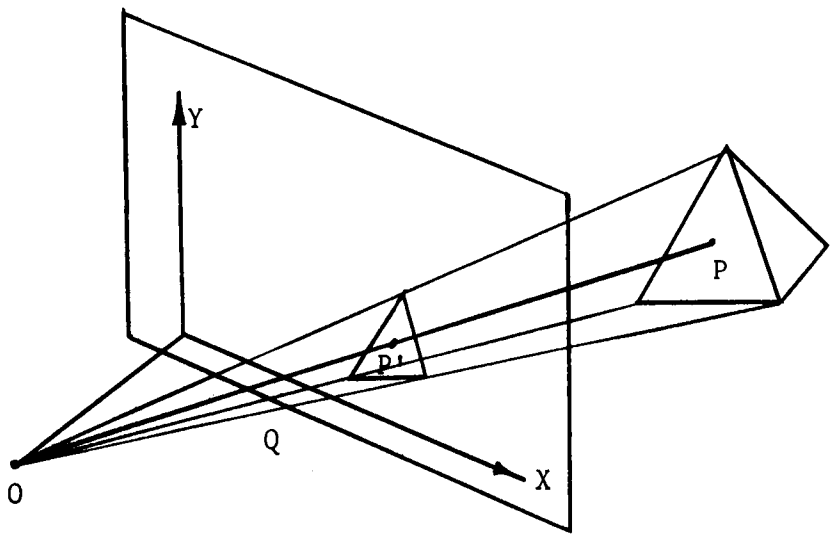

Figure 2. A perspective planar projection.

The perspective planar projection maps arbitrary points P_n on the surface of an object in space onto a plane V such that the lines of projection from all such points intersect at a common point O. The point O is the observation point from which the viewer will see the object. If we define a point $Q(q_x,q_y,q_z)$ and let it be the origin of an (X,Y) axis

system on the plane V such that $\overline{OQ}$ is normal to the plane, then, from geometrical considerations,

$$q_x = c_x + d(\cos \alpha),\ q_y = c_y + d(\cos \beta)\ q_z = c_z + d(\cos \lambda)$$

where α, β, and λ are the angles made by the line of sight with the x,y, and z axes respectively, and the distance d is the length of the line $\overline{OQ}$. Now, if we represent the point P', the projection of P on the plane V, as (P'_x, P'_y, P'_z) we have that

$$\frac{P'_x - c_x}{x - c_x} = \frac{P'_y - c_y}{y - c_y} = \frac{P'_z - c_z}{z - c_z} = K$$

or

$$(P'_x - c_x)\cos \alpha + (P'_y - c_y)\cos \beta + (P'_z - c_z)\cos \lambda = d$$

and

$$P'_x = c_x + K(x - c_x),\ P'_y = c_y + K(y - c_y),\ P'_z = c_z + K(x - c_z).$$

Now it remains to express P' in terms of the (X,Y) coordinate system of the plane V.

The unit vector, in the positive x direction, in terms of the original coordinates is

$$\overline{U}_x = ((\cos \beta)\bar{i} - (\cos \alpha)\bar{j}) / \sin \lambda$$

In order to define the unit vector in the y direction we use

$$\overline{U}_x \cdot \overline{U}_y = 0$$

and let

$$\bar{U}_y = a\bar{i} + b\bar{j} + c\bar{k}$$

Equation (1) requires that

$$a(\cos \alpha) - b(\cos \beta) = 0$$

and since $\overline{U}_y$ is in V and therefore must be normal to $\overline{OQ}$

$$a(\cos\alpha) + b(\cos\beta) + C(\cos\lambda) = 0 .$$

Combining these with the unit length of $\overline{U}_y$ yields

$$a = S_1 \cos\alpha \cos\lambda / \sin\lambda$$

$$b = S_1 \cos\beta \cos\lambda / \sin\lambda$$

$$c = -S_1 \sin\lambda$$

Since the point Q was defined to be the origin of the (X,Y) system in the plane V, the components of P' in the (X,Y) system will be

$$X = ((P'_x - q_x)\cos\beta - (P'_y - q_y)\cos\alpha) / \sin\lambda$$

$$Y = (P'_z - q_z) / \sin\lambda$$

Now, labeling the point O, (x_o, y_o, z_o) and including a scale factor to insure efficient use of the display device's output surface, the plane V may be considered to be the plane of the output surface. The coordinates of the point P' in the plane may then be written, in matrix form, as

$$\begin{vmatrix} X_p \\ Y_p \end{vmatrix} = \frac{\text{scale factor}}{Z_r} \begin{vmatrix} X_r \\ Y_r \end{vmatrix}$$

where (X_r, Y_r, Z_r) are defined as below

$$\begin{vmatrix} X_r \\ Y_r \\ Z_r \end{vmatrix} = \begin{vmatrix} -\sin\beta & \cos\beta & 0 \\ -\sin\alpha\cos\beta & -\sin\alpha\cos\beta & \cos\alpha \\ \cos\alpha\sin\beta & \cos\alpha\sin\beta & \sin\alpha \end{vmatrix} \begin{vmatrix} q_x - x_o \\ q_y - y_o \\ q_z - z_o \end{vmatrix}$$

Now, since the above transformation equations form a linear system, they may be broken up and applied in any order. The rotation of the X-Y

plane will be performed first and then the results of that transformation will be subjected to a rotation of the Y-Z plane. These two independent actions will combine to produce the desired final rotation of the function.

No consideration has been given, thus far, to whether or not all the points on the surface of the object would be visible to an observer at point O. When solid objects are considered, it quickly becomes intuitively obvious that some of the points on the surface of the object are going to be hidden by other points on the surface. That is, some parts of the surface will be hidden by those solid portions of the object between and the observer.

This leads to a consideration of the problem which has dominated the area of graphical display of three-dimensional figures for the past several years, the hidden-line problem. Two algorithms have emerged as very effective methods of solving the hidden-line problem for a general class of three-dimensional objects. These are the "key square" algorithm of Warnock and the "linescan" algorithm of Bourknight.

The principle of the key square algorithm is very simple. Take a hard problem and divide it into a large number of easy problems. First, the viewing plane is considered as a square picture. Either its contents are too complicated to process or they are sufficiently simple. If the contents of the square are too complex to process then subdivide the square into four subsquares and reapply the above question to each of the subsquares. This process is repeated recursively until one of two conditions holds for every square. Either the contents of the

subsquare are simple enough to process, in which case the subsquare is output to a display file, or the subsquare is smaller than the desired resolution of the picture, in which case the nearest object to the observer is output.

The decision procedure for each square is to determine whether there exists a planar figure such that two conditions hold. First, the square must be entirely in the interior of the planar figure. Second, the distances from the four corners of the square along the line of sight to the plane of the figure must be less than the distances to the plane of all those figures which have some intersection with the square when viewed through the square. If both of these conditions are met, the square is output, otherwise the square is subdivided.

This process of subdividing is actually a logarithmic search for points where there is a change in the characteristics of the picture.

A somewhat simpler algorithm is the linescan algorithm of Bourknight. First an attempt is made to break the figure to be displayed up into a set of polygonal surfaces. Once this approximation has been satisfactorily obtained, the coordinates of the corners of the polygonal surfaces are placed in a data file. A second file is set up containing the coefficients of the plane equations of the polygons which make up the figure in three-space. There are actually two forms of the plane coefficients data set. The set sent to the output routines contains the coefficients of the original polygon plane equations calculated in three-space. The set used by the linescan algorithm is calculated using the perspective coordinates for the vertices of the polygons on the viewing plane.

Now, the algorithm moves a scan line across the viewing plane in the horizontal direction. If, at any point along the scan line, an intersection with a polygon is encountered, the distance from that intersection to the observation point is calculated and placed in a save file. After a series of these scan lines have been run, the program processes the save file to determine which intersections occur in front of others, with respect to the observation point and thus which polygons are hidden from view. These polygons are discarded and all others are output.

These two algorithms have proven to be extremely effective solutions to the hidden-line problem for a very general class of three-dimensional figures. Both algorithms, however, involve the use of large amounts of fast memory in order to store the entire picture in core at once. Due to the fact that the functional arrays involved in this project are larger than the maximum amount of core available, the algorithms are not directly applicable. Therefore a new, rather simplistic, approach to the problem was developed which would allow implementation of a hidden-line algorithm on machines with limited memory, such as minicomputers.

Two factors were involved in the development of the hidden-line algorithm used in the program. The first was the use of the PEP-400 video signal storage device. The PEP-400 is a device which is used as an intermediate storage element between the computer and the television monitor used as a display device. Images can be written on the storage chip of the PEP-400 as a dot matrix of 1024 x 1024 points. This dot matrix can be built up a point at a time and, under program control, can be output to the monitor as a video signal.

This dot matrix feature allows lines to be written as a continuous string of individual points rather than requiring that they be comprised of line segments as on most conventional plotters. The ability to plot the function as a continuous string of separate points allows the simplification of the hidden-line problem to a less complicated "hidden-point" problem. A method has been devised which allows the solution of the hidden-point problem with a relatively short and simple algorithm.

The second factor upon which the hidden-point algorithm is based is a simple geometric conjecture. Stated simply, the problem involved is to determine which points on the surface of an object such as that of Figure 3, could be hidden by other points on the surface of the object.

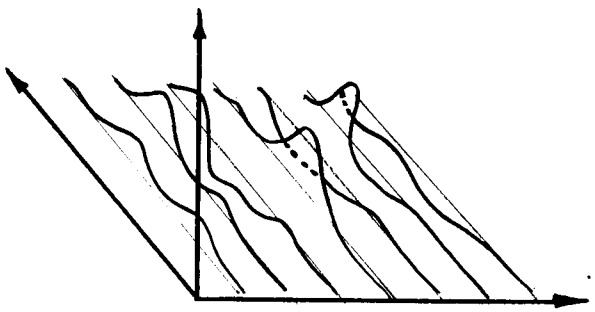

Figure 3. The hidden line conjecture.

The conjecture upon which the algorithm is based is quite simple. Suppose we are given a surface such as that of Figure 3 drawn as a series of dot matrix lines rather than segmented lines as those shown in Figure 4 at a given X value in the Y direction. If the rows are drawn in order of increasing X, i.e. away from the observation point,



Segmented Line

Dot Matrix Line

Figure 4. Line drawings.

then no point on any given row can ever be hidden by a point on a later row. This is obvious from the fact that the later rows are farther from the observation point and thus are behind those previously drawn. (See Figure 5.) The points on the later rows must either be higher than the

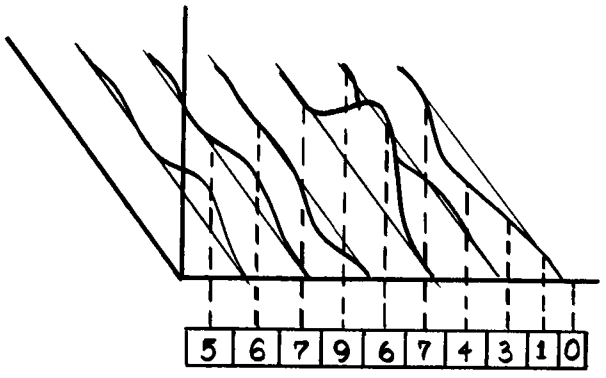

Figure 5. The hidden line array.

previous points, in which case they are visible. Or the points in the later rows must be lower than those previously drawn, in which case they cannot be seen, i.e. they are hidden by the points in front of and above them.

This conjecture allows the algorithm to "forget" all points that have been written except the highest previously drawn point at a given X-value.

The hidden-point algorithm uses two arrays, one representing the maximum Y-values previously defined at a given X-value, the other representing the minimum values. When a point's (X,Y) location on the screen has been computed by the rotational routine, it is passed to the hidden-point routine. There the point's Y-value is compared with the appropriate element of the maximum value array and if the point is higher than the previous maximum value, then the new point is output to the storage tube and the array element is replaced by the new point's Y-value. If the point is lower than the previous maximum then it is compared with the minimum value array element for the same X-value to determine if it might be visible below the function. If it is visible, it is output and the minimum value array is changed; if not the point is simply discarded and the program returns to the rotational routine for the next point.

This simplified routine functions well and is limited only by the requirement that the function must be plotted monotonically with respect to the depth axis of the function as seen by the viewer.

In the next chapter, the problems encountered in the development and implementation of these algorithms will be discussed.

CHAPTER III

DEVELOPMENT OF THE GRAPHICS PACKAGE

As implied in earlier chapters, there are certain inherent compromises involved in the use of a minicomputer as the processor of a graphics system. The very nature of a graphical display requires large quantities of data and a massive amount of computation to process the data. These requirements are not directly compatible with the design objectives of most minicomputer systems, in that, most minicomputers, as the name implies, are small systems. The amount of high speed core memory available is usually limited to a maximum of 64-k bytes and the purchase of expensive high speed mass storage devices defeats the initial purpose of buying a mini; that being its low price.

Any attempt to utilize a minicomputer for a graphical display system must then involve some simplification of the processing algorithms in order to achieve that most important of goals, speed of operation. This necessary simplification normally takes the form of limitations to the algorithm. It is obvious that an algorithm which is to process the same data in the same amount of time (or less) on a smaller system is going to involve some simplification of the processing. This simplification can usually only be achieved by means of some limiting constraints being placed either on the data or on the desired processing. It was with these constraints in mind that this thesis was begun.

Normally the most complex and time-consuming problem involved with

a display of graphical functions is the hidden-line problem and, as was previously mentioned, it is this problem which has received the most study. However, as stated in the previous chapter, the use of the PEP-400 allowed the simplification of the hidden-line problem to the less complex hidden-point problem with no consideration being given to the computer being used. This, combined with the restrictive assumption of a single-valued function being drawn monotonically with respect to the observation point, permitted the development of an extremely simple, and fast, hidden-point algorithm. This algorithm greatly increased the overall processing speed of the program. This increase in processing speed was fortunate in that the difficulty encountered in the implementation of the relatively simple process of axis rotation tended to slow the program down by a considerable factor.

The problem of rotation is, on a large system using a higher level scientific language, a relatively straightforward problem. It is one that involves simply the application of the rotational equations derived in Chapter II. However, the implementation of these equations on a minicomputer where one is, of necessity, limited to assembler level programming, is far from straightforward.

The primary problem involved in the implementation of an effective rotational algorithm on a minicomputer lies in the need for floating point arithmetic to perform the operations required by the equations. The trigonometric formulas involved in the direct applications of the rotation equations require a mixing of large and small (i.e. fractional) numbers which cannot be achieved without floating point arithmetic. The

problem is that most minicomputers either have no floating point arithmetic or they offer only a subroutine package to perform the necessary operations. Any attempt to use such a package would be quickly abandoned when it is realized that the rotation of a 512 x 512 picture, for example, can involve more than two hundred operations on each element of the picture. This is approximately 54 million operations just for the rotation.

The problem of rotation was solved by another compromise. Taking advantage of the geometry of the dot matrix on which the output is being written on the storage tube of the PEP-400, allowed the development of two incremental rotational algorithms which involved only small, fixed point numbers. This compromise still produced reasonably complete rotational algorithms.

The rotational algorithms use the fact that only approximately one-fourth of the points in the dot matrix are needed to draw line figures to allow incremental rotation. The points in a matrix such as that of Figure 6, for instance, can be illuminated so as to allow a line drawn from point A to points B to lie at any of a number of angles to the horizontal. These possibilities are combined by the algorithms to allow the rotation of a figure in approximately twelve degree increments throughout the full 360 degrees of the X-Y plane.

Although both algorithms use the same geometrical properties as a basis, the results produced by the two are quite different. The first algorithm is based on a single application of the properties of Figure 6 and will be called the shifting algorithm. The second applies those

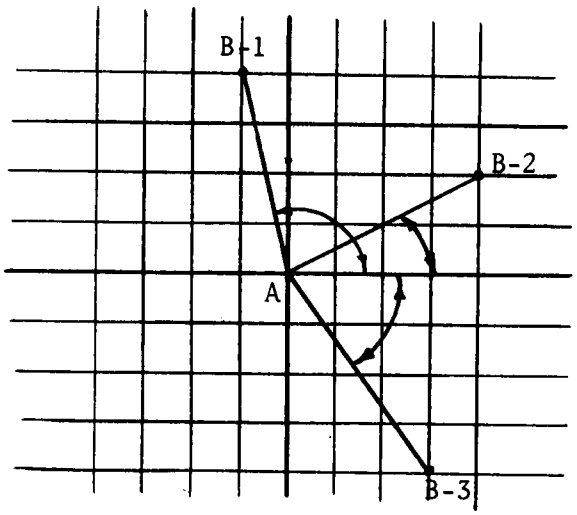

Figure 6. The rotational coordinates.

same properties twice in a somewhat more complex combination. The second algorithm will be referred to as the rotating algorithm.

The shifting algorithm places the frontal or X-Z plane of the function in the plane of the screen and approximates rotation by shifting sequential rows of the function up or down and left or right as appropriate. This algorithm thus always presents the same "face" to the viewer and distorts the shape of the X-Y plane of the function as it is rotated through the various angles. This algorithm effectively approximates rotation by merely distorting the Y-axis of the function space.

The rotating algorithm, on the other hand, more nearly approximates general rotation by operating on all three of the functional axes and maintaining the basic shape of the X-Y plane of the function. The

rotating algorithm is more restricted than the shifting algorithm in that it is not as compatible with the simplified hidden line algorithm described before. This incompatibility limits the angles of rotation which may be specified when using the rotating algorithm to those not less than 10 degrees away from the central angle of rotation; 45 degrees. This restriction is lessened, however, by the practical fact that these angles would be poor viewing angles if they were available and would seldom be used.

In all, the compromises required by the implementation of an efficient functional graphics package consisted of two very tight constraints; size and speed. As it turned out, the size constraint had a direct bearing on the development of the hidden point algorithm, while the speed constraint primarily restricted the rotational algorithm.

In summary, the operation of the completed program is described by the flowchart at the end of this chapter. The program begins by conducting a dialog with the user to determine which picture is to be displayed and how it is to be displayed (see Chapter IV). With this information on hand the program then searches the available tape for the appropriate picture and calculates the rotational constants needed.

When the desired picture file has been found, the program determines the picture type and size and uses these to define more processing constants. Then the program begins to prepare the system to process the picture. The hidden point arrays are cleared and the screen of the display unit is blanked. Now the processing of the picture can begin.

The first record of the picture is read from the tape into memory and while this is in progress the axes of the function space are rotated

and written of the storage tube on the PEP-400. When the first record is in memory and the axes have been written, the next record is read in and the processing of the picture records proceeds.

In order to process a point of the picture, first the (x,y,z) coordinates of the point are computed. These values are processed through the rotational algorithm where their rotated location in the plane of the display are calculated. The point is then passed to the hidden point algorithm where it is compared against the values in the MAXVAL and MINVAL arrays to determine whether or not it is visible. If the point is visible, it is passed then to the output routine to be written on the storage tube. The program now returns to the beginning of the processing area and starts over with another point.

After all the points in one record have been processed, and, when the tape unit has completed with the reading of the next record, another record is read in and the new record is processed as before.

After the last record has been processed, the picture which has been written on the storage tube is transferred to the screen of the display and the program returns to its starting point and the user dialog is reinitiated.

See Figure 7 for a summary flow chart.

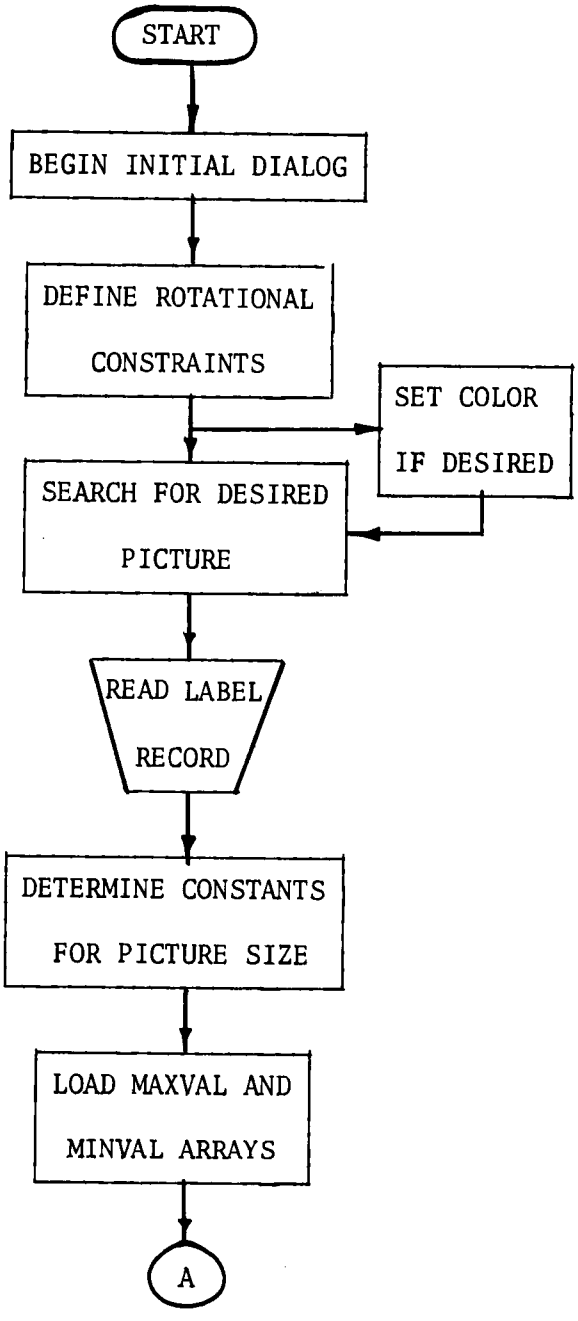

Figure 7. Summary flow chart.

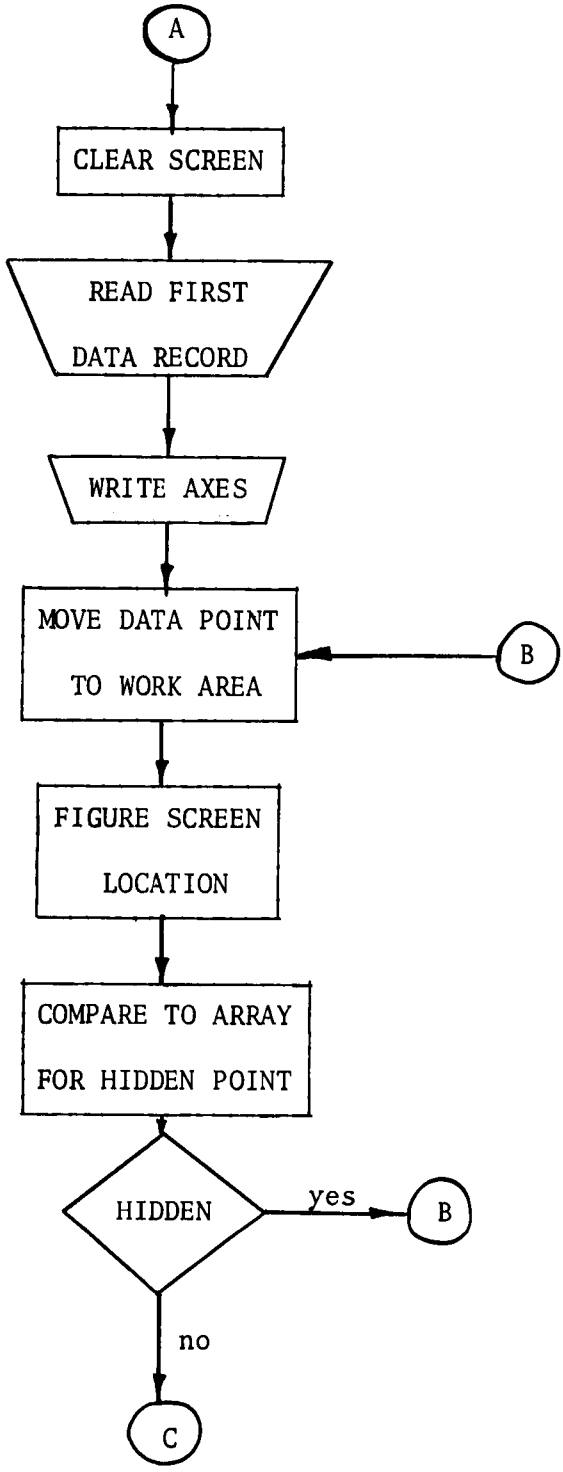

Figure 7 (continued)

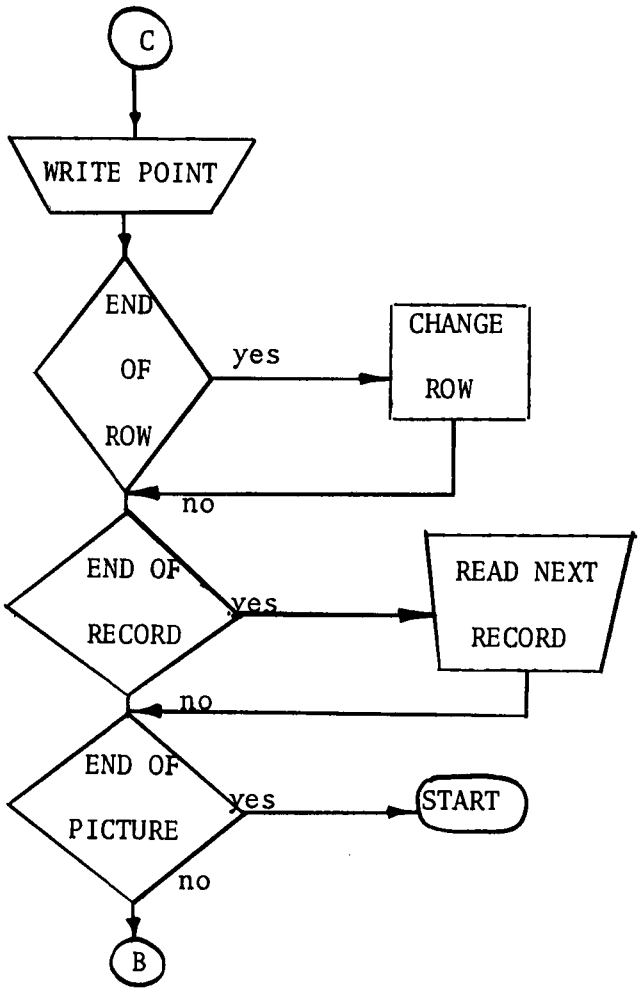

Figure 7 (continued)

Detailed Flow Chart available in Appendix B.

CHAPTER IV

USE AND DISPLAY CONSIDERATIONS

The previous chapters have discussed how the program works and how it was developed. In this chapter the instructions for the use of the program will be outlined.

The use of the program is a simple process. The program is stored in the Image Processing System tape under the name GRAPH. In order to start the program one has merely to start the system and to type, on the teletype, the word GRAPH. The program is self-starting and will respond "FROM?". At this point the user must specify the quadrant and angle from which the function is to be observed.

The desired quadrant and rotation angle for display of the function should be chosen according to the options shown in Figure 8.

After the angle has been specified, the program will respond "VIEWING LEVEL?". This refers to the apparent height from which the picture is to be observed. The user may respond with any number between 1 and 6, corresponding to a range of angles between approximately 10 degrees and approximately 60 degrees with respect to the horizontal. If the standard angle of 45 degrees is preferred, the user may simply type RETURN.

The next portion of the dialog concerns the optional use of an automatic scaling function. The program will type "SCALING FUNCTION?" and the user may respond "Y" if the scaling option is desired, or may

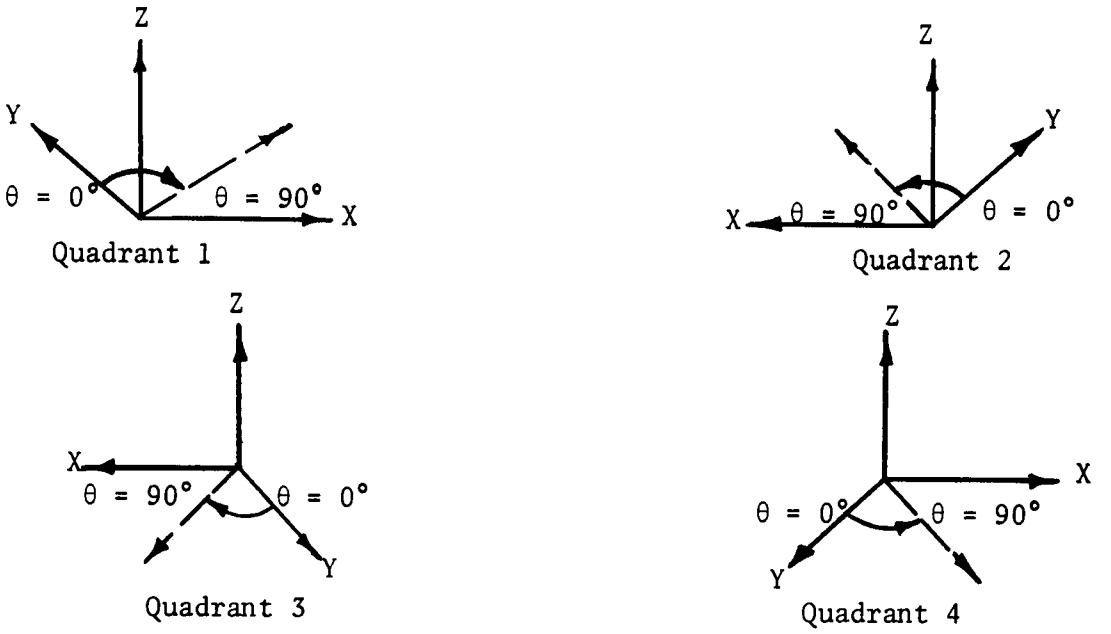

Figure 8. The quadrant orientation

simply type RETURN if the option is not needed. The automatic scaling function, if requested, will determine the largest value in the picture and scale all the data values so as to present the function within the range of the screen. If the scaling function is chosen the program will respond with "THE SCALE FACTOR IS xx".

If the automatic scaling option is not chosen, the program will respond "SCALE FACTOR?". This allows the user to specify the desired scale factor to be used. If no scaling is desired the user merely again responds RETURN.

The program will now respond "SHRINK?". This option will allow the user to reduce the size of the total function space as it appears on the screen. This option may be useful for functions which contain extreme variations in the magnitude of its data points, or when using higher

elevation angles to view a function. If the user types "Y", the size of the function will be reduced by approximately one-fourth. If this reduction in size is not desired the user should respond RETURN.

Next the program will type "MOVE X?". This option will allow the user to move the function to the left or right on the screen by typing an "L" or an "R" respectively. If no horizontal movement is desired the user may simply type RETURN. The program will then respond "MOVE Y?". Now the user may move the function up or down on the screen by typing "U" or "D" respectively. Again a response of RETURN will cause no movement of the function.

The program will next type "ROTATION?". The user must now specify which rotation algorithm is to be used to display the function. If the shifting algorithm is desired, the user should respond "S". If the rotating algorithm is desired, the user should type an "R".

After the scaling dialog has been completed, the program will respond "FILE". At this point the user may specify the file number of the function to be displayed. If the file number is not known the user may type only RETURN.

The program will next respond "FUNCTION?". The user may now specify the name of the function to be displayed. (The format for naming and writing the functions on tape will be specified below.) If the name of the function to be displayed is not known the user may again respond RETURN. However, either the file number or the function name must be specified in order for the program to find the correct file to display.

After the user responds with the function name, the program will begin execution and will process and display the desired function. The

processing and display of a function takes approximately 15 seconds. After displaying the function, the program will again respond "FROM?" The user may now specify a new set of viewing parameters or may respond RETURN to all inquiries except the scaling option. The scaling parameters must be redefined for each display. The displayed function will remain visible on the screen for approximately 10 minutes before any refreshment of the tube is necessary.

The functions to be processed by this program may either be pictures taken by the system camera and written on the system tape, or may be any other square array stored on a compatible tape in a format as specified below.

The first record of the tape is a 4-k byte label record. The format of the label record is described in Tables I and II. Following the label record will be the function itself. The function should be stored, in column order, starting at the upper left corner and proceeding down and then right, in 4-k byte records. The data should be in the form of two's complement binary numbers representing the value of the function at each point along the columns of the function array.

If desired, the program has the capability of displaying a function in a two-dimensional rather than three-dimensional form. This form of display will provide the user with essentially a frontal silhouette view of his function. In order to use the two-dimensional option, the user should respond to the inquiry "FUNCTION?" with the phrase TWODD. The program will then respond "FUNCTION?" again and the user may continue the dialog as before with the results being that the desired function will be displayed in the two-dimensional form described above.

TABLE I

LABEL RECORD FORMAT

Byte Number	Contents
0,1	File number, in binary
2,3	File type, see Table II
4,5	Record length, in binary; 4096 required
6,7	Record count, excluding label record, in binary
10-17	File name, Eight ASCII characters, 0 parity, left justified, no imbedded blanks
20-21	ASCII carriage return, line feed
22-41	Creation/Expiration dates
42-77	Binary zeros
100-7777	Picture title, or comments, in ASCII

TABLE II

FILE TYPE CODES

Byte 2	Byte 3
P - application programs	S - 128 x 128 picture
O - original picture	M - 256 x 256 picture
C - complex, spatial	L - 512 x 512 picture
F - Complex, frequency	
T - temporary, for FFT	

One other display option which is currently available to the user is the use of the color monitor. If the color monitor is being used, the user should respond to the "FUNCTION?" inquiry the first time with the word COLOR. The program will then respond "AXES?", and the user may specify, using the color codes of Table III, the number for the color to be used to write the axes of the function. After the user responds to the "AXES?" inquiry, the program will respond "PICTURE?", and the user should specify the color desired for the function. If a number is specified that is between two values in the table, the program will respond with a shade between the two corresponding colors.

Both the TWODD and the COLOR options may be used for the same function, however, if either is to be used, it must be requested before the function file name is specified.

The system is also designed so as to allow hard copy output from the program. The camera mounts on both monitors are to be used with a camera and may be used to obtain photographs of the functions displayed by the program. See Appendix C for some example photos and a detailed sample dialog.

TABLE III

COLOR CODE

Code Number	Color
037	Red
077	Orange
137	Yellow
177	Green
237	Blue
277	Cyan
337	Violet
377	Magenta

CHAPTER V

CONCLUSIONS

The development of this program has shown that a functional graphics package is easily within the realm of any reasonably well-equipped minicomputer system. The program has demonstrated areas where compromises must be made, either for reasons of speed and complexity or because of limitations in the high-speed memory availability.

One of the problems pointed up by this thesis, although only implicitly, is the inflexibility of the input medium of this system. More work might be warranted in the development of an efficient function generator for minicomputer users.

Another weakness of the program is in the area of the general rotation of the output functions. An efficient algorithm might be developed to produce a general rotation capability with the minicomputer system. However, such an algorithm would necessitate a much more sophisticated hidden line algorithm which might negate its overall efficiency somewhat.

Even in the admittedly limited form which this program now exists, however, it has been demonstrated that a three-dimensional functional representation of two-dimensional functions can be very valuable in the detailed study of the functions, and that such a system can be developed for, and implemented on, a minicomputer.

BIBLIOGRAPHY

BIBLIOGRAPHY

Averat, Edward. "A Three-Dimensional Representation of the Digital Output of a Microdensitometer" (unpublished Master's thesis, University of Tennessee, Knoxville, 1971).

Bourknight, J. "A Procedure for Generation of Three-Diemsnional Half-Tone Computer Graphics Presentations," Communications of the ACM, Vol. XIII, No. 9 (September, 1970), 527-536.

Faiman, M., and J. Nievergelt (ed.). Pertinent Concepts in Computer Graphics. Urbana, Illinois: University of Illinois Press, 1969.

Fetter, William A. Computer Graphics in Communications. New York: McGraw-Hill, 1964.

Fisher, Robert, and Allen Ziebur. Calculus and Analytic Geometry. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1965.

Kreyszig, Erwin. Advanced Engineering Mathematics. New York: John Wiley and Sons, 1967.

Kubert, B., J. Szabo, and S. Guilieri. "The Perspective Representation of Functions of Two Variables," Journal of ACM, Vol. XV, No. 2 (April, 1968), 193-204.

Proceedings of the IBM Scientific Computing Symposium on Man-Machine Communication. White Plains, New York: IBM Data Processing Division, 1966.

The Digest of the 1969 Joint Conference on Mathematical and Computer Aids to Design. Anaheim, California: ACM/SIAM/IEEE, 1969.

Tucker, T. C. CATCH: Computer Assisted Topography, Cartography, and Hypsography Part III, PERSPX: A Subroutine Package for Perspective and Isometric Drawings. Oak Ridge, Tennessee: Oak Ridge National Laboratory, ORNL-TM-3790, 1973.

APPENDIXES

APPENDIX A

The following (Figure 9) is a detailed flow chart of the logical operation of the functional graphics package.

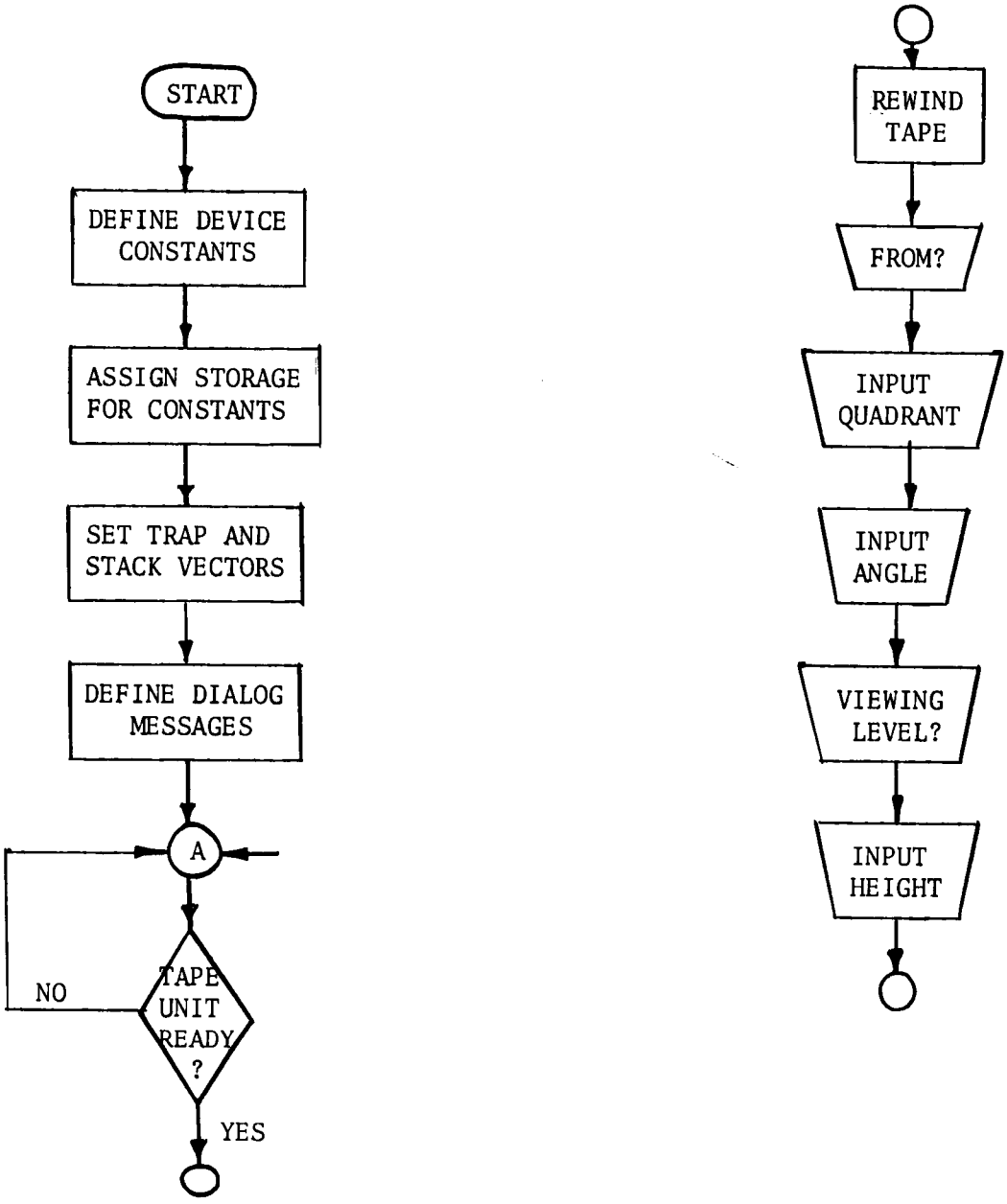

Figure 9. Flow chart.

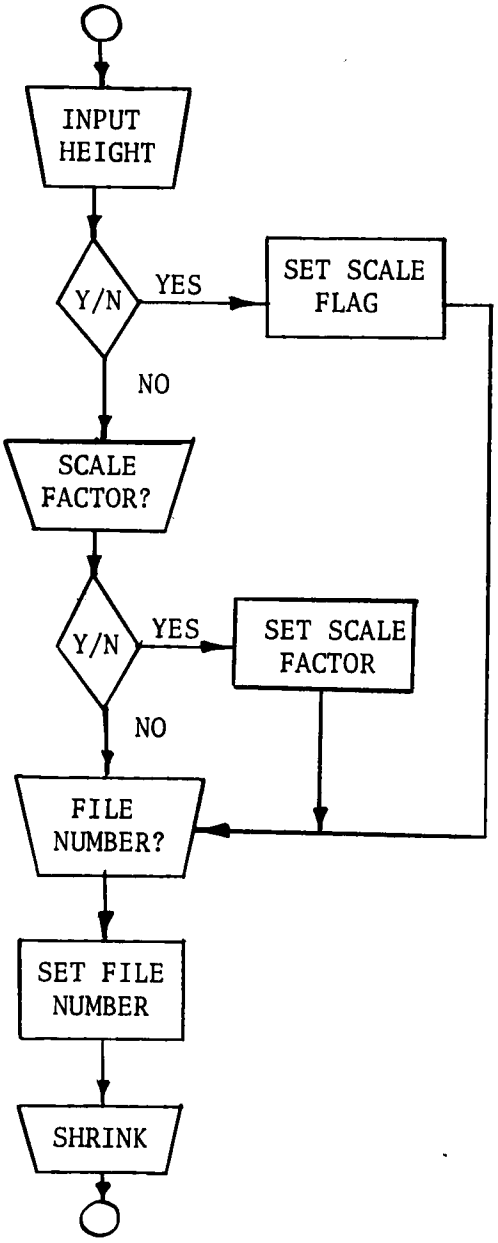

Figure 9 (continued)

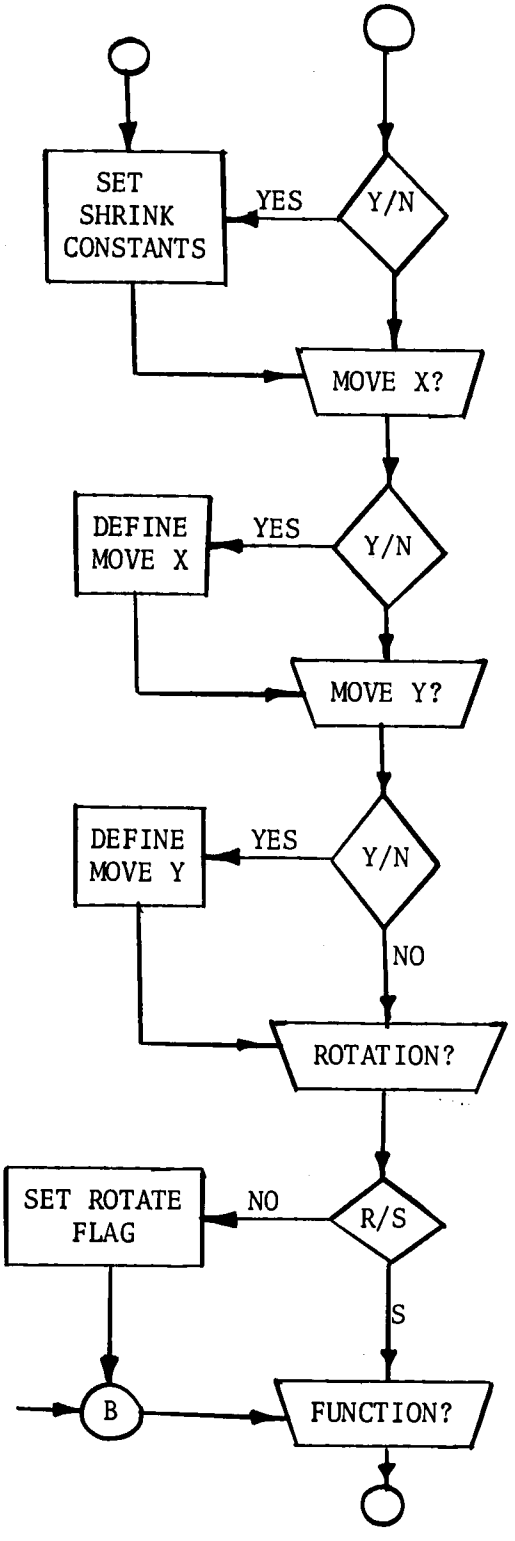

Figure 9 (continued)

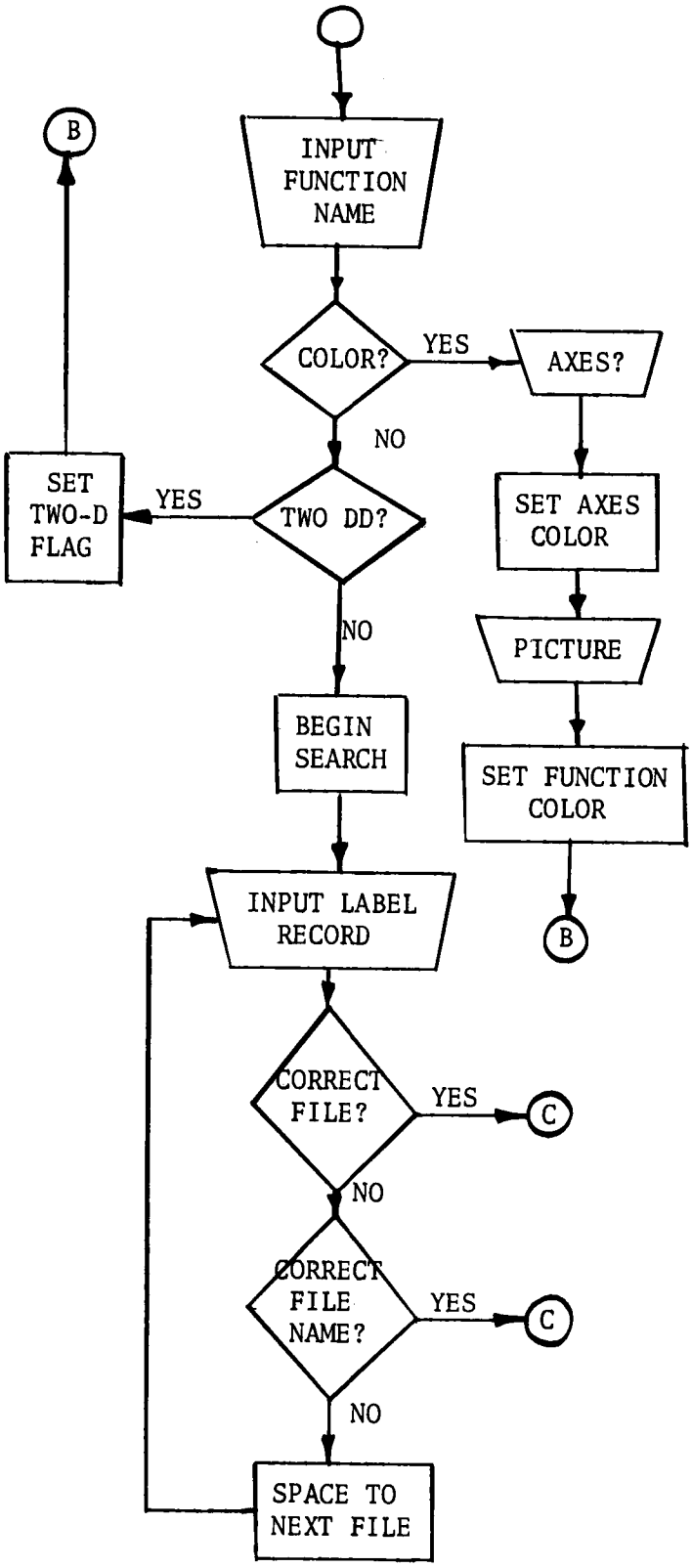

Figure 9 (continued)

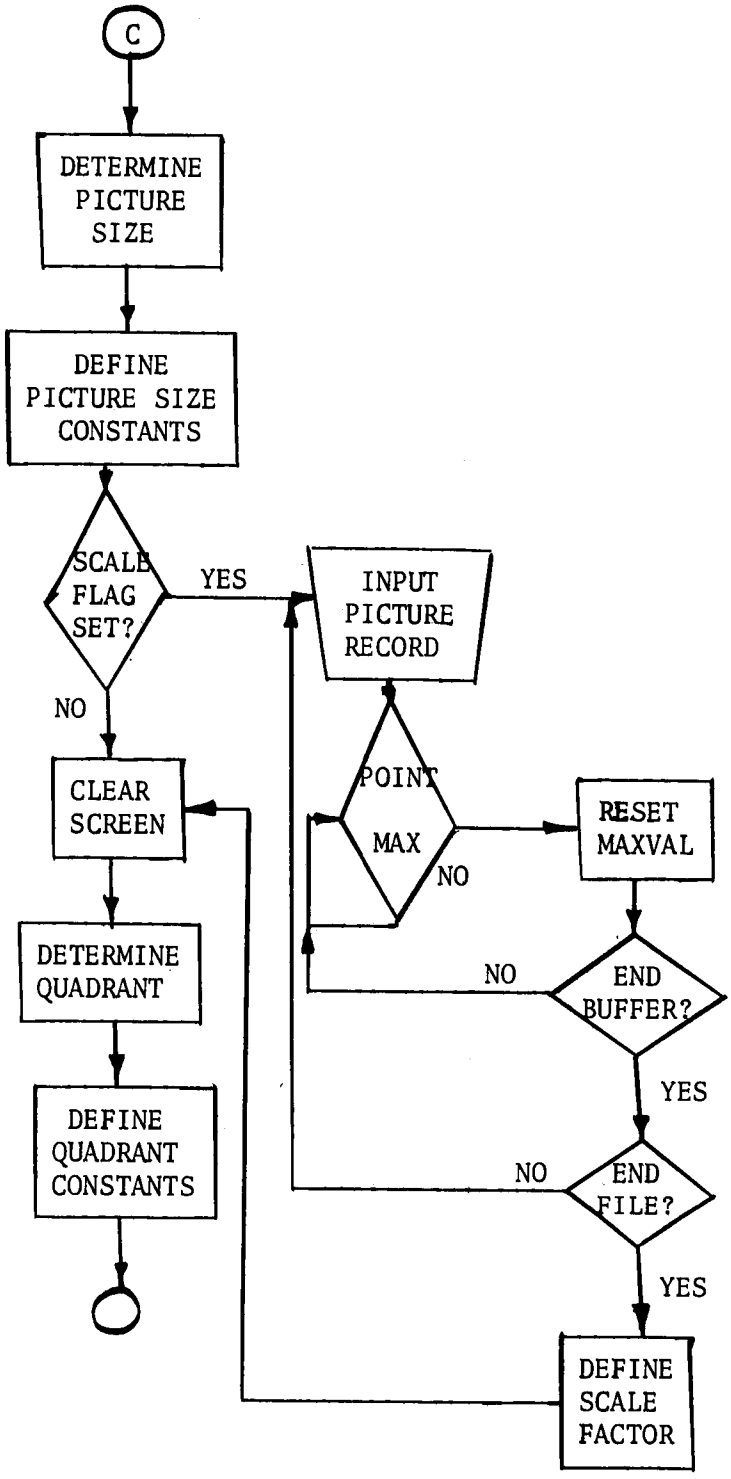

Figure 9 (continued)

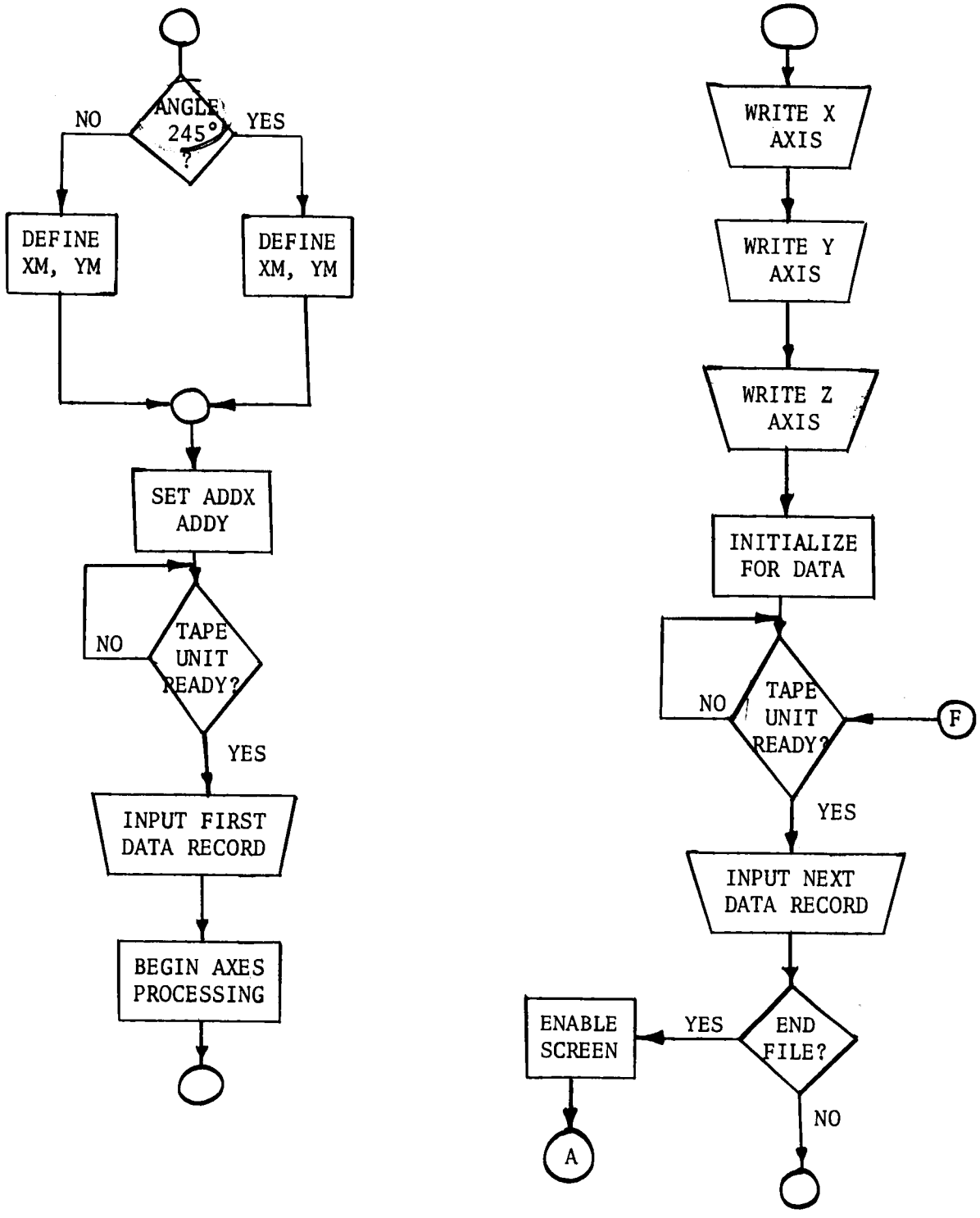

Figure 9 (continued)

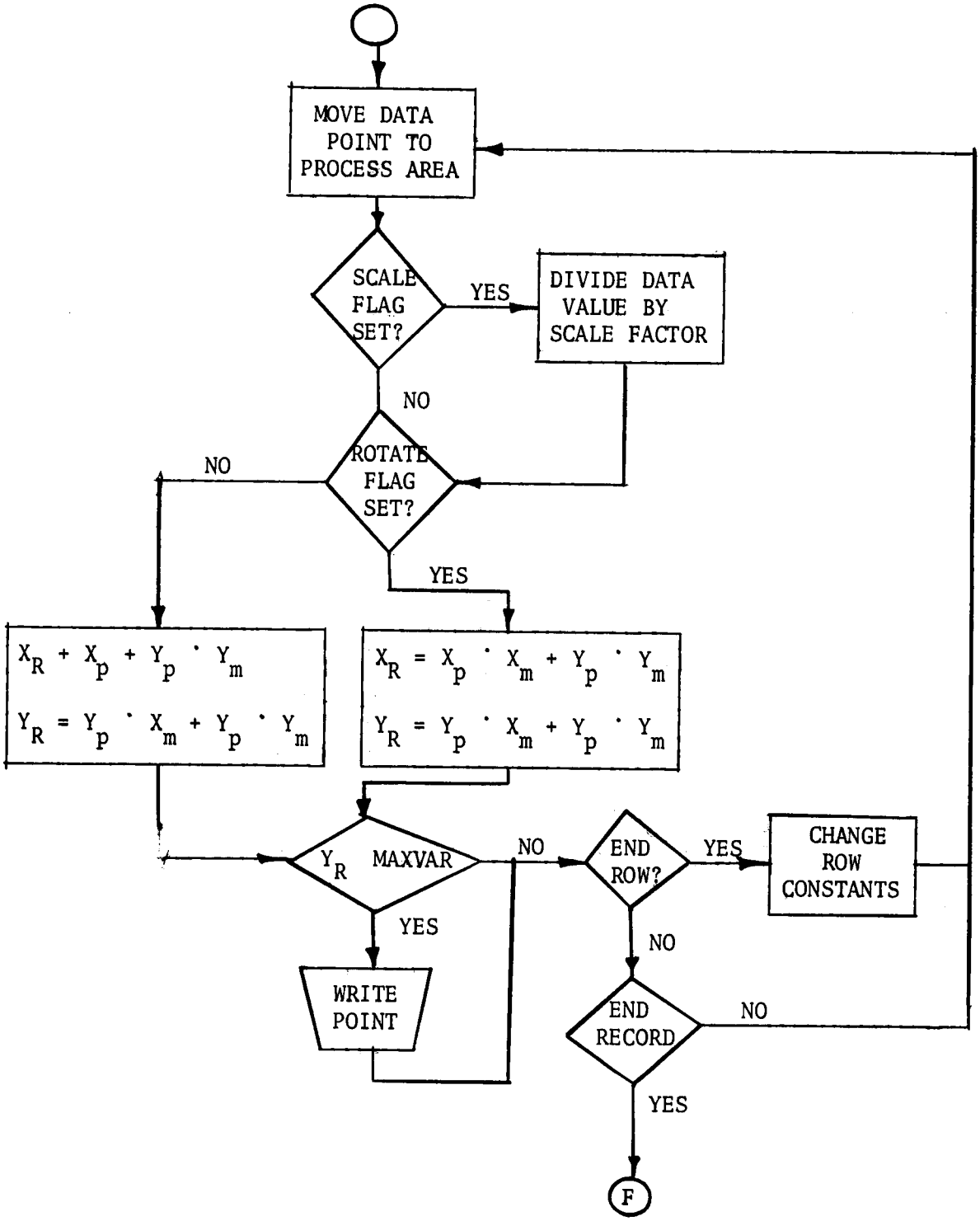

Figure 9 (continued)

APPENDIX B

```
;  UNIVERSITY OF TENNESSEE IMAGE PROCESSING SYSTEM
;  FUNCTIONAL GRAPHICS SOFTWARE PACKAGE
;
;  VERSION  V001
;
;  PROGRAMMER  LARRY J. MINYARD
;
;  CREATION DATE  JULY 1,1973
;
;
;     THE FUNCTIONAL GRAPHICS PACKAGE IS A SUBSET
;  OF THE UNIVERSITY OF TENNESSEE IMAGE PROCESSING
;  SYSTEM.  THE PACKAGE WAS DESIGNED TO ENHANCE
;  THE EXISTING SYSTEM BY INCREASING ITS GRAPHIC
;  CAPABILITIES.
;
;  INPUT MEDIUM  THE PROGRAM IS DESIGNED TO INPUT
;  DATA FROM MAGNETIC TAPE.  THIS DATA SHOULD BE
;  IN THE FORM OF TWO'S COMPLEMENT INTEGER NUMBERS
;  STORED IN 4-K BYTE RECORDS.  EACH DATA FILE IS
;  TO BE PRECEEDED BY A 4-K BYTE LABEL RECORD.
;
;  OUTPUT MEDIUM  THE PROGRAM DISPLAYS ITS OUTPUT
;  ON THE IMAGE PROCESSING SYSTEM'S BLACK AND
;  WHITE OR COLOR TELEVISION MONITORS.  THE OUTPUT
;  IS PROCESSED THROUGH THE PEP-400 VIDEO STORAGE
;  UNIT AND DISPLAYED AS A DOT MATRIX.
;
;  PROCESSING TIME  THE PROCESSING AND DISPLAY OF
;  A PICTURE FILE TAKES APPROXIMATELY 15 SECONDS
;  AFTER THE FILE HAS BEEN LOCATED ON THE TAPE.
;
;
```

```
            R0=     %0
            R1=     %1
            R2=     %2          ;  DEFINE
            R3=     %3
            R4=     %4          ; REGISTERS
            R5=     %5
            SP=     %6          ; STACK POINTER
            PC=     %7          ; PROGRAM COUNTER
            MTS=    172520
            MTC=    172522      ;  MAG TAPE
            MTBRC=  172524      ;  REGISTERS
            MTCMA=  172526
            TKS1=   177560
            TKB1=   177562      ;  TELETYPE
            TPS1=   177564      ;  REGISTERS
            TPB1=   177566
            DIV=    177300
            AC=     177302      ;  MULTIPLY/DIVIDE
            MQ=     177304      ;  REGISTERS
            MUL=    177306
            BUFFR1= 20000       ;  INPUT DATA
            BUFFR2= 30000       ;  BUFFERS
            DABX=   176764
            DABY=   176762      ;  PEP-400
            DAB1=   176760      ;  REGISTERS
            DRSCAN= 177522
            CLKCSB= 172542      ;  CLOCK
            CLKCSR= 172540      ; BUFFERS
            .=0
            JMP @#1200
            .WORD .+2,0,.+2,0,.+2,0,.+2,0  ; SET TRAP
            .WORD .+2,0,.+2,0,.+2,0,.+2,0  ; VECTORS
            .=600
SHIFT:      0                   ; RAISE/LOWER FUNCTION
NUMSIZ:     0
SKPCNT:     0                   ; SKIP COUNTERS FOR CROSS-
NUMSKP:     0                   ; HATCHING
PICSIZ:     0                   ; PICTURE SIZE
BUFFIN:     0                   ; INPUT BUFFER POINTER
ROWNUM:     0
PNTNUM:     0                   ; X,Y,Z COORDINATES OF POINT
POINT:      0
BUFWRK:     0                   ; WORK BUFFER POINTER
CNTNUM:     4
PNTCNT:     1                   ; SPACING COUNTERS
KOUNT:      0
PROD:       0                   ; TEMPORARY
TEMP:       0                   ; WORK LOCATIONS
HOLD:       0
XFROM:      0                   ; DISPACEMENT COUNTERS
```

```
YFROM:    0                    ; FOR ORIGIN OF FUNCTION
XROT:     0
YROT:     0                    ; ROTATED X,Y,Z COORDINATES
ZROT:     0
XPVAL:    0                    ; HIDDEN POINT COUNTERS
TSTVAL:   0
CAXES:    1774                 ; COLOR OF AXES
CFUNCT:   1774                 ; COLOR OF FUNCTION
NSWITCH:  0
FILNUM:   0                    ; FILE NUMBER POINTER
QUADRT:   0                    ; QUADRANT NUMBER
RANGLE:   0                    ; ROTATION ANGLE
FORREV:   0
TWDCNT:   -1                   ; TWO-D COUNTER
XMULT:    0
YMULT:    0                    ; ROTATION CONSTANTS
XVAL:     0                    ; SCREEN X,Y
YVAL:     0                    ; COORDINATES
SPCONT:   0                    ; SMALL PICTURE FLAG
PICCNT:   0
SCALNT:   -1                   ; SCALE FACTOR FLAG
COUNTER:  1                    ; SCALE FACTOR
LEVEL:    4                    ; VIEWING LEVEL
AXLAST:   0                    ; AXES LAST FLAG
FROM:      .BYTE 15,12
           .ASCII /FROM?/
NFUNCT:    .BYTE 15,12
           .ASCII /PICTURE?/
NAXES:     .BYTE 15,12
           .ASCII /AXES?/      ; ASCII DIALOG CHARACTERS
FILL:     .BYTE 15,12
          .ASCII /FILE?/
LITTLE:   .BYTE 15,12
          .ASCII /SHRINK?/
MOVEX:    .BYTE 15,12
          .ASCII /MOVE X?/
MOVEY:    .BYTE 15,12
          .ASCII /MOVE Y?/
ROTPIC:   .BYTE 15,12
          .ASCII /ROTATION?/
HEIGHT:   .BYTE 15,12
          .ASCII /VIEWING LEVEL?/
SCALOT:   .BYTE 15,12
          .ASCII /SCALING FUNCTION?/
SCAMES:   .BYTE 15,12
          .ASCII /THE SCALE FACTOR IS/
SCANUM:   0
SCAFAC:   .BYTE 15,12
          .ASCII /SCALE FACTOR?/
PICT:      .BYTE 15,12
```

```
          .ASCII /FUNCTION?/
          .BYTE 15,12
          .EVEN
NAME:     .WORD , , , , , , ,
COLCD:    .WORK 'O,'L,'C,'R
TWODCD:   .WORD 'W,'O,'D,'D
          .=1200
BEGIN:    MOV #500,SP           ; SET TRAP VECTOR
START:    BIT #40,MTS           ; TAPE UNIT READY?
          BNE .+10              ; WAIT-
          MOV #60017,MTC        ; REWIND TAPE
          MOV #1,NSWITCH
          MOV #1,SPCONT
          CLR PICCNT
          MOV #1,AXLAST
          MOV #1,SCALNT
          MOV #1,COUNTER
          MOV #240,MOVRT
          MOV #240,MOVPT+2
          MOV #240,MOVUP        ; CLEAR THE PICTURE
          MOV #240,MOVUP+2      ; MOVING CONSTANTS
          MOV #6367,SHKPC1
          MOV #167312,SHKPC1+2  ; CLEAR THE SHRINK
          MOV #6367,SHKPC2      ; OPTION CONSTANTS
          MOV #167274,SHKPC2+2
          BIC #100000,ADDXRT
          BIC #100000,ADDYRT
          MOV #6367,TYPFOR
          MOV #170170,TYPFOR+2  ; SET FOR NORMAL OPERATION
          MOV #6367,TYPFR1
          MOV #170166,TYPFR1+2
          MOV #4,CNTNUM         ; SET FOR NORMAL OPERATION
          MOV #1,TWDCNT
          CLR QUADRT            ; ZERO QUADRANT
          CLR RANGLE            ; AND ANGLE
          MOV #FROM,R0
          MOV #11,TEMP          ; BEGIN DIALOG 'FROM?'
          JSR PC,WRTY
          MOV #QUADRT,R1
QUAD:     JSR PC,RDTY           ; READ QUADRANT NUMBER
          CMP #15,R0
          BEQ HLEVEL            ; CONTINUE DIALOG
          CMP #', ,R0
          BEQ ANGLE             ; GO TO ANGLE
          BIT #10,R0
          BNE STRIP
          BIC #-10,R0           ; QUADRANT DIGIT
          BIS R0,QUADRT         ; LESS THAN SEVEN
          BR QUAD
STRIP:    BIC #-20,R0           ; QUADRANT DIGIT
```

```
          BIS RO,QUADRT          ; GREATER THAN SEVEN
          BR QUAD
ANGLE:    JSR PC,RDTY            ; READ ANGLE DIGIT
          CMP #15,RO
          BEQ HLEVEL             ; CONTINUE DIALOG
          BIT #10,RO
          BNE STRANGL
          BIC #-10,RO            ; ANGLE DIGIT
          ASL RANGLE             ; LESS THAN SEVEN
          ASL RANGLE
          ASL RANGLE
          BIS RO,RANGLE
          BR ANGLE
STRANGL:  BIC #-20,RO            ; ANGLE DIGIT
          ASL RANGLE             ; GREATER THAN SEVEN
          ASL RANGLE
          ASL RANGLE
          BIS RO,RANGLE
          BR ANGLE
HLEVEL:   MOV #HEIGHT,RO         ; ASK FOR VIEWING LEVEL
          MOV #22,TEMP
          JSR PC,WRTY
HLEVIN:   JSR PC,RDTY            ; INPUT VIEWING LEVEL
          CMP #15,RO
          BEQ SCALER
          BIC #-10,RO
          MOV RO,LEVEL
          BR HLEVIN
SCALER:   MOV #SCALOT,RO         ; SCALING OPTION DESIRED ?
          MOV #25,TEMP
          JSR PC,WRTY
SCALIN:   JSR PC,RDTY
          CMP #'Y,RO             ; OPTION DESIRED
          BNE SCALE2
          MOV #-1,SCALNT         ; SET SCALING FLAG
          BR REDUCE
SCALE2:   MOV #1,SCALNT
          CLR COUNTER
          MOV #SCAFAC,RO         ; MANUAL SCALING DESIRED ?
          MOV #21,TEMP
          JSR PC,WRTY
SCALE3:   JSR PC,RDTY
          CMP #15,RO
          BEQ REDUCE
          MOV #0,SCALNT
          BIC #10,RO
          BIS RO,COUNTER
          BR SCALE3
REDUCE:   MOV #LITTLE,RO         ; PICTURE REDUCED ?
          MOV #13,TEMP
```

```
        JSR  PC,WRTY
        JSR  PC,RDTY
        CMP  #'Y,R0             ; OPTION DESIRED
        BNE  REDCON
        MOV  #240,SHKPC1+2
        MOV  #240,SHKPC1
        MOV  #240,SHKPC2        ; SET FOR REDUCTION
        MOV  #240,SHKPC2+2
REDCON: MOV  #MOVEX, R0         ; HORIZONTAL SHIFT DESIRED ?
        MOV  #13,TEMP
        JSR  PC,WRTY
REDCNN: JSR  PC,RDTY
        CMP  #15,R0
        BEQ  REDCN2
        CMP  #'R,R0             ; MOVED RIGHT
        BNE  JMPCN1
        MOV  #62767,MOVRT
        MOV  #177700,MOVRT+2
        MOV  #167204,MOVRT+4
        BR REDCNN
JMPCN1: MOV  #100,MOVRT+2       ; MOVED LEFT
        MOV  #167204,MOVRT+4
        BR REDCNN
REDCN2: MOV  #MOVEY,R0          ; VERTICAL SHIFT DESIRED ?
        MOV  #13,TEMP
        JSR  PC,WRTY
REDCN3: JSR  PC,RDTY
        CMP  #15,R0
        BEQ  ROTQST
        CMP  #'D,R0             ; MOVED DOWN
        BNE  JMPCN2
        MOV  #62767,MOVUP
        MOV  #177700,MOVUP+2
        MOV  #167200,MOVUP+4
        BR REDCN3
JMPCN2: MOV  #100,MOVUP+2       ; MOVED UP
        BR REDCN3
ROTQST: MOV  #ROTPIC,R0         ; WHICH ALGORITHM IS TO
        MOV  #15,TEMP           ; BE USED FOR ROTATION ?
        JSR  PC,WRTY
ROTRED: JSR  PC,RDTY
        CMP  #15,R0
        BEQ  FILE
        CMP  #'S,R0             ; SHIFTING ALGORITHM
        BEQ  ROTRED
        MOV  #137,NOROTE        ; ROTATING ALGORITHM
        MOV  #11270,NOROTE+2
        MOV  #0,QONE+2
        MOV  #0,QTWO+2
        MOV  #0,QTHREE+2
```

```
        MOV  #0,QFOUR+2         ; SET FOR ROTATING ALGORITHM
        MOV  #0,AMOV+2
        MOV  #0,BMOV+2
        MOV  #0,CMOV+2
        MOV  #0,DMOV+2
        MOV  #0,EMOV+2
        MOV  #0,FMOV+2
        MOV  #0,GMOV+2
        MOV  #0,HMOV+2
        CMP  LEVEL,#1
        BEQ  TYPE3
        CMP  LEVEL,#4           ; SET FOR QUADRANT
        BEQ  TYPE2
TYPE1:  MOV  #240,SHKPC1
        MOV  #240,SHKPC1+2
        JMP  ROTRED
TYPE2:  CMP  QUADRT,#4
        BNE  TYPCON
        CMP  RANGLE,#45
        BLT  TYPE1
        BR   TYPCN1
TYPCON: CMP  RANGLE,#45
        BGE  TYPE1
TYPCN1: MOV  #240,TYPFOR
        MOV  #240,TYPFOR+2
        MOV  #240,TYPFR1
        MOV  #240,TYPFR1+2
        MOV  #240,SHKPC1
        MOV  #240,SHKPC1+2
TYPE3:  JMP  ROTRED
FILE:   MOV  #FILL,R0           ; FILE NUMBER DESIRED ?
        MOV  #11,TEMP
        JSR  PC,WRTY
        MOV  #-1,TEMP
FILLIN: JSR  PC,RDTY            ; INPUT FILE NUMBER
        CMP  #15,R0
        BEQ  QUEST
        TST  TEMP
        BPL  .+6
        CLR  FILNUM
        MOV  #1,TEMP
        BIT  #10,R0
        BNE  STRPFL
        BIC  #-10,R0
        ASL  FILNUM
        ASL  FILNUM
        ASL  FILNUM
        BIS  R0,FILNUM
        BR   FILLIN
STRPFL: BIC  #-20,R0
```

```
        ASL FILNUM
        ASL FILNUM
        ASL FILNUM
        BIS R0,FILNUM
        BR FILLIN
QUEST:  MOV #PICT,R0        ; ASK FOR PICTURE NAME
        MOV #15,TEMP
        JSP PC,WRTY
        MOV #0,HOLD
        MOV #NAME,R0
NAMBLK: MOVB #' ,(R0)+      ; INPUT PICTURE NAME
        INC HOLD            ; CHARACTER
        CMP HOLD,#20
        BNE NAMBLK
        MOV #NAME,R1
        MOV #COLCD,R2
        MOV #TWODCD,R3
        JSR PC,RDTY
        CMP #15,R0
        BEQ SEARCH
        MOVB R0,(R1)+
        CMP #'C,R0          ; COLOR SPECIFICATION
        BEQ COLOR
        CMP #'T,R0
        BEQ TWODD           ; TWO-D SPECIFICATION
PICNAM: JSR PC,RDTY
        CMP #15,R0
        BEQ SEARCH          ; START TAPE SEARCH
        MOVB R0,(R1)+
        BR PICNAM
COLOR:  JSR PC,RDTY         ; INPUT COLOR SPECIFICATIONS
        CMP #15,R0
        BEQ QCOLOR
        MOVB R0,(R1)+
        CMP R0,(R2)+
        BNE PICNAM
        BR COLOR
TWODD:  JSR PC,RDTY         ; INPUT TWO-D SPECIFICATIONS
        CMP #15,R0
        BEQ SETFTD
        MOVB R0,(R1)+
        CMP R0,(R3)+
        BNE PICNAM
        BR TWODD
SETFTD: JMP PICTWD          ; GO TO TWO-D SET-UP ROUTINE
QCOLOR: MOV #NAXES,R0
        MOV #11,TEMP        ; AXES COLOR ?
        MOV #CAXES,R4
        JSR PC,WRTY
        BR QNUM
```

```
QFUNCT:   MOV  #NFUNCT,R0
          MOV  #14,TEMP
          MOV  #CFUNCT,R4       ; FUNCTION COLOR ?
          JSR  PC,WRTY
QNUM:     CLR  R5
QBRTO:    ASL  R5
          ASL  R5
          ASL  R5
          JSR  PC,RDTY          ; INPUT COLOR DIGIT
          CMP  #15,R0
          BEQ  CHECK
          BIC  #-10,R0
          BIS  R0,R5
          BR QBRTO
QCHECK:   CMP  #2,NSWITCH
          BEQ  FUNCPR
          MOV  #2,NSWITCH
          MOV  R5,(R4)          ; AXES COLOR DIGIT
          BR QFUNCT
FUNCPR:   MOV  R5,(R4)          ; FUNCTION COLOR DIGIT
          BR QUEST
SEARCH:  BIT #1,MTS
         BEQ .-6
         MOV #-10000,MTBRC
          MOV  #BUFFR1,MTCMA    ; SEARCH FOR CORRECT
          MOV  #60003,MTC       ; PICTURE ON TAPE
          BIT  #100200,MTC
          BEQ  .-6
          BMI  MTEROR           ; MAG TAPE ERROR ?
          BIT  #1,MTS
          BEQ  .-6
          CMPB BUFFR1+2,#'P     ; PROGRAM RECORD
          BEQ  CNTSKP
          MOV  BUFFR1,HOLD
          CMP  HOLD,FILNUM      ; FILE NUMBER CORRECT ?
          BEQ  PROGRAM          ; BEGIN PROCESSING
          MOV  #4,R3
          MOV  #BUFFR1+10,R4
          MOV  #NAME,R5
CHECK:    CMPB (R4)+,(R5)+      ; NAME AS REQUESTED ?
          BNE  CNTSKP           ; NO - CONTINUE SEARCH
          DEC  R3
          BNE  CHECK
          BR PROGRAM            ; YES - BEGIN PROCESSING
MTEROR:   JMP  ERROR
CNTSKP:   MOV  #-200,MTBRC
          MOV  #60011,MTC       ; CONTINUE TO SKIP
          MOV  #BUFFR1,BUFFIN   ; FOR NEXT FILE
          BIT  #100200,MTC
          BEQ  .-6
```

```
          BMI  SEARCH            ; CHECK FOR CORRECT FILE
          JMP  ERROR             ; MAG TAPE ERROR
PROGRAM:  CMPB BUFFR1+3,#'S
          BEQ  SML               ; SMALL PICTURE (128)
          CMPB BUFFR1+3,#'M
          BEQ  MED               ; MEDIUM PICTURE (256)
          MOV  #200,PICSIZ
          BR   S128
SML:      MOV  #1000,PICSIZ      ; 512 PICTURE
          BR   S512              ; LARGE PICTURE (512)
MED:      MOV  #400,PICSIZ       ; 256 PICTURE
          BR   S256
S128:     MOV  #1,NUMSIZ
          MOV  #4,SKPCNT         ; SET COUNTER FOR 128
          MOV  #3,NUMSKP
          BR   START
S256:     MOV  #2,NUMSIZ
          MOV  #10,SKPCNT        ; SET COUNTER FOR 256
          MOV  #6,NUMSKP
          BR   START
S512:     MOV  #4,NUMSIZ
          MOV  #20,SKPCNT        ; SET COUNTER FOR 512
          MOV  #14,NUMSKP
START:    TST  SCALNT            ; SCALING FLAG SET ?
          BMI  .+6
          JMP  STAR2
          CLR  COUNTER           ; SET FOR SCALING
          CLR  R4
NAME1:    MOV  #-10000,MTBRC
          MOV  #BUFFR1,MTCMA
          MOV  #60003,MTC        ; READ DATA RECORD
          BIT  #100200,MTC
          BEQ  .-6
          MOV  #10000,R2
          MOV  #BUFFR1,R3
NAME2:    MOVB (R3)+,R5          ; FIND LARGEST DATA POINT
          ADD  #200,R5
          TST  R5
          BPL  .+4
          BR   NAME2
          CMP  R5,R4
          BPL  SWITH2
NAME3:    DEC  R2
          BEQ  NAME1
          BR   NAME1
ENDREAD:  MOV  R4,TEMP
          TST  TEMP              ; DATA ALL NEGATIVE ?
          BMI  SKPBCK
TEST:     CMP  TEMP,#400
          BMI  .+4
```

```
        BR SKPBCK
        ASL COUNTER             ; SHIFT FOR SCALING
        ASR TEMP
        BR TEST
SKPBCK: CLR COUNTER
SAVSCA: MOV #-1,SCALNT
        MOVB COUNTER,SCANUM     ; SAVE SCALE FACTOR
        BIS #600,SCANUM
        MOV #SCAMES,R0
        MOV #31,TEMP
        JSR PC,WRTY
        MOV PICSIZ,TEMP         ; SET FOR PICTURE SIZE
        CMP TEMP,#1000
        BMI MEDSIZ
        MOV #-101,MTBRC
        BR BACSPA
MEDSIZ: CMP TEMP,#400
        BMI SMLSIZ
        MOV #-21,MTBRC
        BR BACSPA
SMLSIZ: CMP TEMP,#200
        BMI SPESIZ
        MOV #-5,MTBRC
        BR BACSPA
SPESIZ: MOV #-2,MTBRC
        BEQ .-6
        BIT #1,MTS
        BEQ .-6
        BIT #100200,MTC
BACSPA: MOV #60013,MTC          ; RETURN TO FIRST DATA RECORD
STA:2:   MOV #MAXVAL,R0
         MOV #-2000,(R0)+       ; CLEAR HIDDEN
        CMP R0,#14000           ; POINT ARRAYS
         BNE .-10
CLRCR:   MOV #20,DRSCAN
         CLR DRSCAN
         MOV #100,CLKCSB        ; CLEAR DISPLAY SCREEN
         MOV #5,CLKCSR
         TSTB CLKCSR
         BPL .-4
         MOV #200,DRSCAN
         CLR CLKCSR
DECID2:  MOV #1,FORREV
        CMP #1,QUADRT
        BEQ DECID3
        CMP #2,QUADRT
        BEQ DECID3
        MOV #-1,AXLAST
DECID3   TST TWDCNT             ; TEST FOR TWO-D
         BPL .+6
```

```
          JMP FIRST
          CMP #1,QUADRT          ; FIRST QUADRANT ?
          BEQ QONE
          CMP #2,QUADRT          ; SECOND QUADRANT ?
          BEQ QTWO
          CMP #3,QUADRT          ; THIRD QUADRANT ?
          BEQ QTHREE
          CMP #4,QUADRT          ; FOURTH QUADRANT ?
          BNE .+6
          JMP QFOUR
          JMP ERROR              ; QUADRANT NUMBER INCORRECT
QONE:     MOV #150,XFROM
AMOV:     MOV #-1400,YFROM
          CMP RANGLE,#54
          BLE .+6                ; ANGLE LESS THAN 45 DEGREES
          JMP QTONE
          MOV LEVEL,YMULT        ; SET YMULT
          BIS #100000,ADDYRT
          CMP RANGLE,#41
          BLT .+14               ; ANGLE LESS THAN 33 DEGREES
          MOV #0,XMULT           ; SET XMULT TO 0
          JMP FIRST
          CMP RANGLE,#26
          BLT .+14               ; ANGLE LESS THAN 22 DEGREES
          MOV #1,XMULT           ; SET XMULT TO 1
          JMP FIRST
          CMP RANGLE,#13         ; ANGLE LESS THAN 11 DEGREES
          BLT .+14
          MOV #2,XMULT           ; SET XMULT TO 2
          JMP FIRST
          CMP RANGLE,#0          ; ANGLE EQUAL 0 DEGREES
          BLE .+14
          MOV #3,XMULT           ; SET XMULT TO 3
          JMP FIRST
          MOV #4,XMULT           ; SET XMULT TO 4
          JMP FIRST
QTWO:     MOV #0,XFROM
BMOV:     MOV #-1000,YFROM
          CMP RANGLE,#54
          BLE .+6
          JMP QTTWO
          MOV LEVEL,YMULT
          CMP RANGLE,#41
          BLT .+14
          MOV #0,XMULT
          JMP FIRST
          CMP RANGLE,#26
          BLT .+14
          MOV #-1,XMULT
          JMP FIRST
```

```
            CMP  RANGLE,#13
            BLT  .+14
            MOV  #-2,XMULT
            JMP  FIRST
            CMP  RANGLE,#0
            BLE  .+14
            MOV  #-3,XMULT
            JMP  FIRST
            MOV  #-4,XMULT
            JMP  FIRST
QTHREE:     MOV  #-1400,XFROM
CMOV:       MOV  #140,YFROM
            CMP  RANGLE,#54
            BGE  .+6
            JMP  QTTHRE
            MOV  LEVEL,YMULT
            NEG  YMULT
            BIS  #100000,ADDXRT
            BIS  #100000,ADDYRT
            CMP  RANGLE,#70
            BGE  .+14
            MOV  #0,XMULT
            JMP  FIRST
            CMP  RANGLE,#103
            BGE  .+14
            MOV  #1,XMULT
            JMP  FIRST
            CMP  RANGLE,#117
            BGE  .+14
            MOV  #2,XMULT
            JMP  FIRST
            CMP  RANGLE,#132
            BGE  .+14
            MOV  #3,XMULT
            JMP  FIRST
            MOV  #4,XMULT
            JMP  FIRST
QFOUR:      MOV  #1100,XFROM
DMOV:       MOV  #100,YFROM
            CMP  RANGLE,#54
            BGE  .+6
            JMP  QTFOUR
            MOV  LEVEL,YMULT
            NEG  YMULT
            BIS  #100000,ADDXRT
            CMP  RANGLE,#70
            BGE  .+14
            MOV  #0,XMULT
            JMP  FIRST
            CMP  RANGLE,#103
```

```
          BGE .+14
          MOV #-1,XMULT
          JMP FIRST
          CMP RANGLE,#117
          BGE .+14
          MOV #-2,XMULT
          JMP FIRST
          CMP RANGLE,#132
          BGE .+14
          MOV #-3,XMULT
          JMP FIRST
          MOV #-4,XMULT
          JMP FIRST
SECOND:   JMP FRTTWO          ; GO TO SECOND ROUTINE
QTONE:    MOV LEVEL,YMULT
EMOV:     MOV #1200,XFROM
          CMP RANGLE,#70
          BGE .+14
          MOV #0,XMULT
          JMP SECOND
          CMP RANGLE,#103
          BGE .+14
          MOV #-1,XMULT
          JMP SECOND
          CMP RANGLE,#117
          BGE .+14
          MOV #-2,XMULT
          JMP SECOND
          CMP RANGLE,#132
          BGE .+14
          MOV #-3,XMULT
          JMP SECOND
          MOV #-4,XMULT
          JMP SECOND
QTTWO:    MOV LEVEL,YMULT
FMOV:     MOV #-1400,XFROM
          BIS #100000,ADDYRT
          CMP RANGLE,#70
          BGE .+14
          MOV #0,XMULT
          JMP SECOND
          CMP RANGLE,#103
          BGE .+14
          MOV #1,XMULT
          JMP SECOND
          CMP RANGLE,#117
          BGE .+14
          MOV #2,XMULT
          JMP SECOND
          CMP RANGLE,#132
```

```
            BGE  .+14
            MOV  #3,XMULT
            JMP  SECOND
            MOV  #4,XMULT
            JMP  SECOND
QTTHRE:     MOV  LEVEL,YMULT
            NEG  YMULT
GMOV:       MOV  #-100,XFROM
            BIS  #100000,ADDXRT
            CMP  RANGLE,#41
            BLT  .+14
            MOV  #0,XMULT
            JMP  SECOND
            CMP  RANGLE,#26
            BLT  .+14
            MOV  #-1,XMULT
            JMP  SECOND
            CMP  RANGLE,#13
            BLT  .+14
            MOV  #-2,XMULT
            JMP  SECOND
            CMP  RANGLE,#0
            BLE  .+14
            MOV  #-3,XMULT
            JMP  SECOND
            MOV  #-4,XMULT
            JMP  SECOND
QTFOUR:     MOV  LEVEL,YMULT
            NEG  YMULT
HMOV:       MOV  #0,XFROM
            BIS  #100000,ADDXRT
            BIS  #100000,ADDYRT
            CMP  RANGLE,#41
            BLT  .+14
            MOV  #0,XMULT
            JMP  SECOND
            CMP  RANGLE,#26
            BLT  .+14
            MOV  #1,XMULT
            JMP  SECOND
            CMP  RANGLE,#13
            BLT  .+14
            MOV  #2,XMULT
            JMP  SECOND
            CMP  RANGLE,#0
            BLE  .+14
            MOV  #3,XMULT
            JMP  SECOND
            MOV  #4,XMULT
            JMP  SECOND
```

```
ERRJMP:   JMP ERROR              ; AN ERROR HAS OCCURRED
FIRST:    BIT #1,MTS             ; FIRST INPUT ROUTINE
          BEQ .-6 ;              ; TAPE UNIT READY
          MOV #BUFFR1,BUFFIN
          MOV #-10000,MTBRC      ; SET BUFFER AND BYTE COUNTER
          MOV #BUFFR1,MTCMA
          MOV #60003,MTC         ; READ A RECORD
          TST AXLAST
          BPL .+6
          JMP INITAL
AXES:     MOV #3,R5
          MOV #200,R4            ; INITIALIZE TO THE ORIGIN
          CLR ROWNUM             ; TO WRITE THE FUNCTIONAL
          CLR PNTNUM             ; AXES
          CLR POINT
          MOV #1,KOUNT
JUMPTO:   JMP ROTATE             ; TAKE THE POINT TO BE ROTATED
JUMPBK:   DEC R5
          BEQ SLOPE              ; Y-AXIS ?
          DEC R5
          BEQ VERT               ; Z-AXIS ?
          INC ROWNUM             ; X-AXIS
          MOV #3,R5
          DEC R4
          BNE JUMPTO
          MOV #200,R4            ; INITIALIZE FOR Z-AXIS
          MOV #0,ROWNUM
VERT:     INC PNTNUM             ; Z-AXIS
          MOV #2,R5
          DEC R4
          BNE JUMPTO
          MOV #200,R4            ; INITIALIZE FOR Y-AXIS
          CLR PNTNUM
SLOPE:    INC POINT              ; Y-AXIS
          MOV #1,R5
          DEC R4
          BNE JUMPTO
          TST AXLAST
          BPL .+6
          JMP CTPICT
          CMP #-1,FORREV         ; FIRST/SECOND INPUT ROUTINE ?
          BNE INITAL
          JMP INITWO
INITAL:   MOV #1,ROWNUM          ; FIRST ROUTINE CONTINUED
          MOV PICSIZ,PNTNUM
          CLR POINT              ; INITIALIZE FOR FUNCTION
          MOV #2,KOUNT
          TST SPCONT
          BMI PREVBF-6
RECORD:   CMP #BUFFR1,BUFFIN
```

```
        BEQ BUFTWO          ; WHICH BUFFER IS FULL ?
        MOV #BUFFR1,BUFFIN
        MOV #BUFFR2,BUFWRK
        BR READER
BUFTWO: MOV #BUFFR2,BUFFIN  ; SET TO USE EMPTY BUFFER
        MOV #BUFFR1,BUFWRK
READER: BIT #100200,MTC
        BEQ .-6 ;           ; TAPE UNIT READY ?
        BMI ENDPIC
        BIT #1,MTS
        BEQ .-6
        MOV #-10000,MTBRC   ; SET BUFFER AND BYTE COUNTER
        MOV BUFFIN,MTCMA
        MOV #60003,MTC      ; READ NEXT RECORD
        MOV #1,PNTCNT
PREVBF: MOV BUFWRK,R3       ; SET TO PROCESS RECORD
        MOVB (R3)+,P2       ; READ NEXT POINT
        TST SCALNT
        BGT ADSHFT
        MOV COUNTER,HOLD
        TST HOLD
        BEQ ADSFHT
SHFTR2: ASR R2
        ASR HOLD
        BEQ ADSHFT
        BR SHFTR2
ADSHFT: ADD SHIFT,R2        ; SHIFT AS DESIRED
        MOV R2,POINT
        JMP ROTATE          ; TAKE THE POINT TO BE ROTATED
JMPBUF: DEC PNTNUM
        ADD NUMSIZ,PNTCNT   ; SET UP FOR NEXT POINT
        CMP #0,PNTNUM
        BGE ROWCNG          ; END OF ROW ?
        CMP SKPCNT,CNTNUM
        BEQ PREVBF+4        ; GET NEXT POINT
        SUB NUMSKP,PNTNUM   ; SKIP OMITTED POINTS
        ADD NUMSKP,PNTCNT
        ADD NUMSKP,R3
        CMP #0,PNTNUM
        BGE ROWCHG          ; END OF ROW ?
        BR PREVBF+4         ; GET NEXT POINT
ROWCHG: INC ROWNUM          ; CHANGE TO NEXT ROW
        CMP ROWNUM,PICSIZ
        BGE ENDPIC          ; END OF PICTURE ?
        CMP #10000,PNTCNT
        BLE ENDBUF          ; END OF RECORD BUFFER ?
        MOV PICSIZ,PNTNUM   ; MOVE TO BEGINNING
        DEC CNTNUM          ; OF NEW ROW
        CMP #1,CNTNUM
        BEQ CHGNUM
```

```
          BR PREVBF+4           ; GET NEXT POINT
ENDBUF:   JMP RECORD            ; GET NEXT RECORD
CHGNUM:   MOV #4,CNTNUM
          BR PREVBF+4           ; GET NEXT POINT
ENDPIC:  TST AXLAST
         BPL .+6
         JMP AXES
          MOV #2,DRSCAN
QTPICT:   MOV #0,#0             ; WRITE CHIP TO SCREEN
          CLR DRSCAN
          JMP START             ; RETURN TO START OF DIALOG
ERROR:   MOV #10101,#177700     ; SET ERROR MESSAGE
          HALT
ERRCHK:  BIT #40000,MTS
         BNE ENDPIC
         BR ERROR
ERTTWO:  TST SPCONT             ; SECOND INPUT ROUTINE
         BMI READIN
         BIT #1,MTS
          BEQ .-6
          MOV #-1,FORREV        ; SET FOR SECOND ROUTINE
          MOV #BUFFR1,BUFFIN
          MOV #-1000,MTBRC      ;SET TO SKIP TO END OF FILE
          MOV #60011,MTC
          BIT #100200,MTC       ; TAPE UNIT THROUGH
          BEQ .-6
          BIT #40000,MTC        ; TAPE MARK READ
          BEQ ERROR
          MOV #-2,MTBRC         ; BACKSPACE ONE RECORD
          MOV #60013,MTC
READIN:  BIT #100200,MTC
         BEQ .-6
         BMI ERROR
          BIT #1,MTS            ; TAPE UNIT READY ?
          BEQ .-6
         MOV #-1,FORREV
          MOV #-10000,MTBRC
          MOV BUFFIN,MTCMA      ; SET BUFFER AND BYTE COUNTER
          MOV #60003,MTC
         TST AXLAST
         BMI .+6
          JMP AXES              ; GO TO AXES ROUTINE
INITWC:   MOV PICSIZ,ROWNUM     ; SECOND ROUTINE CONTINUED
          MOV #1,PNTNUM
          CLR POINT             ; INITIALIZE FOR FUNCTION
          MOV #2,KOUNT
RECRD2:  TST SPCONT
         BMI PREVTO-6
         CMP #BUFFR1,BUFFIN
          BEQ BUFTW2            ; WHICH BUFFER IS FULL ?
```

```
            MOV #BUFFR1,BUFFIN
            MOV #BUFFR2,BUFWRK
            BR READTW
BUFTW2:     MOV #BUFFR2,BUFFIN  ; SET TO EMPTY BUFFER
            MOV #BUFFR1,BUFWRK
READTW:     BIT #1,MTS
            BEQ .-6             ; TAPE UNIT READY ?
            MOV #-2,MTBRC
            MOV #60013,MTC      ; BACKSPACE TWO RECORDS
            BIT #1,MTS
            BEQ .-6
            TST MTC             ; CHECK FOR ERROR
            BPL ERROR
            BIT #40000,MTC      ; TAPE MARK ENCOUNTERED
            BEQ ENDTWO
            MOV #-10000,MTBRC   ; SET BUFFER AND BYTE COUNTER
            MOV BUFFIN,MTCMA
            MOV #60003,MTC      ; READ NEXT RECORD
            MOV #1,PNTCNT       ; SET TO PROCESS RECORD
PREVTC:     MOV BUFWRK,R3
           TST SPCONT
           BPL .+10
           ADD PICCNT,R3
           BR .+6
            ADD #10000,R3
RETURN:     MOVB (R3)-,R2       ; READ NEXT POINT
           TST SCALNT
           BGT ADDSFT
           MOV COUNTER,HOLD
           TST HOLD
           BEQ ADDSFT
SHIFT2:    ASR R2
           ASR HOLD
           BEQ ADDSFT
           BR SHIFT2
ADDSFT:     ADD SHIFT,R2        ; SHIFT AS DESIRED
           MOV R2,POINT
            JMP ROTATE          ; TAKE POINT TO BE ROTATED
JMPBF2:     INC PNTNUM
            ADD NUMSIZ,PNTCNT   ; SET UP FOR NEXT POINT
            CMP PICSIZ,PNTNUM
            BLE ROWCH2          ; END OF ROW ?
            CMP SKPCNT,CNTNUM
            BEQ RETURN          ; GET NEXT POINT
            ADD NUMSKP,PNTNUM   ; SKIP OMITTED POINTS
            ADD NUMSKP,PNTCNT
            ADD NUMSKP,R3
            CMP PICSIZ,PNTNUM
            BLE ROWCH2          ; END OF ROW ?
            BR RETURN           ; GET NEXT POINT
```

```
ROWCH2:   DEC  ROWNUM            ; CHANGE TO NEXT ROW
          CMP  ROWNUM,#0
          BLE  ENDTWO            ; END OF PICTURE ?
          CMP  #10000,PNTCNT
          BLE  ENDBF2            ; END OF RECORD BUFFER ?
          MOV  #1,PNTNUM         ; MOVE TO BEGINNING
          DEC  CNTNUM            ; OF NEW ROW
          CMP  #1,CNTNUM
          BEQ  CHGNM2
          BR  RETURN             ; GET NEXT POINT
ENDBF2:   JMP  RECRD2            ; GET NEXT RECORD
CHGNM2:   MOV  #4,CNTNUM
          BR  RETURN             ; GET NEXT POINT
ENDTWO:   JMP  ENDPIC            ; GO TO ENDING ROUTINE
ROTATE:   CMP  #1,QUADRT
          BEQ  NEGROW            ; QUADRANT ONE OR FOUR ?
          CMP  #4,QUADRT
          BNE  ROTOTE
NEGROW:   NEG  ROWNUM            ; NEGATE THE X-VECTOR
ROTOTE:   MOV  ROWNUM,XROT       ; LOAD THE X-ROTATION BUFFER
          TST  ROWNUM
         BPL  .+6
          NEG  ROWNUM            ; RE-NEGATE THE X-VECTOR
          MOV  PNTNUM,YROT       ; LOAD THE Y-ROTATION BUFFER
          MOV  POINT,ZROT        ; LOAD THE Z-ROTATION BUFFER
         CMP  #1,KOUNT
         BNE  .+6
         JMP  ROTOTE
         CMP  #200,PICSIZ
         BLE  SHRINK
         ASL  XROT               ; EXPAND THE SMALL PICTURES
         ASL  YROT
         CMP  #100,PICSIZ
         BNE  .+6
         JMP  NOROTE
         ASL  XROT
         ASL  YROT
         CMP  #40,PICSIZ
         BEQ  NOROTE
         ASL  XROT
         ASL  YROT
         CMP  #20,PICSIZ
         ASL  XROT
         ASL  YROT
         BR  NOROTE
SHRINK:  CMP  #200,PICSIZ
         BGE  NOROTE
          ASR  XROT              ; SHRINK THE 512 PICTURE
         ASR  YROT
         CMP  #400,PICSIZ
```

```
          BGE NOROTE
           ASR YROT              ; SHRINK THE 256 PICTURE
          ASR YROT
NOROTE:    MOV YROT,MQ           ; FIGURE THE X-ROTATION
           MOV XMULT,MUL
           ASL XROT
           ASL XROT
           ASL ZROT
           ASL ZROT
           ADD MQ,XROT
ROTCON:    MOV YROT,MQ           ; FIGURE THE Y-ROTATION
           MOV YMULT,MUL
           MOV MQ,YROT
           ADD ZROT,YROT
ROTTWO:    MOV XROT,XVAL
           MOV YROT,YVAL
TYPFOR:    ASL XVAL              ; EXPAND THE PICTURE
TYPFR1:    ASL YVAL
           CMP #1,KOUNT
          BNE HIDDEN
          TST AXLAST
          BPL WRTPNT             ; GO TO THE WRITE ROUTINE
HIDDEN:    MOV XVAL,XPVAL
           ADD #2000,XPVAL       ; LOCATE THE HIDDEN-POINT
           ASR XPVAL             ; VECTOR
           BIC #1,XPVAL
           ADD #MAXVAL,XPVAL
           MOV XPVAL,R1
           MOV (R1),TSTVAL
           CMP YVAL,TSTVAL       ; CHECK FOR HIDDEN OR
           BLT REPLACE           ; VISIBLE POINT
           JMP DECIDE
REPLACE:   MOV YVAL,(R1)         ; RELOAD HIDDEN-POINT ARRAY
WRTPNT:    ADD XFROM,XVAL        ; RECENTER THE PICTURE
           ADD YFROM,YVAL
           CMP #1,KOUNT
           BNE .+12
           MOV CAXES,DABI        ; WRITE AXES LEVEL
           BR .+10
           MOV CFUNCT,DABI       ; WRITE FUNCTION LEVEL
           MOV XVAL,DABX
           MOV YVAL,DABY         ; LOAD OUTPUT REGISTERS
           MOV #6,DRSCAN
           MOV #0,#0             ; WRITE THE POINT ON THE CHIP
           MOV #0,#0
           INC DRSCAN
           DEC DRSCAN
DECIDE:    CMP #1,KOUNT          ; DECIDE FIRST/SECOND
           BEQ JMPTWO            ; ROUTINE PROCESSING
           TST FORREV            ; CURRENT PICTURE
```

```
            BPL JMPONE
            JMP JMPBF2          ; RETURN TO SECOND ROUTINE
JMPONE:     JMP JMPBUF          ; RETURN TO FIRST ROUTINE
JMPTWO:     JMP JUMPBK          ; RETURN TO AXES ROUTINE
WRTY:       TSTB TPS1
            BPL .-4 ;           ; WRITE TO TTY SUBROUTINE
            MOVB (R0)+,TPB1
            DEC TEMP
            BNE WRTY
            RTS PC
RDTY:       TSTB TKS1           ; READ FROM TTY SUBROUTINE
            BPL .-4
            MOVB TKB1,R0        ; INPUT CHARACTER
            TSTB TPS1
            BPL .-4
            MOV R0,TPB1         ; ECHO CHARACTER
            BIC #-200,R0        ; STRIP PARITY BITS
            RTS PC
PICTWD:     MOV #0,XMULT        ; SET FOR TWO-DIMENSIONAL
            MOV #0,YMULT        ; OUTPUT ROUTINE
            MOV #1,QUADRT
            MOV #-1,TWDCNT
            JMP QUEST
SPECAL:     MOV #1,SPCONT       ; SET SMALL PICTURE FLAG
            CMPB BUFFR1+3,#6
            BEQ SIXTY
            CMPB BUFFR1+3,#3
            BEQ THIRTY          ; DETERMINE PICTURE SIZE
            CMPB BUFFR1+3,#1
            BEQ SIXTEN
            CMPB BUFFR1+3,#4
            BEQ EIGHT
            JMP ERROR
SIXTY:      MOV #100,PICSIZ     ; 64 X 64 PICTURE
            BR FIGPIC
THIRTY:     MOV #40,PICSIZ      ; 32 X 32 PICTURE
            BR FIGPIC
SIXTEN:     MOV #20,PICSIZ      ; 16 X 16 PICTURE
            BR FIGPIC
EIGHT:      MOV #10,PICSIZ      ; 8 X 8 PICTURE
FIGPIC MOV PICSIZ,MQ
            MOV PICSIZ,MUL
            MOV MQ,PICCNT
            MOV #1,NUMSIZ       ; SET FOR SMALL PICTURES
            MOV #4,SKPCNT
            CLR NUMSKP
            JMP START
            MOV YROT,MQ
            MOV MQ,TEMP
            MOV YMULT,MUL
```

```
NOSHFT:  MOV  XROT,MQ          ; ROTATING ALGORITHM
         MOV  YMULT,MUL        ; FIGURE THE X-ROTATION
SHKPC1:  ASL  XROT
         ASL  XROT
         ASL  ZROT
         ASL  ZROT
SHKPC2:  ASL  YROT
         ADD  MQ,XROT
ADDXRT:  ADD  TEMP,XROT
         MOV  YROT,MQ
         MOV  YMULT,MUL        ; FIGURE THE Y-ROTATION
         MOV  MQ,YROT
         ADD  ZROT,YROT
ADDYRT:  ADD  TEMP,YROT
         SUB  SHIFT,YROT
MOVRT:   ADD  #100,XROT
MOVUP:   ADD  #100,YROT
         JMP  ROTTWO
          .=14000              ; HIDDEN POINT ROUTINE ARRAYS
MAXVAL:   0                    ; MAXIMUM VALUE ARRAY
          .=16000
MINVAL:   0                    ; MINIMUM VALUE ARRAY
          .END
```

APPENDIX C

A TEST CASE

The following case study is included to allow the prospective user to more fully understand and appreciate the operation of the graphical display package. It is not intended to be an exhaustive display of all the possible combinations of options available with the program but more as an example of the manner in which the graphics package might be used to study a given set of pictures or functions. Included will be examples of the graphical output of the program and the man-machine dialog which produced that output.

The test picture for the first case is the function

$$Y(x) = 50 \times \sin(\sqrt{x^2 + y^2}) / \sqrt{x^2 + y^2}$$

Figure 10 is the function in quadrant one at an angle of 0 degrees, rotated by the rotating routine. The dialog which produced this output is below

```
FROM?
1,0
VIEWING LEVEL?
1
SCALING FUNCTION?

SCALING FACTOR?

FILE?
56404
FUNCTION?
```

Figure 11 is the same function at the same quadrant and angle with the viewing level raised to 3.

The function is again displayed in Figure 12, now rotated to an angle of 60 degrees in the fourth quadrant, at a viewing level of 2.

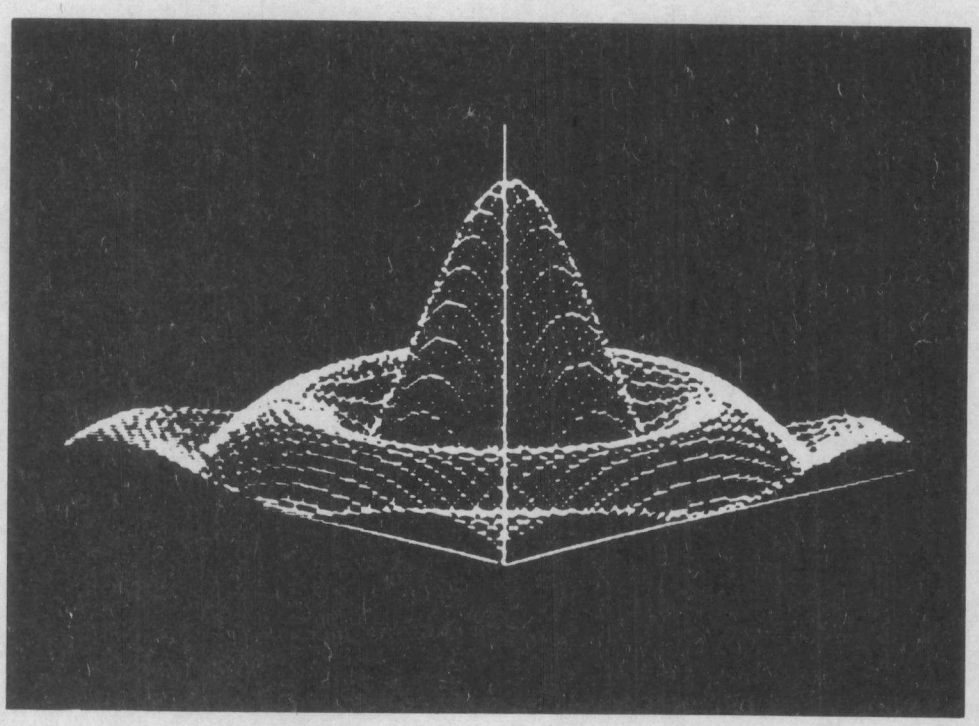

Figure 10. The function in quadrant one.

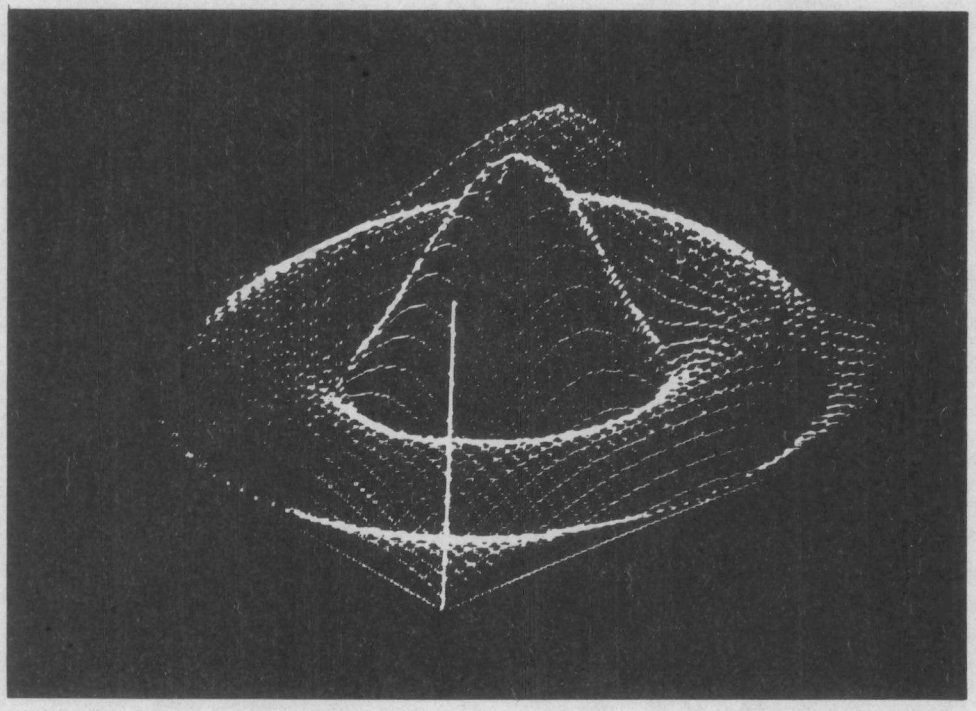

Figure 11. The function raised for viewing.

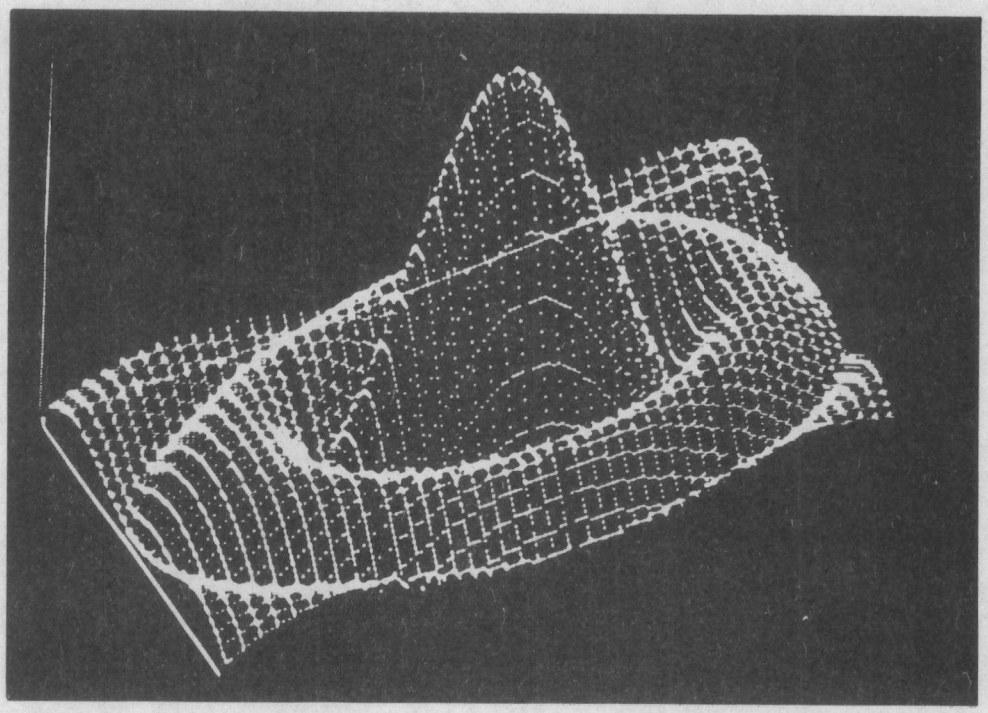

Figure 12. The function in quadrant four.

The function for case two is a photograph taken by the system camera. This photograph is shown in Figure 13. The photograph is displayed in Figure 14 in quadrant one at an angle of 10 degrees, using the shifting algorithm. It should be noted that the photograph of Figure 13 is a bright, clean picture, while the display in Figure 14 shows a substantial amount of noise content in the function. This enhancement of detail was one of the primary reasons for the development of the functional graphics package.

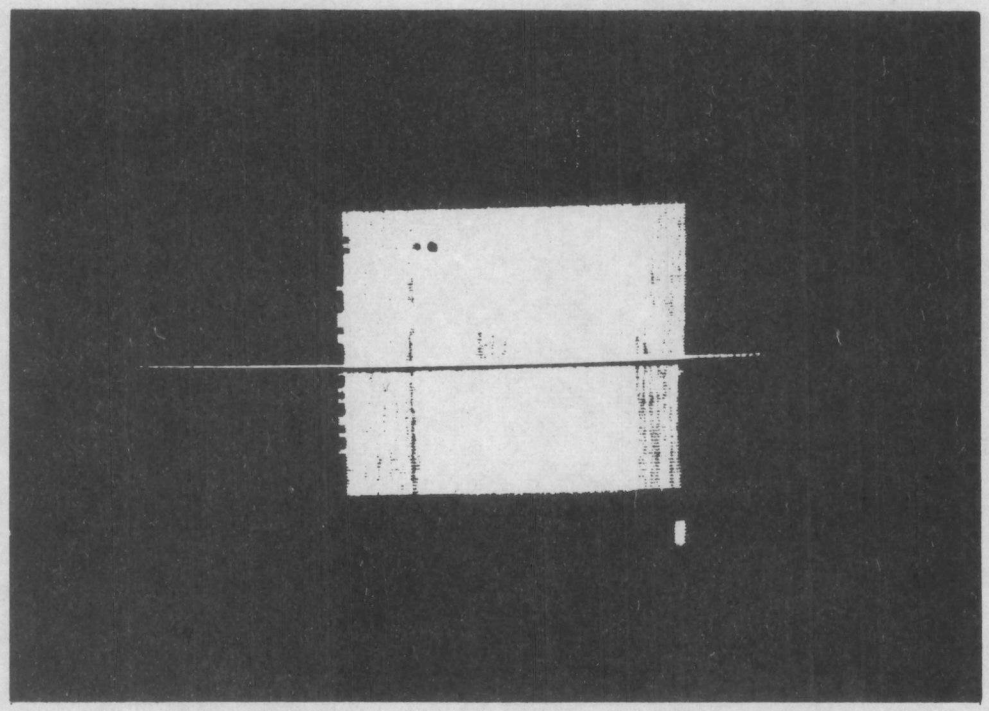

Figure 13. A system photograph.

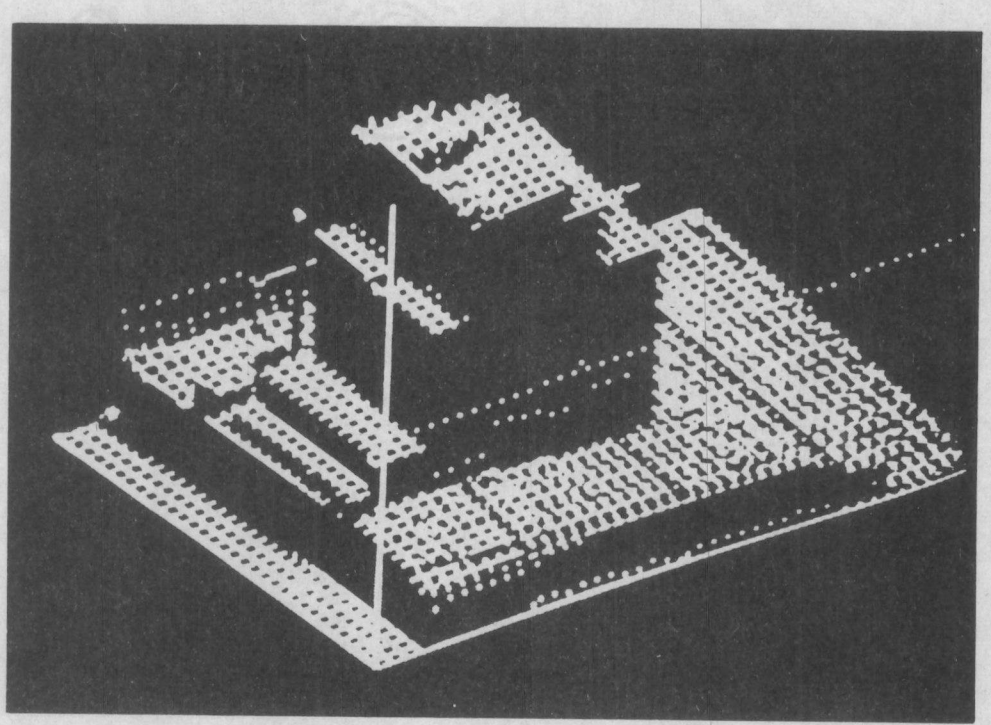

Figure 14. The photograph processed and displayed.

VITA

Larry Joe Minyard was born on January 5, 1948, in Memphis, Tennessee. In 1951 his father joined the U.S. Air Force and he attended schools in four states and two foreign countries. He was graduated from Forrest Sherman High School in Naples, Italy, in June of 1966. In the fall of 1967 he enrolled at The University of Tennessee in Knoxville. In June of 1971 he received the Bachelor of Science degree in Electrical Engineering and the following September enrolled in the Graduate School. He received the Master of Science degree in Electrical Engineering in August, 1973.

He is married to the former Alice Ann Dixon of Chattanooga, Tennessee.