Three-Dimensional Graphics Utilities

Part No. 09845-10061

Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Printing History

New revisions of this manual will incorporate all material updated since the previous revision.

The manual printing date and part number indicate its current revision. The printing date changes when a new revision is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

April 1980...Revision A

April 1981...Revision B. Updated pages: 2

August 1981...Revision C. Updated pages: 88

Table of Contents

Chapter 4: The Data Base

Chapter 5: Input Utilities

Preface

Computer graphics are being used more and more in areas of design simulation and control. As a result, there is increased interest in three dimensional graphics. For this reason, Hewlett-Packard has developed a pack of 3-D Utilities. These utilities can aid you in designing applications software that requires 3-D graphics. Using the pack's documented subprograms should reduce your software development time and help you to minimize programming problems.

For your convenience, each subprogram is explained in detail in this manual. In addition, variable lists, references, and formulas are also provided.

The pack furnishes subprograms to use for input, manipulation, viewing, and output of three dimensional figures. Also included, are subprograms that make use of advanced algorithms for polygon clipping and hidden-surface removal.

Four "demo" drivers are made available to illustrate the features of the 3-D utilities and to demonstrate the interaction that takes place between subprograms.

A versatile data base structure is used to maintain image data. Each 3-D object has: 1) status, 2) vertex, 3) edge, 4) surface, and 5) object information associated with it. You will find this hierarchical data base method to be a powerful yet flexible way of handling your graphical data.

Important

The tape cartridge or disc containing the programs is very reliable, but being a mechanical device, is subject to wear over a period of time. To avoid having to purchase a replacement medium, we recommend that you immediately duplicate the contents of the tape onto a permanent backup tape or disc. You should also keep back-up copies of your important programs and data on a separate medium to minimize the risk of permanent loss.

Chapter 1
Overview

Description

"Three-Dimensional Graphics Utilities" includes a set of subprograms to use in the input, manipulation, viewing, and output of 3-D figures. It also provides several general utilities such as a graphics input subprogram, a subprogram to dump the CRT graphics to an external printer, and a comment stripper program. This pack is not meant to be used as a stand alone but rather as a "kernel" for the development of higher-level applications programs.

The input (or figure construction) subprograms are divided into two categories, manual and digitizer. The manual routines are used to input objects by entering the coordinates of the vertices for each surface making up the object. These surfaces can be composed of line segments, circles, and arcs. The digitizer subprograms are used to input an object from a mechanical drawing of the object. This is done by digitizing all of it surfaces, visible and hidden, in each of the three orthographic views (top, front, and side). Refer to Chapters Two and Three of this manual for more detail. Circles and arcs are not provided as primitives when using digitizer entry. However, they can be approximated by using line segments. For example, you can approximate a circle by digitizing a 24-sided regular polygon.

The manipulation and viewing subprograms provide choosing a viewpoint, rotation, scaling, translation, and hither and yon clipping. Objects may then be plotted on the CRT or HP-IB plotters via the output routines as either wireframes or with hidden surfaces removed.

The information for all figures is stored in a hierarchical data base that is described in Chapter Four. The data base may be used as is, or tailored to a specific application.

It is important that you realize that many of the subprograms call one or more other subprograms. See the Reference Table provided in Chapter One. To help you get started, example driver programs have also been included in this manual. See Chapter Ten.

System Configuration

A 9845B Model 150 Desktop Computer is required for the "Three-Dimensional Graphics Utilities", P/N 09845-10060, even though the second tape drive and the thermal line printer are not mandatory. The standard 9845C Model 150 or 250 with 187K bytes of memory is required for "Three-Dimensional Graphics Utilities", P/N 09845-10080. A digitizer can be used as a means of entering data. Therefore, some of the subprograms included in this pack were designed for use with the 9874A. Disks are recommended (though not required) due to the large amount of storage space needed for data and programs. The following peripherals are compatible:

Digitizer

9874A Digitizer (requires I/O ROM)

Mass Storage

(Requires Mass Storage ROM)
9885M/S Floppy Disk Master and Slave
7906 Hard Disk Series

Plotters

9872A/S Plotter
7245 Printer/Plotter
7225A Mini-plotter

Printers (for use with DUMP GRAPHICS # binary)

9876 Stand Alone Flash
7245 Printer/Plotter
2631G Graphics Printer

NOTE

Unless otherwise noted, all references to the 9845B or 9845T in this manual are considered to mean the 9845B Model 150. References to the 9845C imply a 9845C Model 150 or 250.

Manual Usage Guide

The purpose of this section is to familiarize you with this manual so that you can use it more efficiently. Since this manual may be used with either the 09845-10060 or 09845-10080 pack, references are made to programs in both packs.

In Chapter One you will find an alphabetized quick reference of the utilities and the dependent subprograms needed in memory to support them.

If you desire general information on 3-D coordinate systems, mechanical drawing, viewpoints, clipping or hidden surface, refer to Chapter Two. Here you will find an overview of these concepts.

For greater detail, including formulas, turn to Chapter Three of the manual. This section contains the actual matrices and algorithms used to perform the functions defined and discussed in Chapter Two.

Chapter Four provides a complete description of the data base used. This section is pertinent to those of you planning to incorporate any of the utilities in your applications software.

Sections Five through Eight include the local variable lists for the input, manipulation, viewing, and output utilities respectively. Here you will also find a description of each of the utilities.

If you require more information on the additional BASIC and binary utilities, turn to Chapter Nine. A brief discussion of binaries and how to use them also appears in this section.

Chapter Ten contains demonstration programs. These provide examples of how the utilities interact. In addition, several programming tips are provided.

Lastly, a glossary of common terms used in 3-D Graphics is provided at the end of the manual along with the bibliography.

Reference Table

The following reference table lists all of the utilities in alphabetical order, along with their parameter lists, definitions, dependent subprograms and page numbers. If a utility is unique to either the pack 09845-10060 or 09845-10080, the correct pack number will be designated along with the utility. In some cases only the parameters of a utility may differ from one pack to the other. Check the detailed subprogram descriptions found in Chapters 5 through 9 for any "*" subprogram.

Name & Parameter Lists	Definition	Other Subprograms Needed in Memory	Page #
Arc_gen(INTEGER Subscript, SHORT Curve_array(*),Radius, Xa,Xb,Xc,X_center,Ya,Yb,Yc, Y_center,Z_value)	Provides information necessary for arc generation		83
Center(INTEGER Edge(*), N,SHORT Status(*), Transform(*),Vertex(*),X_mid, Y_mid,Z_mid)	Finds approximate center of an object	Arc_gen Curve_setup Multiply	80
Change_point(INTEGER Edge(*),Object(*),Pic, Surface(*),Surface_count, SHORT Offset(*),Range(*), Status_inv(*),Vertex(*))	Changes a single vertex of a surface	Plane_equation	66
Check_arrays(INTEGER Edge(*),Edge_flag,Vertex_flag, SHORT,Vertex(*))	Checks to see if Vertex(*) or Edge(*) is full		64
Circle_by_3_pts(SHORT Radius,Xa,Xb,Xc,X_center,Ya, Yb,Yc,Y_center)	Finds the radius and center point of a circle given three points		83
Clip(INTEGER N,Sur_tmp(*), SHORT Transform(*))	Clips three dimensional objects	Close_poly Level	87
Close_poly(INTEGER First_flag(*),Level, SHORT First(*),Q(*),Save(*))	Closes the clipped polygons of an object	Level	88
(09845-10080) Color_entry(INTEGER Object(*),Pic,Surface(*), SHORT Color(*))	Enters an object's color parameters		61
Constviewpln(INTEGER Error, Digi)	Draws viewplanes on CRT		70
Convert3d(INTEGER Digi, Error,Ne,No,Ns,Nv,Redundfctr)	Converts 3 orthogonal views to 3-D object	Draw2d Findsurf Indbad	73

Name & Parameter Lists	Definition	Other Subprograms Needed in Memory	Page #
(09845-10060) Crt_tra(INTEGER Edge(*), Hid,N,Object(*),Pic,Surface(*), Sur_tmp(*),Trans, SHORT Status(*),Transform(*),Vertex(*), REAL Xvp,Yvp,Zvp)	Overhead for the translations of 3-D objects	Center *Ginput (09845-10060) *Parse_response (09845-10060) Plot Rotate_x Rotate_y Rotate_z Scale Translate Unplot	77
(09845-10080) Crt_tra(INTEGER Edge(*),Hid, N,Object(*),Pic,Surface(*), Sur_tmp(*),Trans,SHORT Color(*),Status(*),Transform(*), Vertex(*),REAL Xvp,Yvp,Zvp)	Overhead for translation of 3-D objects	Center Plot Rotate_x Rotate_y Rotate_z Scale Translate Unplot	78
Curve_setup(INTEGER Edge(*),I,SHORT Radius, Vertex(*),Xa,Xb,Xc,X_center, Ya,Yb,Yc,Y_center,Z_value, REAL Rotation(*))	Rotates an arc or circle into a plane parallel to the X-Y plane	Circle_by_3_pts	82
Database_delete(INTEGER Edge(*),Object,Object(*),Pic, Surface(*),Surface_count, SHORT Vertex(*))	Deletes a surface from the data base		67
Display(A$,INTEGER Select)	Provides for digitizer display		71
Dump	Dumps graphics raster to desired device	Dmpg#b (09845-10060) *Ginput (09845-10060)	98
Draw_char (INTEGER Basic_element(*),Char, Char_index(*),Char_table(*), First_compound,Index, Penc)	Draws characters for lettering		100
Draw2d (INTEGER Digi)	Draws 2D orthogonal views	Constviewpln Recvpcoords	71
Edge_ver_entry(INTEGER Edge(*),Edge_count, Edge_index,Edge_type, SHORT Real_pts(*),Vertex(*))	Enters edge into Edge(*) and Vertex(*)		66

Name & Parameter Lists	Definition	Other Subprograms Needed in Memory	Page #
Enter_arc(INTEGER Edge_count,Surface_flag, SHORT Firstpt(*),Lastpt(*), Offset(*),Range(*),Real_pts(*), Tran_pts(*),REAL A,A(*),B, B(*),C,C(*),D,D(*))	Enters X,Y,Z coordinates of 3 points of an arc	Plane_equation	64
Enter_circle(SHORT Offset(*),Range(*),Real_pts(*), Tran_pts(*))	Enters X,Y,Z coordinate of 3 points of a circle		64
Enter_line(INTEGER Edge_count,Line,Surf_flag, SHORT Firstpt(*),Lastpt(*), Offset(*),Range(*),Real_pts(*), Tran_pts(*)REAL A,A(*), B,B(*),C,C(*),D,D(*))	Enters X,Y,Z coordinates of endpoints of a line segment	Plane_equation	64
Findsurf(INTEGER Edgno, Firstedg,Fsrffg,Prvedg)	Finds all possible 3D surfaces represented by a 2D surface		76
(09845-10060) Ginput(Gprompt$,Var$)	Inputs information while in graphics mode	Binary (09845-10060)	98
(09845-10060) Hidden_surface(INTEGER N, Sur_tmp(*),SHORT Transform(*))	Setup for hidden surface removal	Scan	91
(09845-10080) Hidden_surface(INTEGER N, Sur_tmp(*),SHORT Color(*), Transform(*)	Set up for hidden surface removal;also sets up plotter	Scan	92
Indbad(INTEGER Badplane, Plane,Vtx_edg_srf,Vtxedgptr)	Detects inconsistent vertex information during convertion to 3D	Recvpcoords	75
Initialize(INTEGER Edge(*),N, Object(*),Surface(*), SHORT Vertex(*))	Initialization for 3D data base		62
Initviewpln(INTEGER Digi,Ne2, No2,Ns2,Nv2)	Initialize data base and viewplanes	Constviewpln	70
Input(INTEGER Digi,Mneps, Ne,Ne2,No,No2,Ns,Ns2, Nv,Nv2,Redundfctr)	Digitizer input through digitizer key choice	Convert3d Display Draw2d Dump Initviewpln Modvp One_obj_store Plot Plotter_setup	69

Name & Parameter Lists	Definition	Other Subprograms Needed in Memory	Page #
Letter(INTEGER Basic_element(*),Char_index(*), Char_table(*),Old_font, Spacing(*))	Set up for lettering	Binary (09845-10060) Draw_char Ginput (09845-10060)	100
Level(INTEGER First_flag(*), Level,SHORT First(*),Px,Py, Pz,Pw,Q(*),Save(*))	Keeps track of clipping levels: top, bottom, left, right, hither, yon		88
List(File$,INTEGER List_flag)	List X,Y,Z coordinates of a 3-D object		62
Modvp(S$,INTEGER Add_del, Digi,Mneps,Redundfctr)	Input and modify three orthogonal views	Recvpcoords	71
Multiply(INTEGER Flag,N, SHORT Status(*),Transform(*), Vertex(*))	Multiplies vertices by object's current Status(*) to get transformed points		84
One_obj_entry(File$, INTEGER Edge(*),N,Object, Object(*),Pic,Surface(*), SHORT Status(*),Vertex(*))	Enters one previously stored 3-D object into data base		61
One_obj_store(INTEGER Edge(*),N,Object(*),Pic, Surface(*),SHORT Status(*), Vertex(*))	Stores 3-D object on desired mass storage medium		68
(09845-10060) Parse_response(Responses$(*), Str$,INTEGER Num_of_res)	Parses the response returned from the call to Ginput		99
Perspective(INTEGER N, SHORT Transform(*))	Provides perspective distortion		87
Plane_equation(REAL A,A(*), B,B(*),C,C(*),D,D(*))	Find parameters of a plane Equation: AX+BY+CZ−D=0		68
(09845-10060) Plot(INTEGER Edge(*),Hid,N, Object(*),Pic,Surface(*), Sur_tmp(*),SHORT Status(*), Transform(*),Vertex(*),REAL Xvp,Yvp,Zvp)	Plots 3-D object in data base	Arc_gen Curve_setup Hidden_surface Multiply Viewcoord	89
(09845-10080) Plot(INTEGER Edge(*),Hid,N, Object(*),Pic,Surface(*), Sur_tmp(*),SHORT Color(*), Status(*),Transform(*), Vertex(*),REAL Xvp,Yvp,Zvp)	Plots 3-D object in data base	Arc_gen Curve_setup Hidden_surface Multiply Viewcoord	89

Name & Parameter Lists	Definition	Other Subprograms Needed in Memory	Page #
Plotter_setup(Plotter$, INTEGER Bus_address, Select_code)	Setup for any plotter		96
Recvpcoords(INTEGER Plane, Vtxptr,REAL Xvp,Yvp)	Recover original viewplane coordinates form the 2D Vertex array coordinates		73
(09845-10060) Relist(INTEGER Edge(*), Object(*),Pic,Surface(*), Surface_count,SHORT Offset(*),Range(*),Transform(*))	List X,Y,Z coordinate of an object		62
(09845-10080) Relist(INTEGER Edge(*),Flag, Object(*),Pic,Surface(*), Surface_count,SHORT Color(*),Offset(*),Range(*), Transform(*))	List X,Y,Z coordinates of an object;also used to add color parameters to a surface		63
Rotate_x(SHORT Angle,Rx, Ry,Rz,Status(*))	Calculates 3-D rotation about X-axis		79
Rotate_y(SHORT Angle,Rx, Ry,Rz,Status(*))	Calculates 3-D rotation about Y-axis		79
Rotate_z(SHORT Angle,Rx, Ry,Rz,Status(*))	Calculates 3-D rotation about Z-axis		80
Save_arc(INTEGER Edge_type,SHORT Real_pts(*),Status_inv(*))	Prepares arc to be added to data base		65
Save_circle(INTEGER Edge_type,SHORT Real_pts(*),Status_inv(*))	Prepares circle to be added to data base		65
Save_line(INTEGER Edge_type,SHORT Real_pts(*),Status_inv(*))	Prepares line to be added to data base		65
Scale(SHORT Status(*), X_scale,Y_scale,Z_scale)	Provides the utility to increase or diminish the size of an object		80
(09845-10060) Scan(INTEGER Edgecount, Elist,Polycount,SHORT Edge(*),Poly(*))	Scan algorithm for hidden-surface removal		94
(09845-10080) Scan(INTEGER Edgecount, Elist,Plot,Polycount,SHORT Edge(*),Poly(*))	Scan algorithm for hidden-surface removal		95

Name & Parameter Lists	Definition	Other Subprograms Needed in Memory	Page #
(09845-10080) Store_color(INTEGER Object(*),SHORT Color(*))	Store color information		68
Surface_entry(INTEGER Closed_flag,Edge_count, Edge_index,Surface,Surface(*), Surface_flag,SHORT Firstpt(*), Lastpt(*))	Store edge in Surface array		64
Surface_setup(INTEGER Edge_count,Object,Object(*), Pic,Surface,Surface(*), Surface_count,Surface_flag, REAL A(*),B(*),C(*),D(*))	Overhead for new surface addition		63
Translate(SHORT Dx,Dy,Dz, Status(*))	Provides the utility of moving an object in X, Y, and Z directions		79
Unplot(INTEGER Sur_tmp(*), SHORT Transform(*))	Erases a plot		91
Viewcoord(INTEGER Hid,N, Sur_tmp(*),SHORT Transform(*),REAL Vcx,Vcy, Vsx,Vsy,Xvp,Yvp,Zvp)	Changes point from which an object is viewed	Clip Multiply Perspective	86
Viewpoint(INTEGER View, REAL Xvp,Yvp,Zvp)	Sets up new viewpoint	*Ginput (09845-10060) *Parse_response (09845-10060)	85

Tape Catalogues

Corresponding files between the packs 09845-10060 and 09845-10080 which contain differences, can be distinguished by the last letter of the file name. For example, the last letters of the files DginpB and DginpC are B and C, corresponding to 09845-10060 and 09845-10080.

09845-10084

NAME	PRO	TYPE	REC/FILE	BYTES/REC	ADDRESS
T15		2			
Obj_en		DATA	11	256	5
Cl_enC		DATA	9	256	16
Init		DATA	7	256	25
List		DATA	5	256	32
RlistC		DATA	20	256	37
Surset		DATA	10	256	57
Check		DATA	5	256	67
Surent		DATA	8	256	72
Line		DATA	19	256	80
Circ		DATA	12	256	99
Arc		DATA	27	256	111
Sline		DATA	3	256	138
Scir		DATA	4	256	141
Sarc		DATA	3	256	145
Edgver		DATA	22	256	148
Chngpt		DATA	39	256	170
Surdel		DATA	24	256	209
Obj_st		DATA	9	256	233
Cl_stC		DATA	6	256	242
Plnequ		DATA	4	256	248
DginpC		DATA	59	256	252
DginiC		DATA	13	256	311
DgcvwC		DATA	9	256	324
Digdsp		DATA	7	256	333
Digdrw		DATA	9	256	340
DgmodC		DATA	91	256	349
Digrvc		DATA	7	256	440
DgcvtC		DATA	72	256	447
Digibd		DATA	27	256	519
Digfsr		DATA	11	256	546
REVID		DATA	1	256	557

09845-10085

NAME	PRO TYPE	REC/FILE	BYTES/REC	ADDRESS
T15	2			
CrtC	DATA	21	256	5
Trans	DATA	3	256	26
Rotx	DATA	6	256	29
Roty	DATA	6	256	35
Rotz	DATA	6	256	41
Scale	DATA	4	256	47
Centr	DATA	16	256	51
Crvset	DATA	28	256	67
C_by_3	DATA	16	256	95
Arcgn	DATA	19	256	111
Mult	DATA	7	256	130
VwptC	DATA	6	256	137
Viewcr	DATA	22	256	143
Perspt	DATA	4	256	165
ClipC	DATA	11	256	169
Level	DATA	25	256	180
Close	DATA	20	256	205
PlotC	DATA	37	256	225
Unplot	DATA	4	256	262
HidsrC	DATA	37	256	266
ScanC	DATA	50	256	303
PltstC	DATA	9	256	353
Strip	DATA	4	256	362
DumpC	DATA	7	256	366
LetC	DATA	14	256	373
Drawch	DATA	8	256	387
roman	DATA	65	256	395
stick	DATA	33	256	460
script	DATA	23	256	493
gothic	DATA	55	256	516
DRIV1C	DATA	37	256	571
Keys3D	KEYS	2	256	608
DRIV2C	DATA	18	256	610
KeysTR	KEYS	1	256	628
DRIV3C	DATA	5	256	629
DRIV4C	DATA	6	256	634
REVID	DATA	1	256	640

09845-10064

NAME	PRO	TYPE	REC/FILE	BYTES/REC	ADDRESS
T15		2			
Obj_en		DATA	11	256	5
Init		DATA	7	256	16
List		DATA	5	256	23
RlistB		DATA	16	256	28
Surset		DATA	10	256	44
Check		DATA	5	256	54
Surent		DATA	8	256	59
Line		DATA	19	256	67
Circ		DATA	12	256	86
Arc		DATA	27	256	98
Sline		DATA	3	256	125
Scir		DATA	4	256	128
Sarc		DATA	3	256	132
Edgver		DATA	22	256	135
Chngpt		DATA	39	256	157
Surdel		DATA	24	256	196
Obj_st		DATA	9	256	220
Plnequ		DATA	4	256	229
DginpB		DATA	57	256	233
DginiB		DATA	12	256	290
DgcvwB		DATA	9	256	302
Digdsp		DATA	7	256	311
Digdrw		DATA	9	256	318
DgmodB		DATA	89	256	327
Digrvc		DATA	7	256	416
DgcvtB		DATA	72	256	423
Digibd		DATA	27	256	495
Digfsr		DATA	11	256	522
REVID		DATA	1	256	533

09845-10065

NAME	PRO	TYPE	REC/FILE	BYTES/REC	ADDRESS
T15		2			
CrtB		DATA	24	256	5
Trans		DATA	3	256	29
Rotx		DATA	6	256	32
Roty		DATA	6	256	38
Rotz		DATA	6	256	44
Scale		DATA	4	256	50
Centr		DATA	16	256	54
Cryset		DATA	28	256	70
C_by_3		DATA	16	256	98
Arcgn		DATA	19	256	114
Mult		DATA	7	256	133
VwptB		DATA	7	256	140
Viewcr		DATA	22	256	147
Perspt		DATA	4	256	169
ClipB		DATA	11	256	173
Level		DATA	25	256	184
Close		DATA	20	256	209
PlotB		DATA	37	256	229
Unplot		DATA	4	256	266
HidsrB		DATA	28	256	270
ScanB		DATA	56	256	298
PltstB		DATA	10	256	354
Strip		DATA	4	256	364
DumpB		DATA	9	256	368
Ginput		DATA	26	256	377
Parse		DATA	5	256	403
LetB		DATA	14	256	408
Drawch		DATA	8	256	422
roman		DATA	65	256	430
stick		DATA	33	256	495
script		DATA	23	256	528
gothic		DATA	55	256	551
Dmpg#b		BPRG	4	256	606
Binary		BPRG	20	256	610
DRIV1B		DATA	36	256	630
Keys3D		KEYS	2	256	666
DRIV2B		DATA	19	256	668
KeysTR		KEYS	1	256	687
DRIV3B		DATA	5	256	688
REVID		DATA	1	256	693

Chapter 2
Methodology

Standards

In SIGGRAPH's August 1979 "Status Report of the Graphics Standards Planning Committee", the basic concepts needed in graphics packages were defined. In an attempt to adhere to these criteria the "Three-Dimensional Graphics Utilities" provides:

1. distinct input and output functions
2. similar methods for outputting to a plotter or to an interactive display
3. the concept of data being converted from the world coordinate system to the device or "screen" coordinate system
4. for data display files that contain screen coordinate information
5. for the display file "segments" that may be modified (for example, a vertex, edge, or surface "segment" of an object may be modified.)
6. the user with the functions needed to transform world-coordinate data into screen coordinates

The interaction required between the various utilities is depicted in the following diagram.

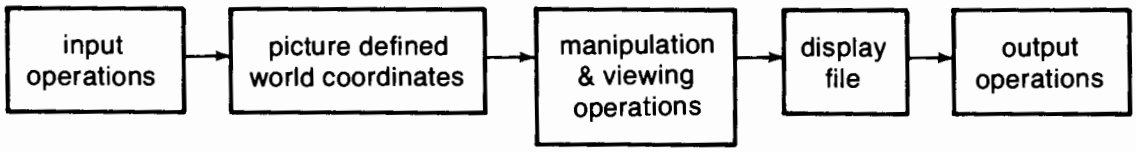


Figure 2.1

3-D Data Bases

Before beginning a discussion on graphics data bases, it is necessary that you understand the cartesian coordinate system. In two dimensions, points (or vertices) of an object are addressed by their X and Y numeric coordinates; the value of X increases from left to right along the X axis, and Y likewise from bottom to top along the Y axis. Below is an example of two points (marked with "•"s) and their (X,Y) coordinate pairs. The point where the two axes intersect has the coordinate pair (0,0) and is called the origin of the coordinate system.

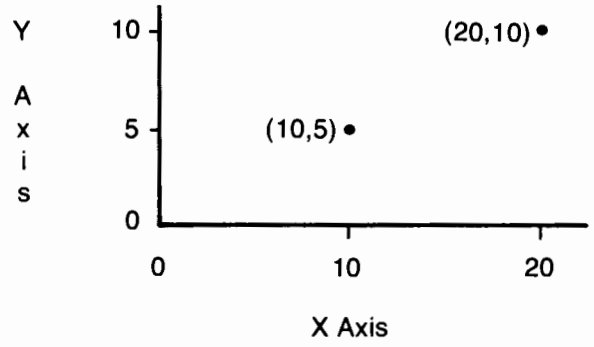


Figure 2.2

The 2-D coordinate system can be easily extended into three dimensions by adding another axis, the Z axis. Each point (or vertex) of a 3-D object is addressed by an X,Y and Z coordinate triplet. There are two 3-D coordinate systems, a right-handed and a left-handed one. They differ only in the orientation of the Z axis.

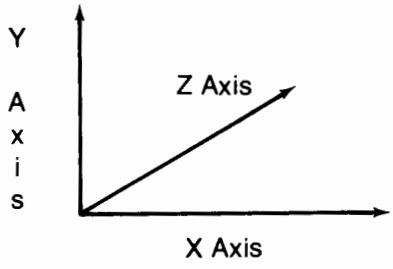


Figure 2.3a: Left-handed System

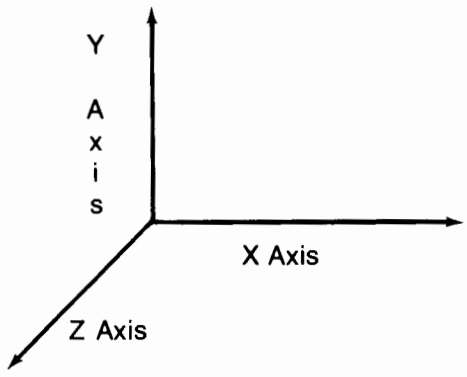


Figure 2.3b: Right-handed System

In order to manipulate and display a three-dimensional object, information about that object must be stored into a data base. In general, there are two types of 3-D data bases, linked vertices and mathematical models.

The "linked vertices" approach is the most common and that used by "Three-Dimensional Graphics Utilities". Each vertex of an object is represented by an X,Y and Z coordinate triplet, from either the left-handed or right-handed coordinate system. The values of each coordinate triplet are stored in an array structure such as this:

X1	Y1	Z1
X2	Y2	Z2
X3	Y3	Z3
.	.	.
.	.	.
.	.	.
Xn	Yn	Zn

Vertex array

Figure: 2.4: Vertex array

In order to draw the object, information describing which vertices have edges between them must be maintained. Therefore, another structure in the data base must include this linking information. This can also be an array. The array elements then are pointers to the vertices that create edges when connected. For example, if you wanted to store a triangle, with (X1,Y1,Z1), (X2,Y2,Z2), and (X3,Y3,Z3) representing the coordinates of its three vertices, the vertex linking array (or edge array) would look like this:

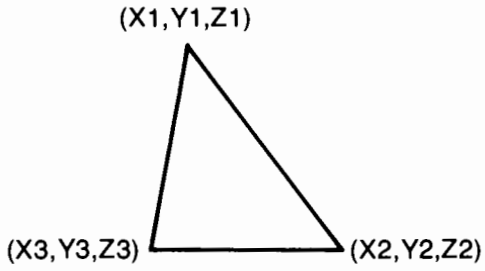


Figure 2.5a

1	2
2	3
3	1
.	.
.	.

Edge array

Figure 2.5b

Now, in order to form surfaces or polygons from these edges, another structure of edge connecting information must be kept. Likewise, to create objects from surfaces, more linking data must be stored. In summary, this sort of data base is a hierarchy of several data structures, each linking together the information of the last level.

Another commonly used technique differs markedly from the linked approach. Objects are stored in patches whose shapes are represented analytically by some mathematical model. Vertices are not used, but rather a set of parameters defining each patch is stored into an array. These parameters can then be used by a function to generate points for the patch. The number of points depends upon the degree of detail needed.

Generally, it is not possible to devise one function that will mathematically describe all patches of an object. Two of the more well known methods are the Bezier and B-Spline models. A complete discussion of these can be found in PRINCIPLES OF INTERACTIVE COMPUTER GRAPHICS, edition 2, by Newman and Sproull.

Input

The need to input and alter data presents a difficult problem when dealing with three-dimensional figures. Three of the most common techniques are described in the following discussion.

Manual Input

The alphanumeric keyboard provides one means of entering data. This "manual" method, although somewhat inconvenient, allows you to enter the precise values you desire for the vertices of a 3-D object. Each vertex is represented by an X, Y, and Z coordinate triplet as described in the previous section.

Digitizer Input

A pleasing alternative is to use a two dimensional input device to aid in entering a 3-D figure. One such method is to enter the X and Y coordinates using the digitizer while entering the Z coordinate from the keyboard. Another way is to enter the object using two views, one of which provides the X and Z coordinates of each vertex and the other of which provides the Y and Z coordinates for that same vertex in the second view. See the example below.

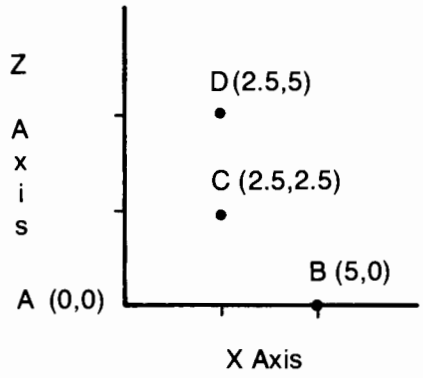


Figure 2.6a: X axis

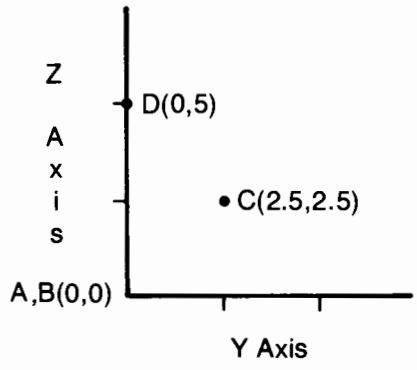


Figure 2.6b: Y Axis

After digitizing coordinates from the X and Z axes view and the corresponding coordinates from the Y and Z axes view, the coordinates of each of the vertices can be determined. In this example the vertices will be:

A – (0,0,0)
B – (5,0,0)
C – (2.5,2.5,2.5)
D – (2.5,0,5)

Another method, and one that is provided in "Three-Dimensional Utilities", is that of using a two dimensional digitizer to enter a 3-D object from a mechanical drawing. The basic requirement is that three orthogonal views of the object be drawn according to standard mechanical drawing principles. (See Chapter Three for more detail.) This method provides a simple means of entering objects from two-dimensional drawings.

If you have a model of an object, a three-dimensional input device can be used. Several such devices have been developed, the simplest of which is an acoustic device. It uses three microphones aligned with the axes. A stylus is used to point to the desired location in space. This stylus generates a small spark and the microphones pick up the sound that is generated. The delay between the spark being created and its being received is used to determine the distance of the point from each axis. (For more information refer to PRINCIPLES OF INTERACTIVE COMPUTER GRAPHICS, edition 2 by Newman and Sproull.)

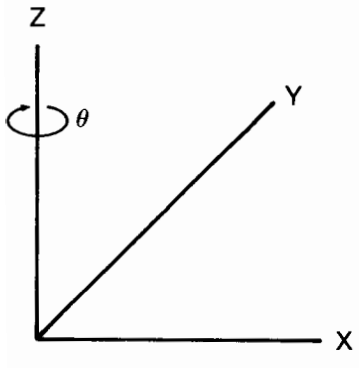
Manipulation

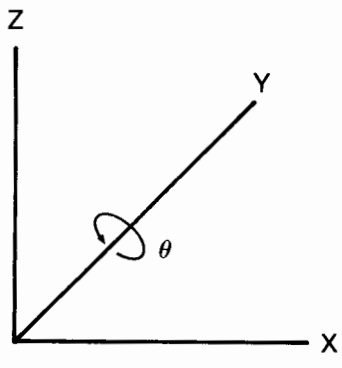
Once the information defining a figure has been stored into a data base, you may wish to perform some transformations on these points. The transformations provided include translation, rotation, and scaling.

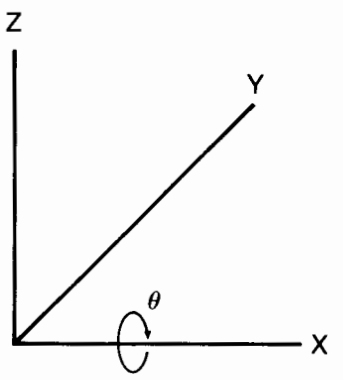
Translation consists of moving all the vertices of an object a certain number of units along the X, Y, and Z axes. For example, if one vertex of an object is located at (20,40,100) and the object is translated 100 units along the X axis, 20 units along the Y, and 30 along the Z, the new location of that vertex will be (120,60,130).

Rotation provides the facility to move an object about a point in space. If the point happens to be the center of the object, the object will appear to remain as a fixed location in space as it rotates. The rotation will allow you to view different surfaces. However, if the point that the object is rotated about is an arbitrary point in space, the affect achieved is similar to that of the earth (an object) rotating about the sun (a fixed point).

The rotation angle you desire is determined by measuring clockwise about the origin from a point in the positive direction along the axis you wish to rotate about [3]. See the diagram below:







An angle θ is measured clockwise when looking toward the origin from a point on the rotation axis.

Figure 2.7a [3] **Figure 2.7b** [3] **Figure 2.7c** [3]

For further detail, refer to the descriptions in Chapter 3.

Scaling allows you to enlarge or diminish an object. Be aware that an object that is scaled may appear to "move" since the center will not remain the same. Measures can be taken to prevent this from happening. Turn to Chapter Three for a complete discussion.

Viewing

To display a two-dimensional object, it is only necessary to draw lines between connected vertices. Most graphics systems provide automatic clipping so that no error occurs if you try to draw outside the hard clip limits of the plotter.

When displaying 3-D objects, a two-dimensional "canvas" is still being used, but special techniques must be applied to produce a realistic scene. For example, when an artist sketches a scene with a road going off into the distance, the road seems to get narrower. In reality it is the same width in all places, but some "depth clues" must be given to the viewer so the scene does not appear flat. The artist also selects other viewing information such as where the observer is standing and how much peripheral vision he has, and at what point the road vanishes. If there was a car on the road, it would conceal some of the road and scenery, conveying that the car is closer to the viewer.

The computer must mathematically calculate these visual effects to produce a realistic representation of an actual scene. This process begins with the observer choosing a point in a three-dimensional coordinate system from which he wishes to view an object or scene. This point is called the "viewpoint". Imagine yourself standing in front of a window looking outside. The place you are standing is your viewpoint and the window is your "viewport" to the outside world. Note that your peripheral vision does not continue straight in front of you, but rather increases to the right, left, up and down, for as far as you can see. This viewing area outside the window is often referred to as a truncated viewing pyramid. It is bounded by right, left, top, bottom, hither and yon clipping planes. Objects lying outside of the viewing pyramid must be clipped from the observer's view. In computer graphics, think of the viewport or hither clipping plane as your CRT or plotter.

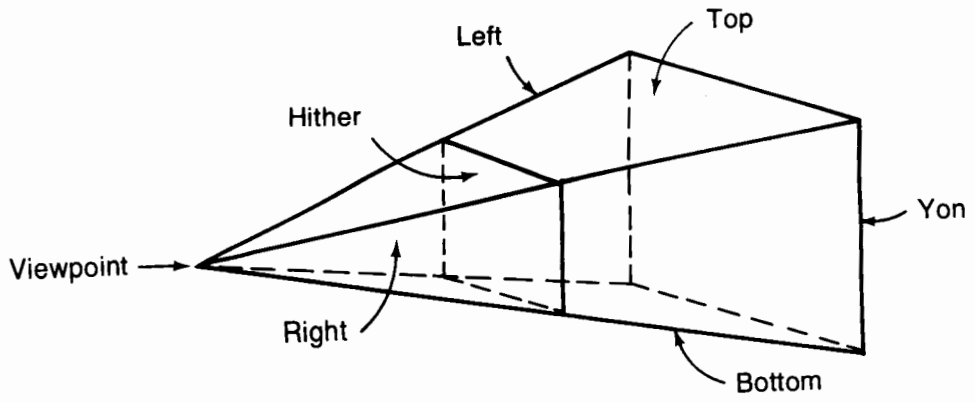


Figure 2.8: Truncated Viewing Pyramid

Once a scene has been clipped for viewing, it can be displayed. The simplest display technique is that of parallel projection. Lines of an object that are parallel in reality remain parallel in the display. This type of display offers no depth information to the viewer. If, however, perspective projection is used, parallel edges of an object may appear to converge and distant objects appear smaller than near ones. Thus, an illusion of depth is created.

Output

In order to output the scene, one of two methods can be used. A quick way to display a scene is in "wire-frame" form. By wire-frame is meant that all surfaces of an object are treated as if they were transparent. The outline of each surface is drawn and no consideration is given as to which surface is in front of another. A not so simple task is that of hiding surfaces that are partially or completely obstructed from view by other surfaces. This technique is commonly referred to as "hidden-surface removal". It is a time-consuming process and is usually not done until the viewer is ready for a hard copy output. Below is an example of a cube drawn in wire-frame form and with hidden surfaces removed.

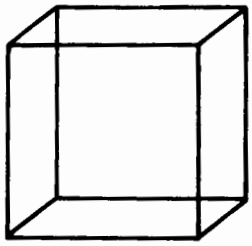


Figure 2.9a: Wire-frame

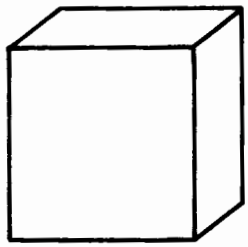


Figure 2.9b: Hidden Surfaces Removed

All of the above processes are provided in this pack. Other advanced display techniques include shadowing, highlighting and shading. A discussion of these may be found in PRINCIPLES OF INTERACTIVE COMPUTER GRAPHICS, edition 2, by Newman and Sproull.

Chapter 3
Formulas

A basic knowledge of graphics theory is helpful, but a clear presentation of the necessary algorithms is a must. For this reason, the formulas and methods used to do three-dimensional input, manipulation, viewing, and output in this pack are included in this section.

Input

Manual Input

The "manual" method of data entry found in "Three-Dimensional Utilities" provides three basic components to use in the construction of surfaces. These are line, arc, and circle.

Line Segments

When the "line" primitive is used, you must simply provide X, Y, and Z coordinates of the endpoints for each line segment desired in a surface. If more than three lines are entered for a surface, checking is done to insure that all of the segments lie in one plane. The algorithm used to determine the equation of a plane is called the "Three-point form". Given three non-collinear points:

P1 = (X1,Y1,Z1)
P2 = (X2,Y2,Z2)
P3 = (X3,Y3,Z3)

The equation of the plane is then $AX + BY + CZ - D = 0$ where:

$$A = \begin{vmatrix} Y1 & Z1 & 1 \\ Y2 & Z2 & 1 \\ Y3 & Z3 & 1 \end{vmatrix}$$

$$B = \begin{vmatrix} Z1 & X1 & 1 \\ Z2 & X2 & 1 \\ Z3 & X3 & 1 \end{vmatrix}$$

$$C = \begin{vmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{vmatrix}$$

$$D = \begin{vmatrix} X1 & Y1 & Z1 \\ X2 & Y2 & Z2 \\ X3 & Y3 & Z3 \end{vmatrix}$$

Circles and Arcs

If either a circle or arc is desired, you must provide three X, Y, Z coordinate triplets. These X, Y, and Z values are stored into the data base as entered. However, in order to be displayed, the center point and radius of the circle or arc must be determined. The subprogram, Circle_by_3_pts, is used to find the center and radius. Before Circle_by_3_pts can be used, the three points determining the circle (arc) must be rotated into a plane parallel to the X-Y plane. This is done by the subprogram Curve_setup. The steps needed to accomplish this are as follows. First, the equation of the plane that the circle or arc lies in is determined. Next, the angle between the Y-Z plane and the plane that the circle or arc lies in is determined and the circle is rotated about the Z-axis. Once this is accomplished, the intersection of the plane of the circle and the X-Y plane is a line parallel to the Y-axis. The angle between the plane of the circle and the X-Y plane is found and the circle (arc) is rotated that number of degrees about the Y-axis. At this time, the plane of the circle will be parallel to the X-Y plane. The coordinates of the three points defining the circle in this plane can now be used to calculate the center and radius of the circle. Refer to Figures 3.1a through 3.1c below.

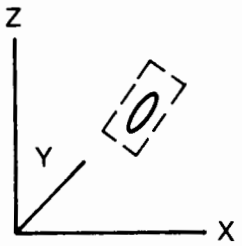


Figure 3.1a: Circle in Original Position

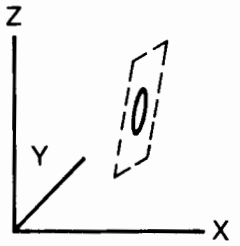


Figure 3.1b: Circle Rotated About Z Axis

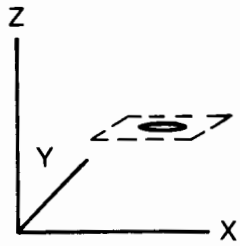


Figure 3.1c: Circle Rotated About Y Axis

A matrix containing the rotations needed to transform the circle into a plane parallel to the X-Y plane must be saved. Its inverse is used to display the circle at its original orientation in space.

Circles

A circle can actually be considered to be a surface (think of it as a disk). Since three points determine a plane, no checking is needed to insure that the points are coplanar. The algorithm used to generate the center and radius of the circle is found in the subprogram called Circle_by_3_pts. The three points entered define a triangle in space. The triangle is inscribed in the desired circle. The length of each side is calculated and the radius of the circle is then found by applying the formula:

$$r = \frac{abc}{4s(s-a)\tan(A/2)}$$

In the above formula,

$$s = \frac{a+b+c}{2}$$

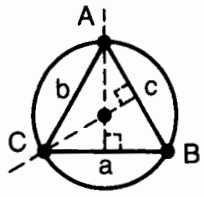


Figure 3.2:

The center point is the intersection of the perpendicular bisectors of any two sides of the inscribed triangle. The equations of the perpendicular bisectors of $\overline{AB}$ and $\overline{BC}$ are derived and the point of intersection of these equations is the center of the circle. Special cases involving horizontal and vertical lines are also handled by the algorithm. For more accurate results, you should attempt to choose the points, A, B, and C approximately equidistant from one another.

Arcs

An arc is also determined by three points. The first point entered is an endpoint, the second is any point on the arc other than its endpoints, and the third is the other endpoint of the arc.

Arcs, unlike circles, cannot be used as surfaces since they are not "closed". They can be used, however, in conjunction with line segments or arcs to create surfaces. Of course, this requires that checking be done to insure the surface being constructed is planar. Similar methods to those described for line segments are used.

In order to display an arc, the subprogram Arc_gen is used. All arcs are drawn in a clockwise manner from the first point entered to the last. This means that if the points were entered counterclockwise, some manipulation must be performed before the arc can be drawn. Observe in Figure 3.3a that the arc was entered clockwise. In Figure 3.3b, it was entered counterclockwise.

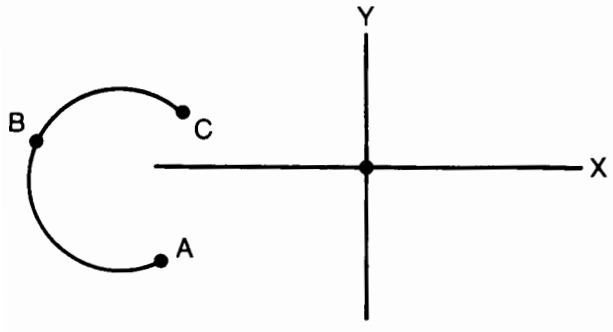


Figure 3.3a

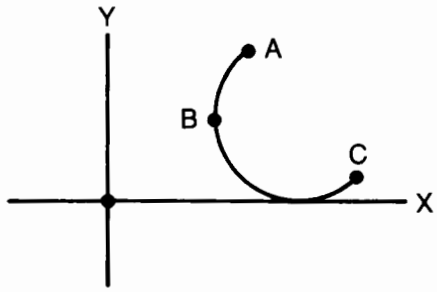


Figure 3.3b

To guarantee arcs are displayed in clockwise fashion, a complicated process is followed. As described earlier, all arcs must be rotated into a plane parallel to the X-Y plane before the center and radius can be determined. Next, processing is done to insure the arc is drawn clockwise. Since the arc is now in a plane parallel to the X-Y plane, only the X and Y coordinates need be used in further computations. You may think of the arc as being two-dimensional and the coordinate system as being the standard cartesian system.

To begin with, the center of each arc must be translated to the origin of the X-Y coordinate system. See figures below.

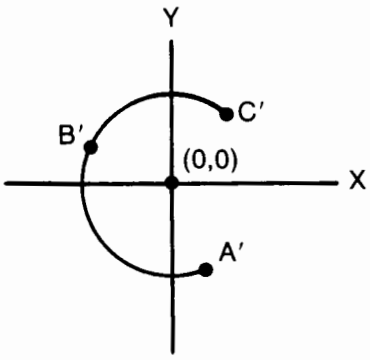


Figure 3.4a

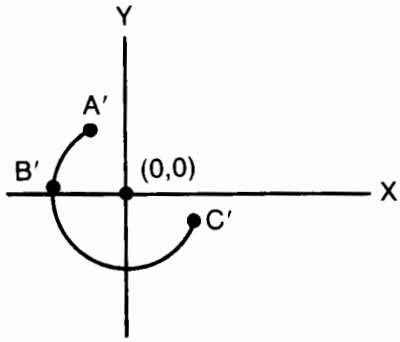


Figure 3.4b

Next, (referring to the diagrams below) the angle between segment OA′ and the X axis, OB′ and the X axis, and OC′ and the X axis are determined. These angles will be referred to as Theta1, Theta2, and Theta3 respectively.

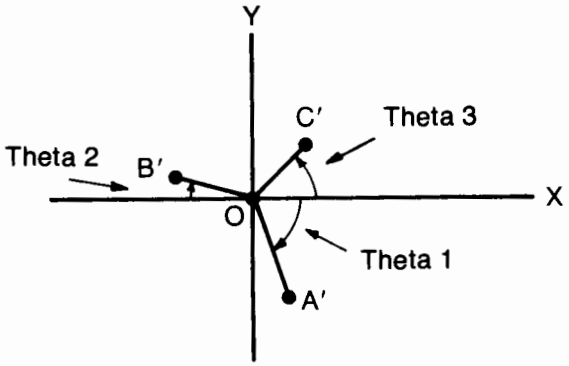


Figure 3.5a

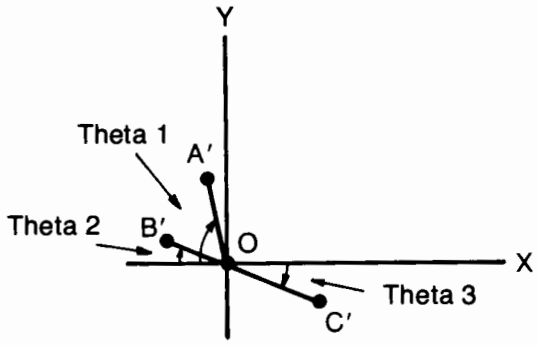


Figure 3.5b

In general, an angle between the X-axis and a line segment with one endpoint at the origin is determined by taking the Arctangent of | X/Y |, where X and Y are the coordinates of the other endpoint of the line segment.

If the X and Y coordinates of the endpoint are both negative, then the point must lie in the third quadrant. Therefore, to determine the number of degrees from the positive Y-axis to the line segment joining the origin to the point (X,Y), you must subtract Theta from 270. The result will be referred to as Theta'. See Figure 3.6 below.

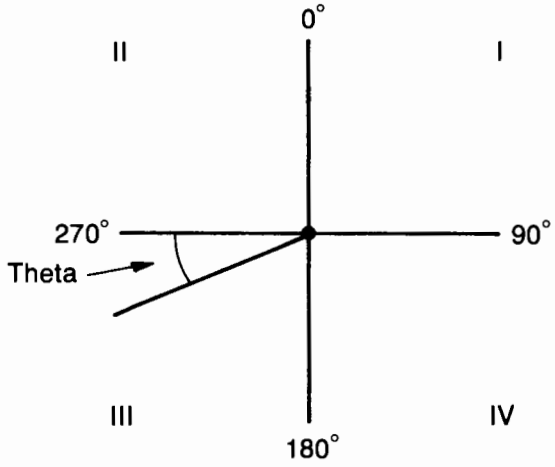


Figure 3.6: Theta' = 270 − Theta

Therefore, if (Y<0) and (X<0), Theta' = 270 − Theta. Using the same process you can derive the following rules:

1. If (Y>0) and (X>0) then Theta' = 90 − Theta
2. If (Y>0) and (X<0) then Theta' = 270 + Theta
3. If (Y<0) and (X<0) then Theta' = 270 − Theta
4. If (Y<0) and (X>0) then Theta' = 90 + Theta

Note that special cases exist when an endpoint lies on the X or Y axis. These cases are:

If the endpoint lies on the positive Y axis, Theta' = 0.
If the endpoint lies on the positive X axis, Theta' = 90.
If the endpoint lies on the negative Y axis, Theta' = 180.
If the endpoint lies on the negative X axis, Theta' = 270.

For a more complete understanding, an example showing the arcs in Figures 3.4a and 3.4b along with plausable values for Theta' are given below.

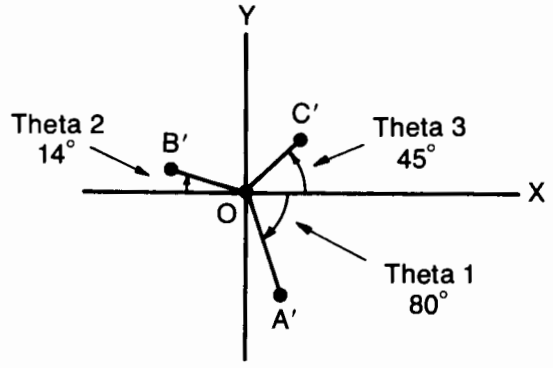


Figure 3.7a

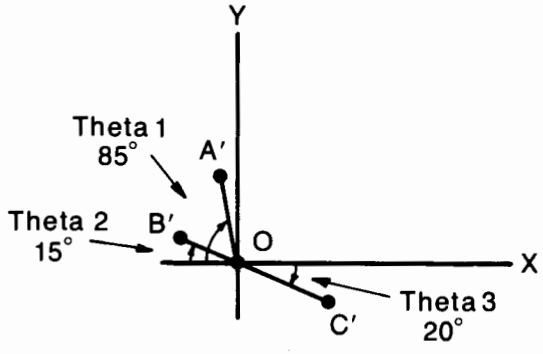


Figure 3.7b

Using the angles provided in Figures 3.7a and 3.7b and the above algorithm for determining Theta1′, Theta2′, and Theta3′ results in:

Theta1′ = 90 + 80 = 170
Theta2′ = 270 + 14 = 284
Theta3′ = 90 − 45 = 45

Figure 3.8a

Theta1′ = 270 + 85 = 355
Theta2′ = 270 + 15 = 285
Theta3′ = 90 + 20 = 110

Figure 3.8b

You are now ready to determine if an arc is already in clockwise form or if the endpoints need to be switched. If Theta1′ < Theta2′ and Theta2′ < Theta3′ then the points were entered in clockwise fashion and no changes are necessary. If, however, Theta3′ < Theta2′ and Theta2′ < Theta1′, the points were entered counterclockwise. (See 3.1b). The same is true if Theta1′ < Theta3′ but Theta2′ does not lie between them. In these cases, before the arc is drawn, the first and last endpoints must be switched.

In the cases where it appears that the second point of the arc does not lie between the first and last endpoint, special attention must be taken. This problem arises when the arc crosses the positive Y axis. In order to use a "FOR-NEXT" loop to draw the arc, an offset must be added in cases where Theta2′ does not lie between Theta1′ and Theta3′ and at the same time, Theta3′ < Theta1′. The offset in such a situation is 360 degrees, otherwise the offset is 0 degrees.

When circles and arcs are plotted on the CRT or on a device such as the 9872, they must be generated by plotting to a sequence of points n degrees apart. (Where n is chosen sufficiently small to generate a smooth curve). For example, circles in this pack are generated with points ever 15 degrees. For arcs, however, the number of degrees in the arc is calculated and then an appropriate value for n is computed.

The number of degrees contained in each arc can be determined by using the angle information discussed in the above section. The number of degrees in an arc will be referred to as its Range.

Range = | Theta3′ + Offset − Theta1′ |

Once the range is determined, an appropriate step size for drawing the arc is found. The following algorithm is used:

If Range < 360 then Stepsize = Range/24
If Range < 270 then Stepsize = Range/18
If Range < 180 then Stepsize = Range/12
If Range < 90 then Stepsize = Range/6

Observe that the range is divided by the number of segments that you wish to use in approximating a particular curve. This gives you the number of degrees between the points used to approximate each arc. It is also possible to choose a fixed number of degrees for the step size. This eliminates some computation.

The following examples show the range, offset, and step size for drawing the arcs displayed in Figures 3.3a and 3.3b.

Offset = 360
Range = | 45 + 360 − 170 | = 235
Stepsize = 13.05

Figure 3.9a

Offset = 0
Range = | 110 + 0 − 355 | = 245
Stepsize = 13.61

Figure 3.9b

Digitizer Input

The 9874A can be used as a means of data entry for 3-D objects. This method requires that you:

1. use 3 orthogonal views of the object you wish to enter. These are the top, front, and side view.
2. draw the object according to "accepted mechanical drawing principles".
3. center each view in its view plane.
4. enter every surface of the object in each view – "hidden" surfaces as well as visible surfaces.

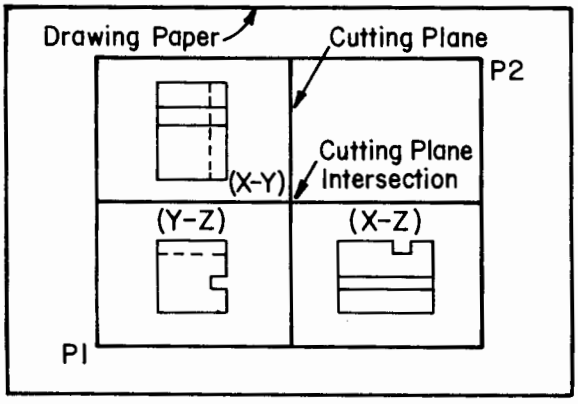


Figure 3.10: Orthogonal View Drawing

Dotted lines are used to indicate surface edges that exist but are not visible from a given view.

Given the three orthogonal views shown in Figure 3.10, the resulting 3-D object is:

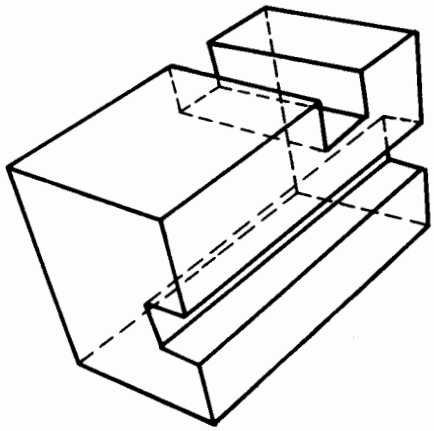


Figure 3.11

Three View Mechanical Drawings

Mechanical drawings can be very complex and the art of creating them can take months to perfect. For an in-depth discussion, you should refer to texts on the subject. This section is only meant to familiarize you with some general concepts.

The easiest method of introduction to this topic is through examples. Figure 3.12 depicts a three dimensional object that can be represented by a drawing showing its three orthogonal views – top, front, and side (right side).

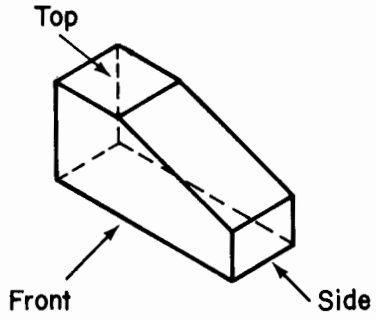


Figure 3.12

Begin by determining the spacing of the views on your drawing surface. The top view will appear in the upper left hand corner of your drawing space, the front view in the lower left, and the side view in the lower right. Assume the measurements of your object are those depicted in Figure 3.13.

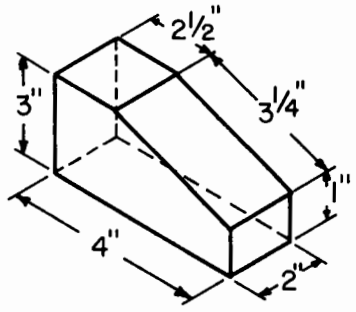


Figure 3.13

The width of the front view is 4" and the depth of the side view is 2". If your drawing work space is 8½" by 11", choose a space W between the two views (front and side). Lets choose W to be 1½.

Next add 4", 2" and 1½" and subtract the total from the work space width of 11". Divide the result by 2 to get the value of X. X represents the width of the left and right margins of the drawing.

Mark off these measurements with vertical tic marks on your drawing paper. See Figure 3.14.

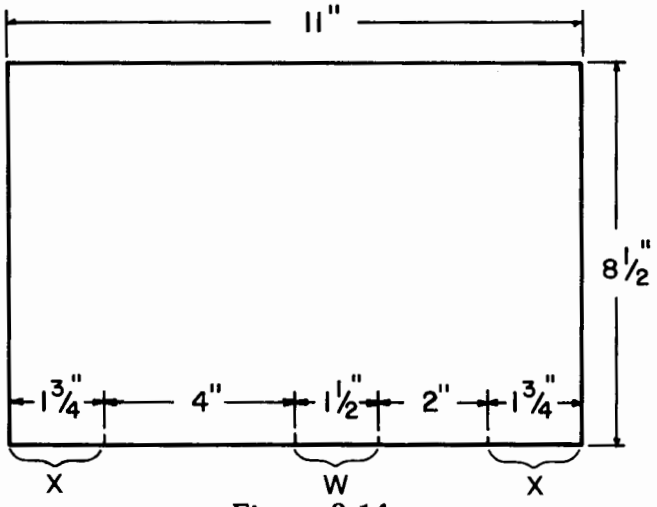


Figure 3.14

Referring back to Figure 3.13, the depth of the top view is 2" and the height of the front view is 3". Choose an appropriate value for the spacing between these views. Call it Y. Let Y equal 1" in this example. Note that the spacing Y does not have to equal that of W.

As before, add the three values. Subtract the total from 8½" and divide the result by 2 to get the value Z. Mark these vertical spacing measurements with tic marks along the left side of your work space.

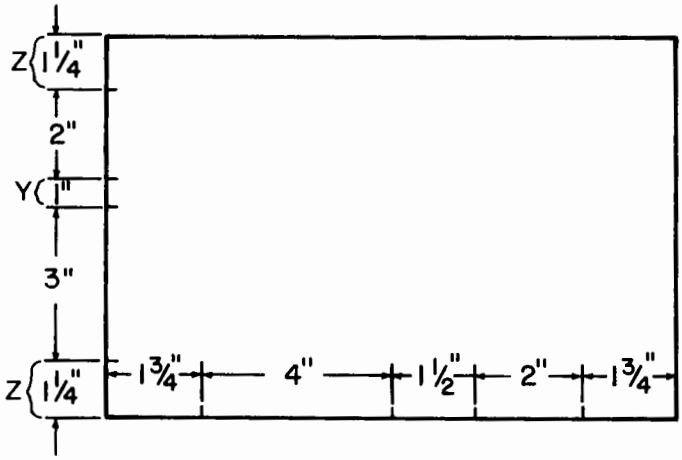


Figure 3.15

Now you are ready to draw the three views of the figure. For best results, try drawing all views simultaneously rather than completing one view at a time. This "first pass" should be drawn lightly for easy revision.

When finished, go back and darken the visible edges in each view and dash the hidden edges. Observe that no dashed lines segments are needed in this particular drawing. See Figure 3.16.

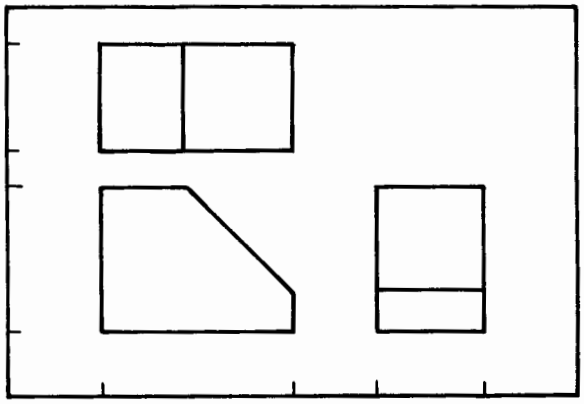


Figure 3.16

If, however, the original orientation of the figure were:

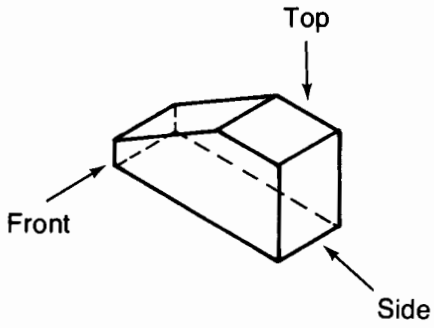


Figure 3.17

the three orthogonal views would be:

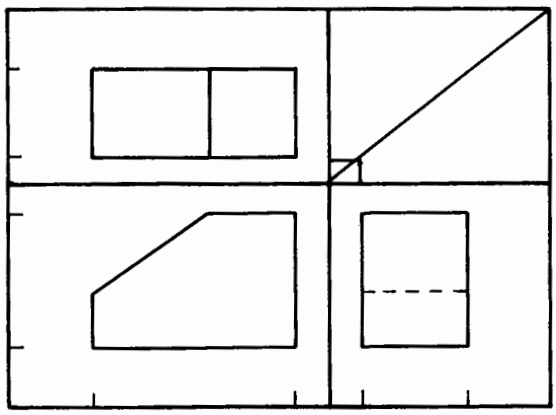


Figure 3.18

Note that a three-dimensional object can be represented by a number of different three-view drawings. Certain views, however, can present ambiguous information to the algorithm that converts the three 2-D views into a 3-D object.

For example, when given the following three orthogonal views:

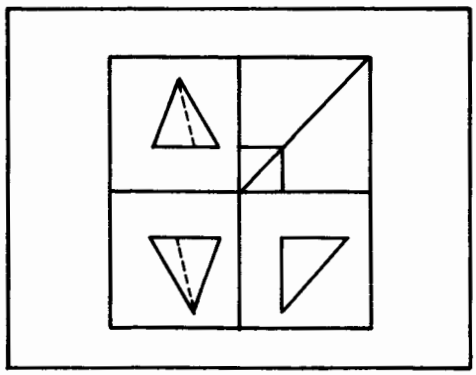


Figure 3.19

the result may be a figure that contains extraneous surfaces due to the ambiguity of the information provided by the previous three views. A possible result is:

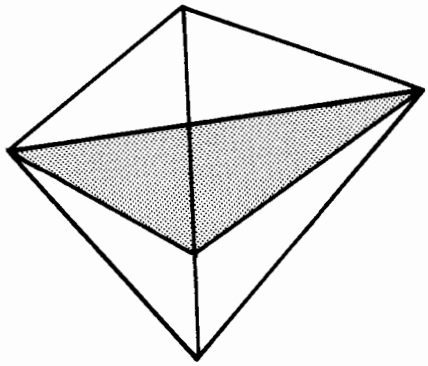


Figure 3.20

The subprogram generates all possible surfaces that can exist given the vertex, edge and surface information entered. Therefore, you must use three orthogonal views that are "unambiguous". For example, a tetrahedron can be generated correctly if the three orthographic views shown below were entered.

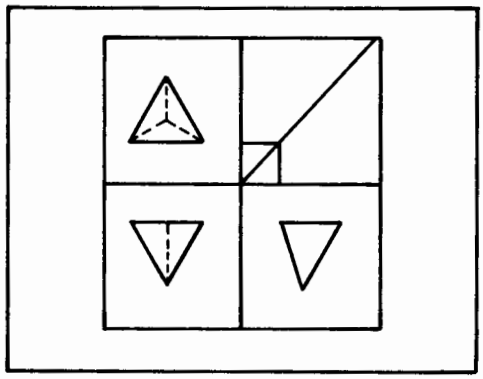
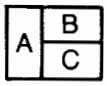


Figure 3.21

Another rule to remember is that adjacent areas must represent surfaces at different levels. If the top view of an object is:



TOP VIEW

Figure 3.22

The object could be one of the following:

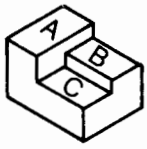


Figure 3.23a

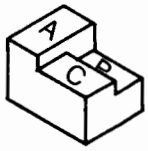


Figure 3.23b

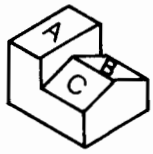


Figure 3.23c

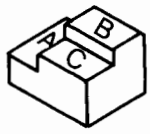


Figure 3.23d

Observe that no two adjacent areas ever lie in the same plane.

Converting Orthogonal Views to a Three-Dimensional Figure

Once the orthogonal views of a figure have been digitized, you are ready to convert the two-dimensional data to three-dimensional information. During this process, the subprogram Convert_3d checks for consistency between views. At each stage of the conversion process any inconsistencies that are found will be displayed. At this point, a modification mode can be entered and corrections made to the view or views that show discrepencies. If no inconsistencies are found, the object is stored as is. The process followed by Convert_3d is described in the following paragraphs.

Given a three orthogonal view drawing, a primary, secondary, and tertiary plane are chosen. First, the X-Y plane can be chosen as the primary plane, the Y-Z as the secondary plane, and X-Z as the tertiary plane. With this orientation, the primary coordinate will be the Y value, the secondary coordinate the Z value, and the tertiary coordinate the X value.

For each vertex, Primpt, in the primary plane, all the verticies in the secondary plane that have a primary coordinate equal to the primary coordinate of Primpt must be found. These vertices are saved in an array called Commonvtx. If none can be found, an "inconsistency" has been detected between the primary and secondary plane.

For each vertex, Cpt, in Commonvtx, all the vertices in the tertiary plane, referred to as Tertpt, that have a secondary coordinate equal to the secondary coordinate of Cpt must be found. If none exist, an "inconsistency" exists between the secondary plane and tertiary plane.

All verticies satisfying the above conditions are examined for one that has a tertiary coordinate equal to the tertiary coordinate of the primary point, Primpt. If none exists, there is an "inconsistency" between the primary plane and the tertiary plane. If one does exist and the primary plane happens to be the X-Y plane, then a point is added to the Vertex array. The primary coordinate of this point is the primary coordinate of Primpt. The secondary coordinate is the secondary coordinate of the Commonvtx point, Cpt. The tertiary coordinate is the tertiary coordinate of the primary point, Primpt.

If the primary plane is either the Y-Z or the X-Z plane, do not add a vertex to the Vertex array since it should have already been added by this time.

When a point is added to the Vertex array, an element is also added to the Vtxlink array. The Vtxlink element is a pointer for each vertex to the vertex in each plane from which it was derived. This information is needed in the "build edge" section of the subprogram.

The above process is repeated with the Y-Z plane being the primary plane, X-Z the secondary and X-Y the tertiary plane. The primary coordinate is the Z, the secondary coordinate is X and the tertiary coordinate is Y.

Lastly, X-Z is considered to be the primary plane, X-Y the secondary and Y-Z the tertiary plane. At this time, the primary coordinate is X, the secondary coordinate is Y, and the teritary coordinate is Z. Again, the process is repeated.

The following flowchart depicts the algorithm used. Three iterations are needed; one for each of the three different plane orientations assigned.

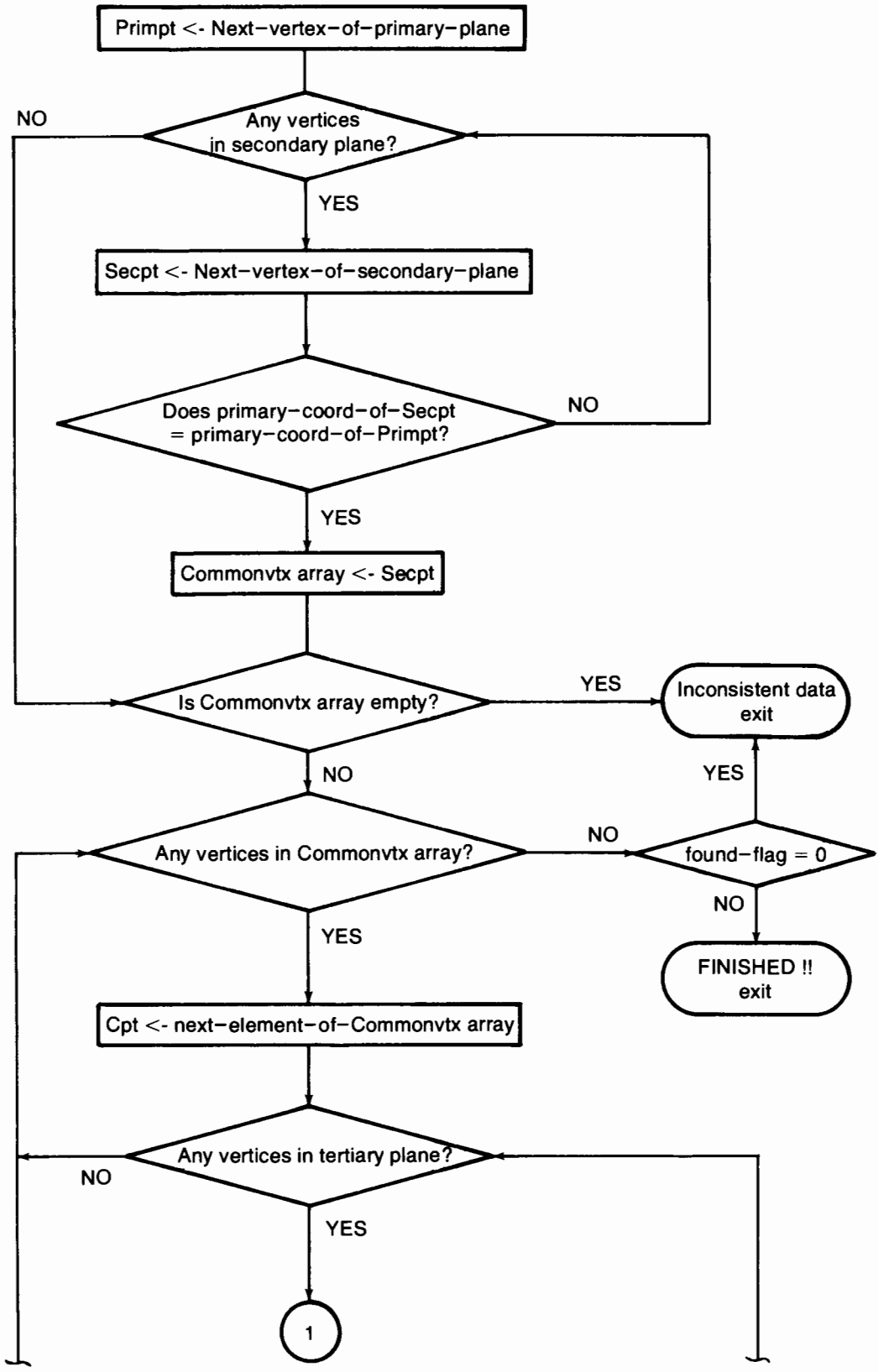


Figure 3.24

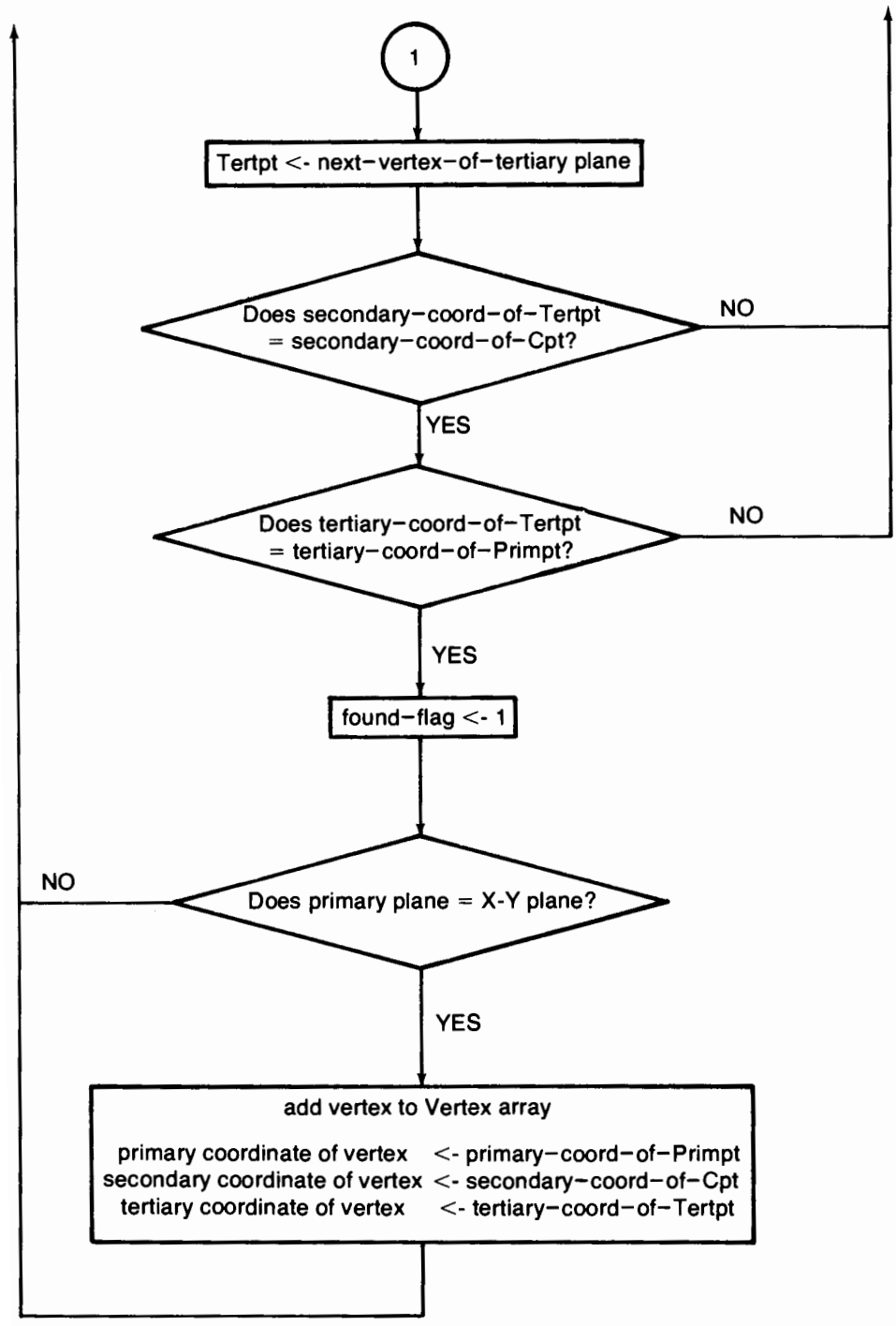


Figure 3.24 cont.

Manipulation

Three-dimensional objects can be moved from one point to another, rotated about an axis or an arbitrary point, and enlarged or diminished in size. These manipulations are accomplished by using transformation matrices. A transformation matrix is a single mathematical result that can be applied to a point (x,y,z) to generate a new point (x',y',z'). The primitive transformations are called translation, rotation, and scaling. Each of these can be expressed as a single matrix. More complex transformations can also be expressed as a single matrix that is the result of concatenating two or more primitive transformation matrices. If you apply this single matrix to a point or set of points, you will obtain the same results as you would obtain by applying each transformation in sequence.

Translation

The translation of a object on the screen is accomplished by adding a Δ to each coordinate. Tx represents the number of units the object is to move along the X axis, Ty represents the number of units the object is to move along the Y axis, and Tz represents movement along the Z axis.

When a matrix containing the vertices of an object to be translated is multiplied by the 4 by 4 translation matrix, the result is a matrix containing the translated vertices.

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{bmatrix}$$

Figure 3.25 [3]

Rotation

In three-dimensional rotation, an object may be rotated about a point with respect to a particular axis. This is accomplished by translating the point to the origin of the coordinate system, performing the desired rotation, and translating the point back to its original position. In the "demos" provided in this pack, all rotations take place about the computed "center" of the object. This "center" is approximated by finding the maximum and minimum values for the X, Y, and Z coordinates of the object and computing the mean for each.

A rotation of θ degrees about any of the coordinate axes is accomplished with the following matrix multiplications. (Refer to Chapter 2 of the manual to determine the sign of the angle θ that you desire.)

Rotation about the X axis:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.26 [3]

Rotation about the Y axis:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.27 [3]

Rotation about the Z axis:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.28 [3]

Scaling

The scaling of an object is achieved by multiplying each X coordinate of an object's vertices by the scaling factor Sx, each Y coordinate by Sy and each Z coordinate by Sz.

Note that when scaling objects containing arcs as edges or when scaling circles, the Sx, Sy, and Sz scaling factors must be equal since ellipses are not provided for in the data base.

Observe that scaling an object will change the location of an object in addition to changing its size. In the figure below, the triangle is scaled by a factor of 2 in the X direction, 3 in the Y direction and 4 in the Z direction.

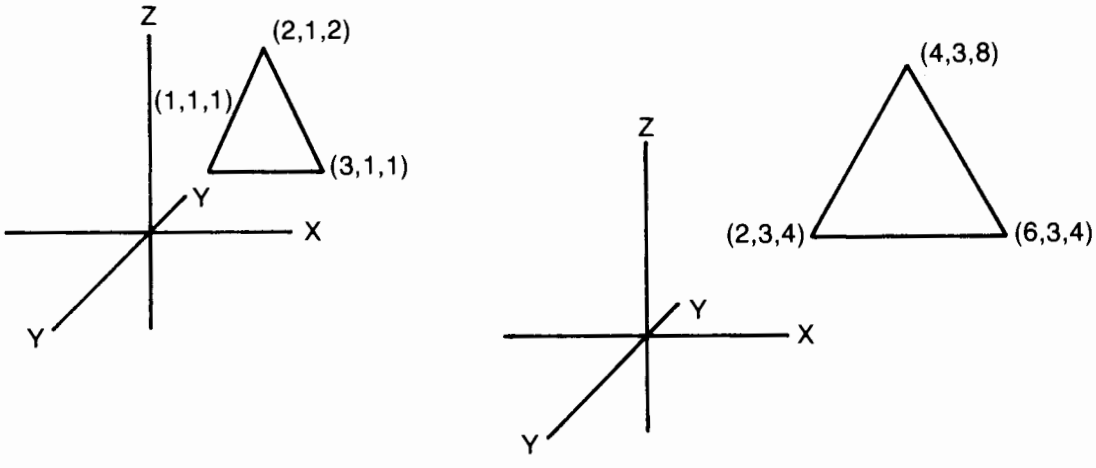


Figure 3.29 [3]

In order to scale an object without appearing to move it, it is necessary to find the center of the object before scaling the object, find the center of the scaled object, and translate from the object's "new center" to the objects "old center". Demo2 demonstrates this process.

The scaling matrix used is:

$$[x'y'z'1] = [x\ y\ z\ 1] \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.30 [3]

Viewing

Viewpoint

In order to establish a viewpoint, five transformations are needed. These transformations are then concatenated into one 4 by 4 viewing matrix, V, whose purpose is to convert vertices in a right-handed world coordinates to a left-handed eye coordinate system. The origin of the eye coordinate system is the chosen viewpoint [3].

The first transformation, T1, translates the origin of the world coordinate system to the viewpoint (Xvp, Yvp, Zvp), creating a "prime" coordinate system.

$$T1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -Xvp & -Yvp & -Zvp & 1 \end{bmatrix}$$

Figure 3.31a [3]

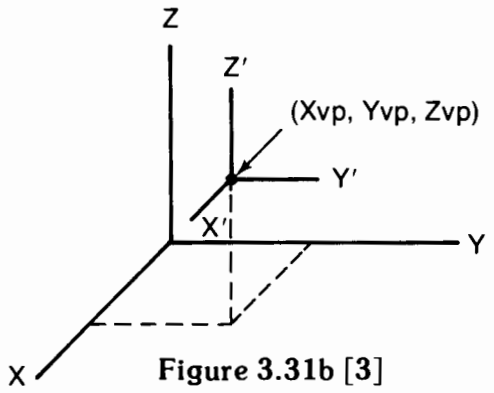


Figure 3.31b [3]

The second transformation, T2, rotates the "prime" coordinate system around the X' axis by −90 degrees.

$$T2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.32a [3]

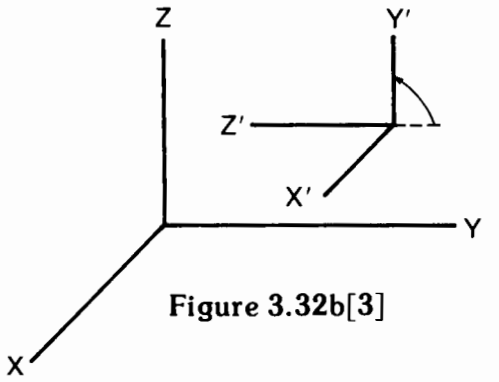


Figure 3.32b[3]

The third transformation, T3, rotates the "prime" coordinate system around the Y′ axis by θ degrees, so that the point (0,0,Zvp) will lie on the Z′ axis.

$$\cos(\theta) = Yvp / \sqrt{Xvp^2 + Yvp^2} \qquad \sin(\theta) = Xvp / \sqrt{Xvp^2 + Yvp^2}$$

$$T3 = \begin{bmatrix} -\cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & -\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.33a [3]

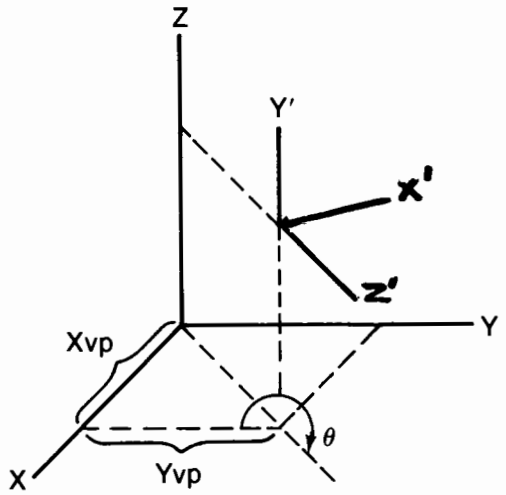


Figure 3.33b [3]

The fourth transformation, T4, rotates the "prime" coordinate system about the X′ axis by $\propto$ degrees, so that the origin of the world coordinate system will lie on the Z′ axis.

$$\cos(\propto) = \sqrt{Xvp^2 + Yvp^2} / \sqrt{Xvp^2 + Yvp^2 + Zvp^2} \qquad \sin(\propto) = Zvp / \sqrt{Xvp^2 + Yvp^2 + Zvp^2}$$

$$T4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\propto & \sin\propto & 0 \\ 0 & -\sin\propto & \cos\propto & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.34a [3]

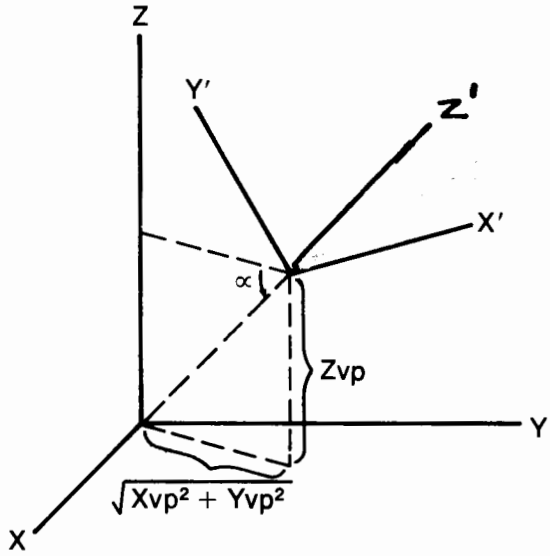


Figure 3.34b [3]

The fifth transformation, T5, reverses the sense of the Z′ axis with a scaling matrix, to produce a left-handed coordinate system. The resulting "prime" coordinate system is called the eye coordinate system.

$$T5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.35a [3]

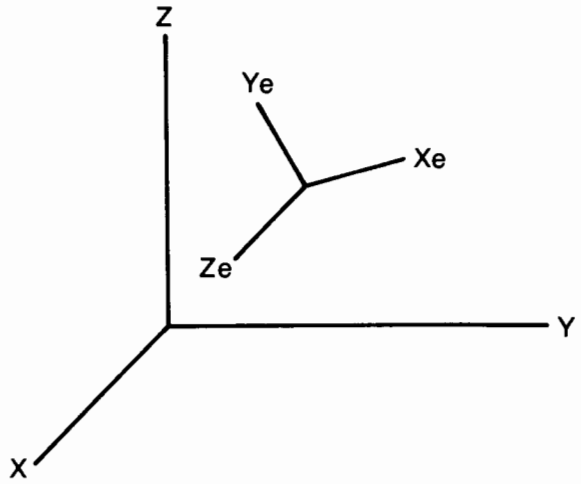


Figure 3.35b [3]

The concatenated viewing matrix V then is: V = T1*T2*T3*T4*T5.

Clipping

Since not all surfaces of an object may be within the truncated viewing pyramid, a clipping routine is necessary. The first step in clipping is to convert the eye coordinates into a normalized range from 0 to 1. This transformation depends on the values S, D and F from the truncated viewing plane, shown below.

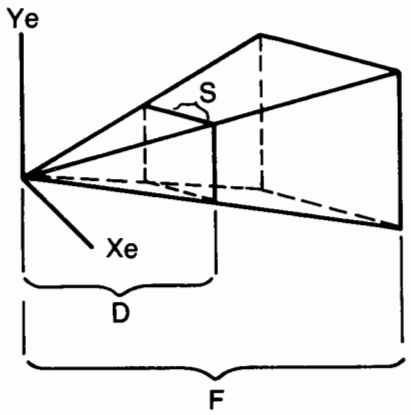


Figure 3.36 [3]

These values are in eye coordinates and must be computed for every viewpoint. In this pack S, D and F are defined as follows:

S = D/4
D = Zmin − (Zmax−Zmin)/10
F = Zmax + (Zmax−Zmin)/10

Zmax and Zmin (also in eye coordinates) are the maximum and minimum Z values of the object. S, D and F are used in the transformation matrix P.

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & S/(D(1-D/F)) & S/D \\ 0 & 0 & -S/(1-D/F) & 0 \end{bmatrix}$$

Figure 3.37 [3]

The vertices of an object are then multiplied by the concatenation of V and P. The results are coordinate quadruplets, (X, Y, Z, W), in the clipping coordinate system. The W value is called the homogeneous coordinate which is used by the clipping algorithm, and then later in the perspective calculation [3].

Since objects are represented as a collection of surfaces in this pack, it is necessary to use a surface clipping algorithm rather than a line clipper. The clipping algorithm used in this pack is called a re-entrant surface clipper because it is recursive and clips surfaces [4]. There are two parts to the algorithm. These are represented in flow diagrams below.

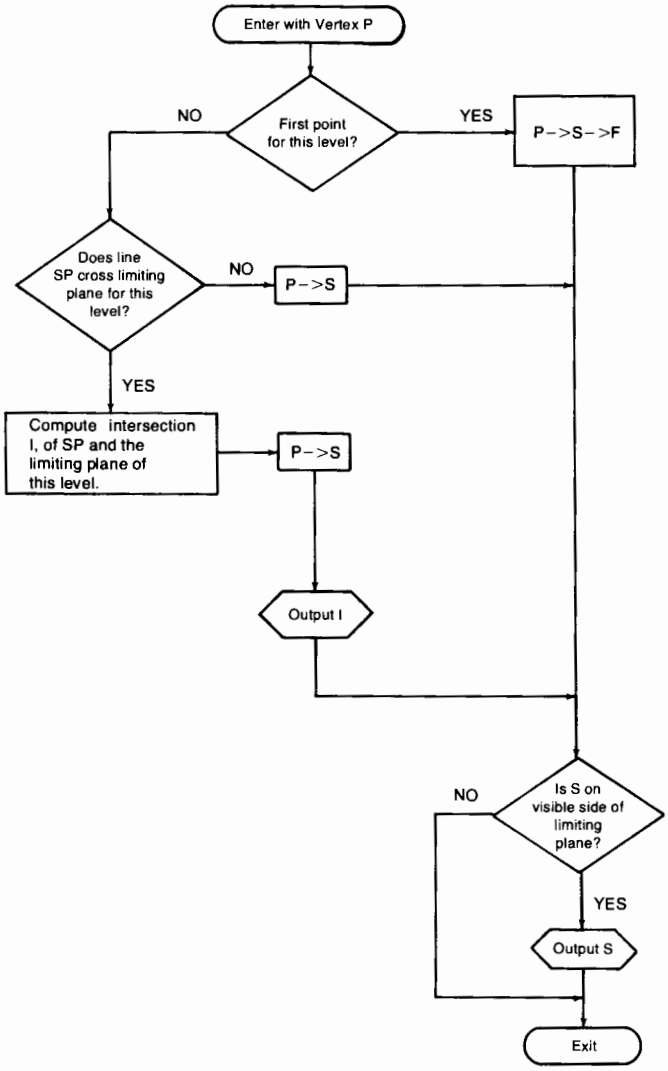


Figure 3.38a Part One of Clipping Algorithm [4]

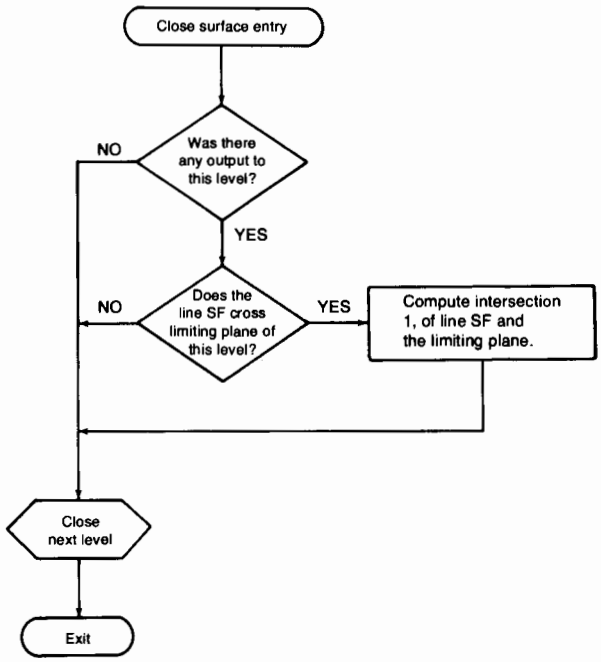


Figure 3.38b Part Two of Clipping Algorithm [4]

Consecutive vertices of a surface are passed to Part One, one at a time. As the program progresses, Part One calls itself 6 times. At each level of recursion a different clipping plane is considered. The order of the limiting planes is: top, bottom, left, right, hither, yon. Note that each level may call Part One twice, at Output I and Output S. However, if the program is at the sixth level, I and S are the vectors of vertices of the clipped surface.

Each time a level is called, the input vertex is saved in S. Once a level has received two vertices, it decides whether to clip the line joining them. If the line is on the visible side of the limiting plane, the first endpoint is sent to the next level. If the line is not visible, the program returns to the previous level. A non-trivial case arises when the line crosses the clipping plane. To find the intersection, the visible fraction of the line, call it $\propto$, is calculated and added to the visible endpoint. In the formula below, S and P are the visible and clipped endpoint vertices [4].

$$I = \propto *(P-S)+S$$

The homogeneous coordinate of a vertex, W, is used in the clipping decision at each level. The bottom and top levels clip a line whose endpoints are outside the range $-W>= Y >=W$. The visible limits of the left and right clipping planes are $-W>=X >=W$. The hither and yon limiting values range from 0 to W. This step is essential to prevent an object from "wrapping around" on the screen [4].

After all vertices of a surface have been passed to Part One, the second part of the clipping program is called to flush out any remaining points. Part Two also calls itself at "close next level" 6 times, once for each clipping plane. If, during Part One, no vertices were ever sent to a level, or if the line between the first and last vertex sent to a level (F and S) does not cross its limiting plane, the program returns to the previous level. Otherwise, the intersection of SF and the current limiting plane is calculated and output to the next level in Part One. If Part Two is at level 6, however, the output vector, I, is a vertex of the clipped surface. The vertices of the new surface are stored in a temporary array Q(*) during the clipping process.

Perspective

In this pack, a single point perspective projection is used to convey depth information by making distant objects smaller than those closer. In preparation for this step, it is necessary to scale and translate the X and Y coordinates of each vertex into the coordinate system in which they will be displayed. The values of the transformation matrix S depend upon whether a wire-frame or hidden surface display is desired. The parameters Vsx and Vsy are equal to one half of the range of the screen display coordinates in the X and Y axes. (Vcx, Vcy) is the center point of the screen coordinates [3]. All vertices are multiplied by the correct matrix, S, before the perspective is calculated, in order to put an object into the screen coordinate system.

$$S = \begin{bmatrix} Vsx & 0 & 0 & 0 \\ 0 & Vsy & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Vcx & Vcy & 0 & 1 \end{bmatrix}$$

Wire-frame: Vsx = 108.13, Vsy = 100
Vcx = 0 , Vcy = 0

Hidden-surface: Vsx = 245.50, Vsy = 227
Vcx = 313.50, Vcy = 227

Figure 3.39 [3]

The perspective calculation itself is quite simple. The X, Y, and Z coordinate of each vertex is divided by its homogeneous coordinate, W.

Output

This pack uses two methods of output, wire-frame and hidden surface removal. The clipping and perspective algorithms previously described allow output using either.

Wire-frame

If your application only requires wire-frame output, the clipping and perspective algorithms can be greatly simplified. See PRINCIPLES OF INTERACTIVE COMPUTER GRAPHICS, edition 2, by Newman and Sproull for details.

After perspective division, the object may be drawn as is, using the coordinate pairs (Xs,Ys) for a wire-frame display. For hidden surfaces to be removed, however, further calculation is needed. The following is a summary of the viewing process up to this point.

[X Y Z W] = [Xw Yw Zw 1]*V*P (clip)*S, where Xw is the X coordinate in the world coordinate system.

Xs = X/W, Ys = Y/W, and Zs = Z/W

Hidden-Surface

There are two subprograms involved in hidden-surface removal. "Hidden_surface" is first used to transform data into the hidden-surface data base as described in Chapter Four, The Data Base. Then the utility "Scan" is called to perform the actual calculations using a modified scan technique.

During the program the Edge(*) is used to hold three linked lists. The first is set in the subprogram "Hidden_surface" and sorts the edges by their initial Y values. The head of the list at any time is held in the variable Elist. After processing begins on an edge, it may be sorted by its X value in either the list for the current line or for the next scan line. It is sorted in the current scan line's list until processed, at which time the edge's X value is updated and resorted into the list for the next scan line. The head of the new list is stored in the variable Xsort.

Poly(*) is used to hold two linked lists. If active, a polygon may either be linked by its Z value at the current sample point, or by its projected Z value for the next sample point. The head of the next sample point list is kept in the variable, Inpoly.

Beginning at the top of the display, each raster scan line is considered one at a time. The program then scans across the line to each occurrence of an edge intersection, called a sample point. At each sample point intersection, all active polygons are sorted by their Z values. This "intersection" is the point where a polygon would be pierced by a line parallel to the Z axis (see the diagram 3.40). The polygon with the smallest Z value (or the head of the Inpoly list) is visible. If an edge belongs to the polygon that is in front of any others at this sample point, the 9845B program outputs a dot, unless the edge is a horizontal edge.

The 9845B program treats horizontal edges as a special case, since all points of such an edge intersect a scan line. The intersecting sample points are then considered to be the endpoints of the edge. This means that after processing the left endpoint, the right one is immediately resorted back into the Xsort list of the current scan line. A flag is then set to tell the program whether it is considering the left or right endpoint of the edge.

Before going on to the next sample point, the Inpoly list is updated to hold only the polygons that are still active at the next sample point. It is at this time that horizontal edges of the top polygon are displayed for the 9845B or surfaces are shaded for the 9845C. When all sample points of a scan line have been processed, the edges that will be active on the next scan line are updated. First their next Z values are computed, and then they are resorted into the Xsort list by their next X values. The program is finished when all scan lines of the display have been considered [1].

A surface can be in any one of four stages, depending on which sample point is being considered. These stages are out, just active, active, and just inactive. If a particular sample point does not intersect a polygon, that polygon is considered to be "out". If it intersects the left edge of a polygon, that polygon attains the mode "just active". If it lies between a polygon's left and right edge, the polygon is considered "active". If it encounters the right edge, the polygon becomes "just inactive" [1]. Note that all surfaces should have the mode of "out" at the end of a scan line. Below is an example of a typical surface, S, and its modes along one scan line.

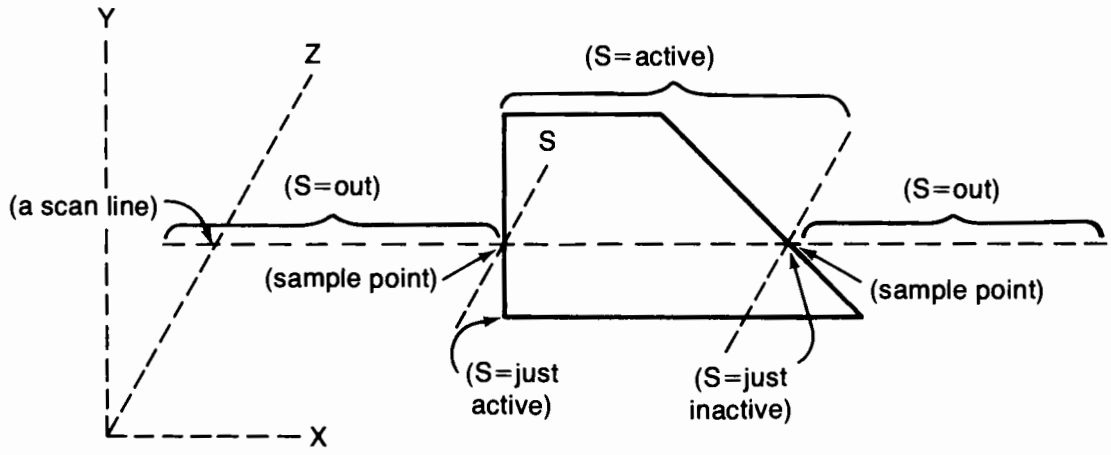


Figure 3.40

Since the algorithm uses calculated slopes to predict the X and Z values of a surface at a particular point, there is some inherent round off error. This does not usually present a problem, but do not be surprised if occasionally the wrong surface appears on a scan line.

The algorithm requires that all surfaces be planar, closed, and clipped in the X, Y and Z directions, and that all (Xs,Ys) coordinate pairs be scaled to the ranges 68<=Xs<=559, and 0<=Ys<=454. These values correspond to the dot units within the clipped plotting area on the CRT.

NOTE

This algorithm does not handle surfaces that intersect other than at edges.

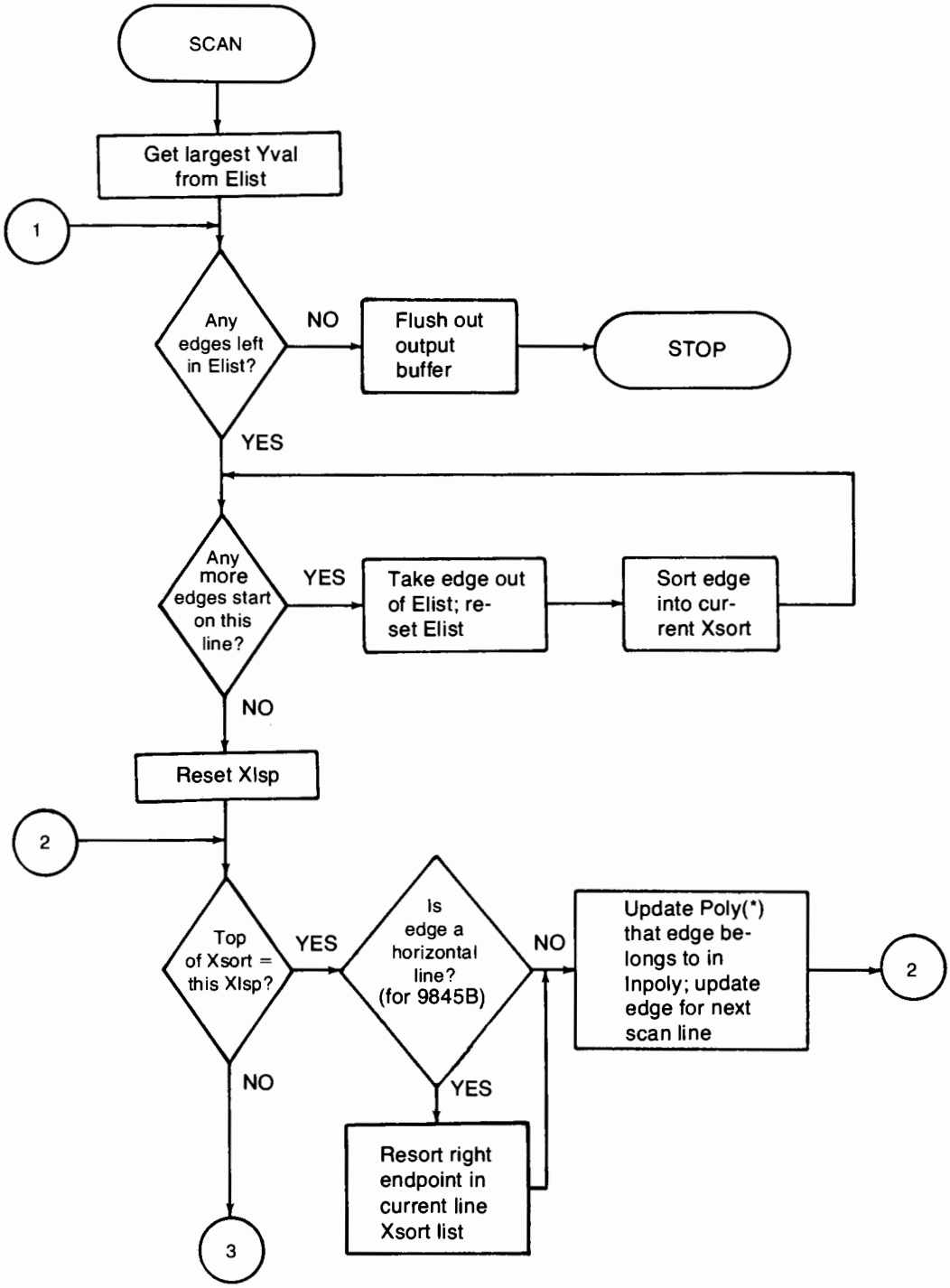


Figure 3.41

Flowchart for Scan Algorithm

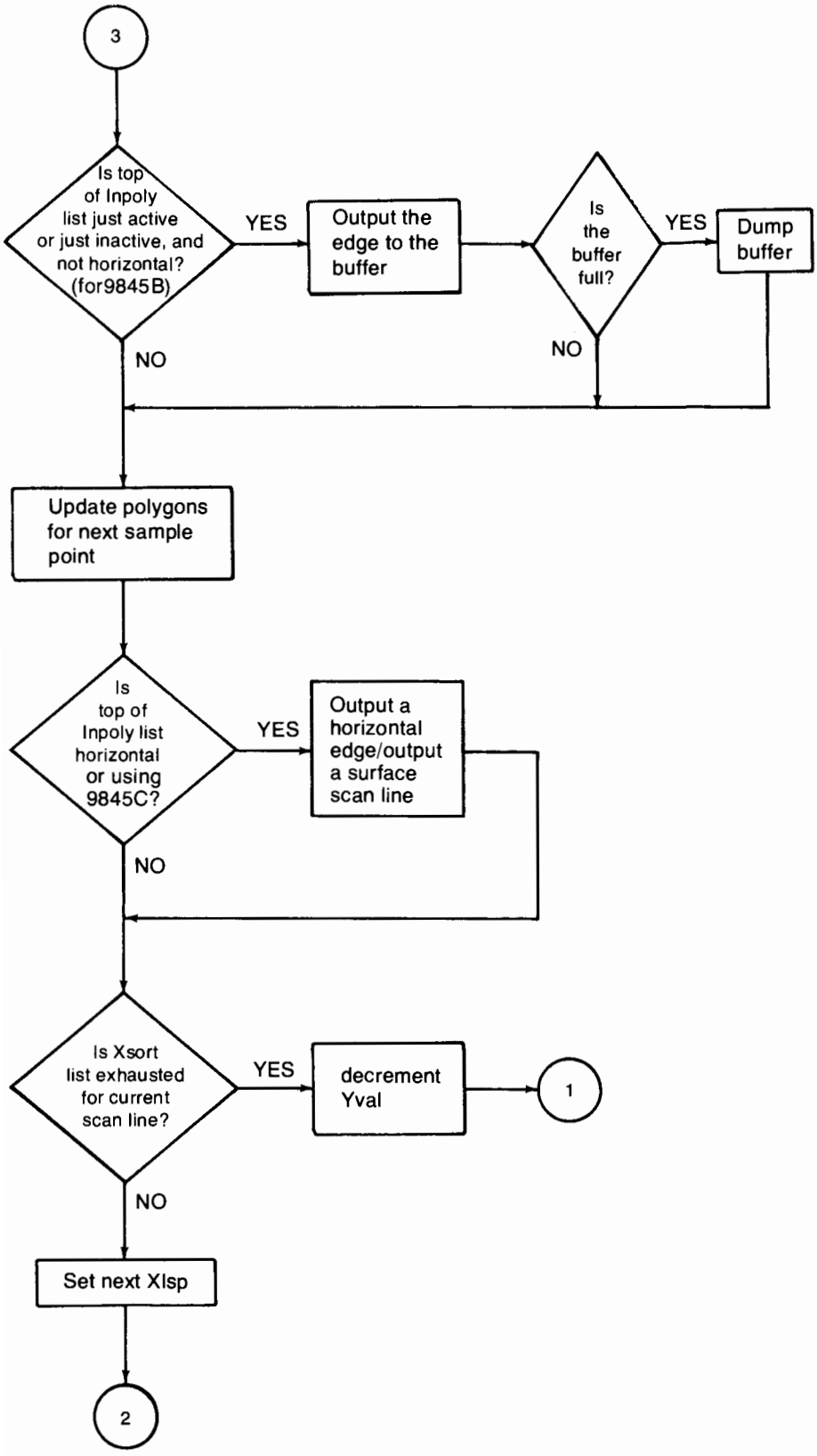


Figure 3.41 cont.

Chapter 4
The Data Base

Overview

As mentioned in Chapter 2, this pack uses the "linked vertices" method for its main data base. The original coordinates of an object are stored in this data base along with all the edge type and edge connecting information. Two temporary data bases are used in the digitizer input and display routines. Their purposes are to get the data into the main data base format and to hold transformed data ready for display.

Input

Digitizer Data Base

The digitizer input routine stores points from the three orthogonal views in a temporary data base until input is complete. This data structure is used to keep track of all vertices and edge connecting information for each view. Vertex2d(*) contains all the vertices for each view, along with flags used to convey information needed to convert the data into a form compatible to the main data base. Edge2d(*) consists of pointers to those vertices which are connected in each plane. As the main data base is built, other columns of Edge2d(*) are used as a "scratch pad" to contain flags and pointers necessary to the conversion algorithm. Surf2d(*) contains linked lists of pointers to the edges making up an object's surfaces. Object2d(*) contains linked lists of pointers into the Surf2d(*) for all three views. The diagrams of the data base drawn below are three dimensional, one dimension per view.

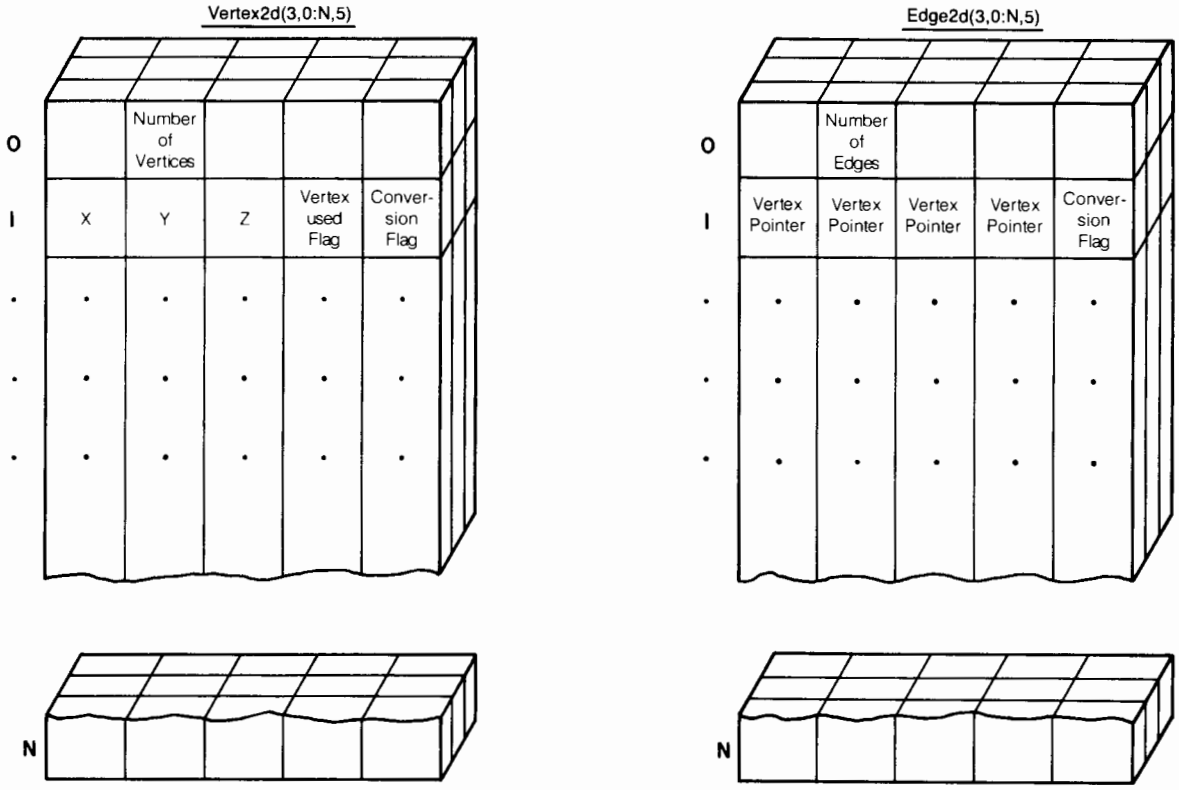


Figure 4.1

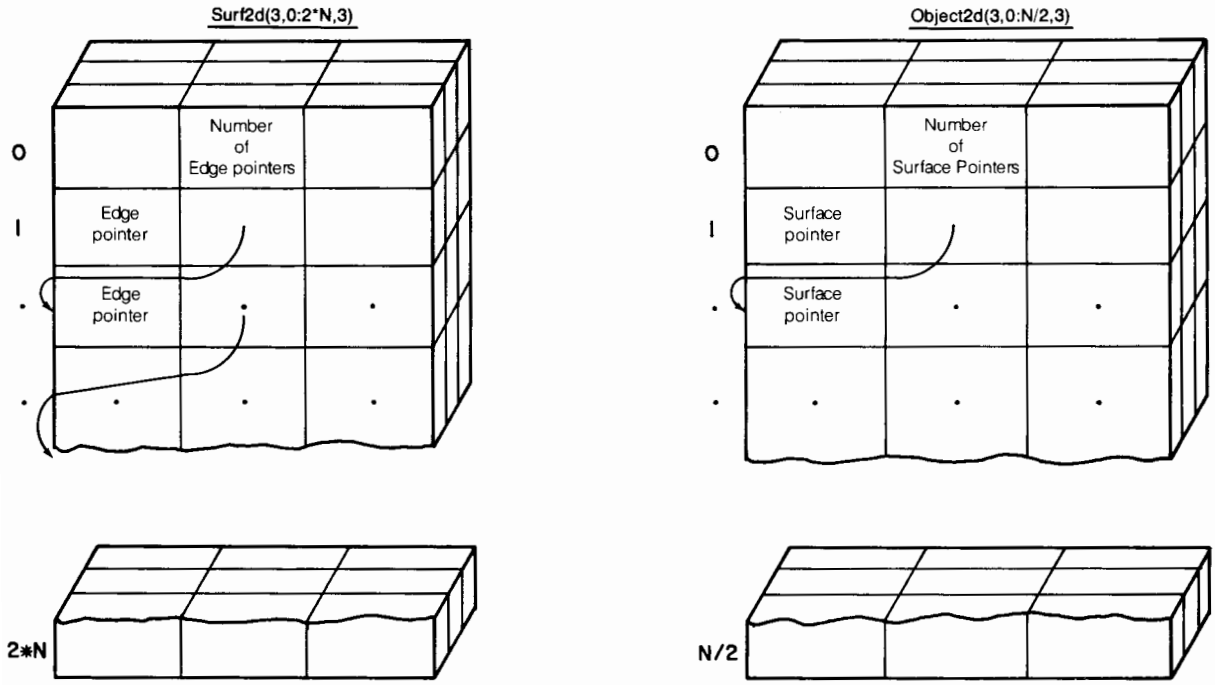


Figure 4.1 cont.

Manual and Post-digitized Data Base

If vertices of an object are entered manually, they are transformed immediately into the standard range and stored in the main data base. The standard range is a cube with values: $-100<=X<=100$, $-100<=Y<=100$, $-100<=Z<=100$. If the object is entered using the digitizer routines, the user must request that the data be converted into the main 3-D data base. Since vertices entered into the temporary digitizer data base are already in the standard range, no conversion is needed.

The main data base contains all the vertices of an object, along with all edge and surface linking information. It's linked structure allows for easy insertions and deletions of input information. The diagrams below display the data base pictorially. (Note an "*" indicates used with 09845-10080.)

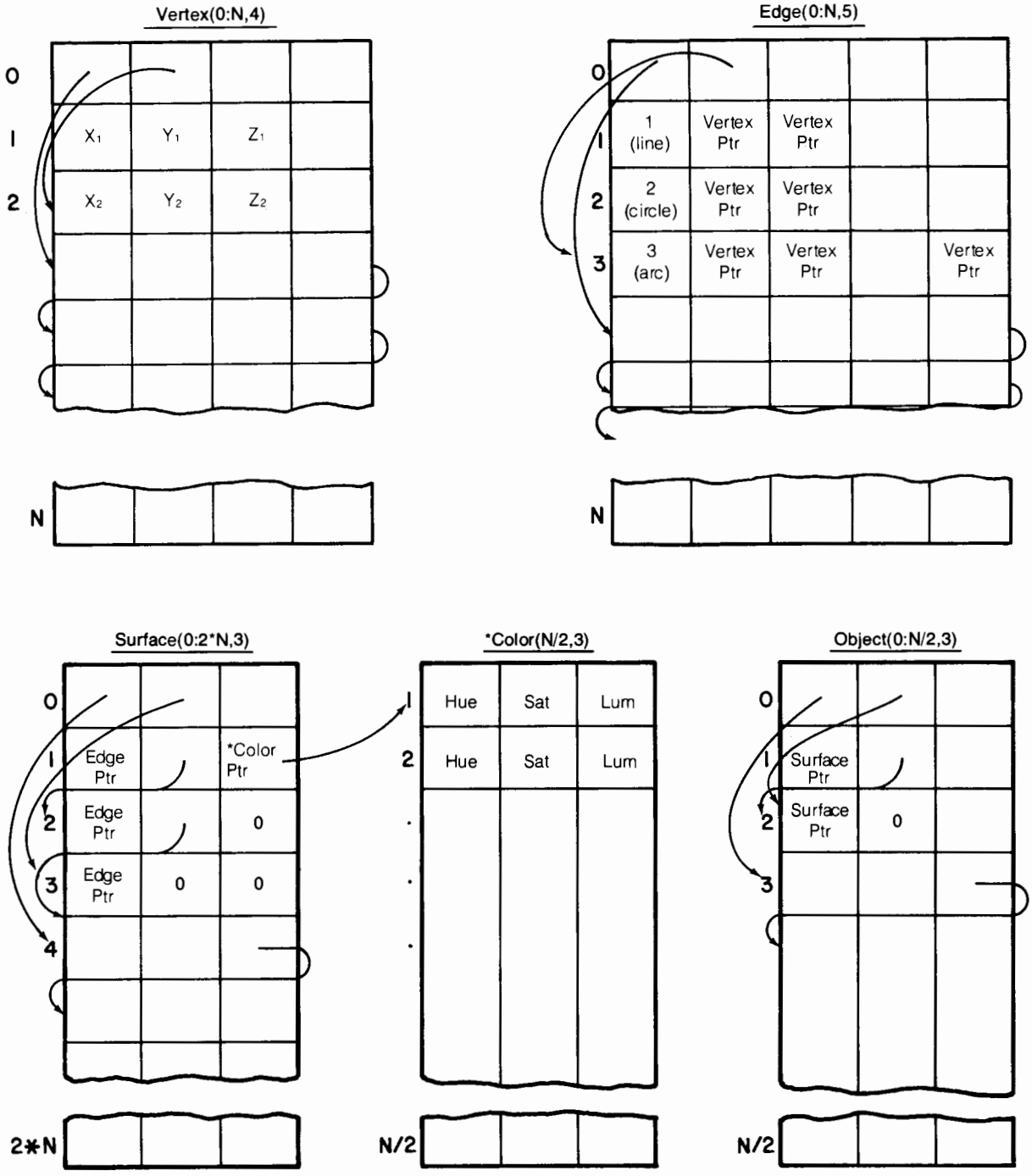


Figure 4.2

Vertex(*) contains all distinct (X,Y,Z) triplets for an object. The fourth column is set to 1 in order to do matrix multiplication for transformations. As discussed in Chapter 3, this 4th column is necessary in order to generate the homogeneous coordinates for clipping and perspective. Row zero is used to hold special information about Vertex(*). Vertex(0,1) and Vertex(0,2) contain pointers to the next free row and the last used row, respectively.

The first column of Edge(*) contains the edge type (1 = line, 2 = circle, 3 = arc). The contents of the next four columns depends on the edge type. If the edge is a line, the 2nd and 3rd columns contain pointers to the endpoints of the line. If the edge type is a circle, the 2nd, 3rd, and 5th columns contains pointers to the three vertices defining the circle. For an arc edge type, columns 2 and 5 are pointers to the endpoints of the arc and column 3 contains a pointer to a point between the two endpoints. When a row is not being used to hold edge information, column 5 is used as a pointer to the next free row. As in the Vertex(*), row zero of Edge(*) contains other information. Edge(0,1) and Edge(0,2) are pointers to the next free row and the last used row.

The fourth column of Edge(*) has purposely been left empty as a convenient way to add non-spatial data into the data base. It can be used directly to contain a line type or a pen number. Also, if you have another matrix of information associated with each edge such as a label, the fourth column could contain a pointer into a string matrix.

Surface(*) contains a group of linked lists of edges which comprise the surfaces of an object. Column 1 is a pointer to the next edge of the surface. Column 2 contains a pointer to the row in Surface(*) containing the next edge pointer. If a row is not being used, column 3 holds a pointer to the next free row. When using 09845-10080, the 3rd column of the head of a surface list contains a pointer into the Color(*). As in the other matrices, Surface(0,1) and Surface(0,2) contain pointers to the next free row and the last used one.

Object(*) is a linked list of pointers to surfaces that comprise an object. The head of the list is pointed to by the variable, Pic. The first column of a used row contains a pointer into Surface(*) where the pointer to the first edge of the surface is found. Column 2 is the pointer to the row in Object(*) where the next surface pointer is found. When the row is not being used, column 3 contains the next free pointer. Object(0,1) and Object(0,2) contain pointers to the next free row and the last used one.

Color(*) is used to hold the color parameters for each surface. These include Hue, Saturation and Luminosity. (09845-10080)

Manipulation

All transformations, whether viewing, translation, scaling or rotation, change the status of an object's coordinates. As discussed in Chapter 3, these transformations may be concatenated into one 4 by 4 matrix. The original points of an object are then multiplied by the "status" matrix to get its transformed points. In the pack, Status(*) is used to hold the current transformation information for an object and Transform(*) contains its transformed points.

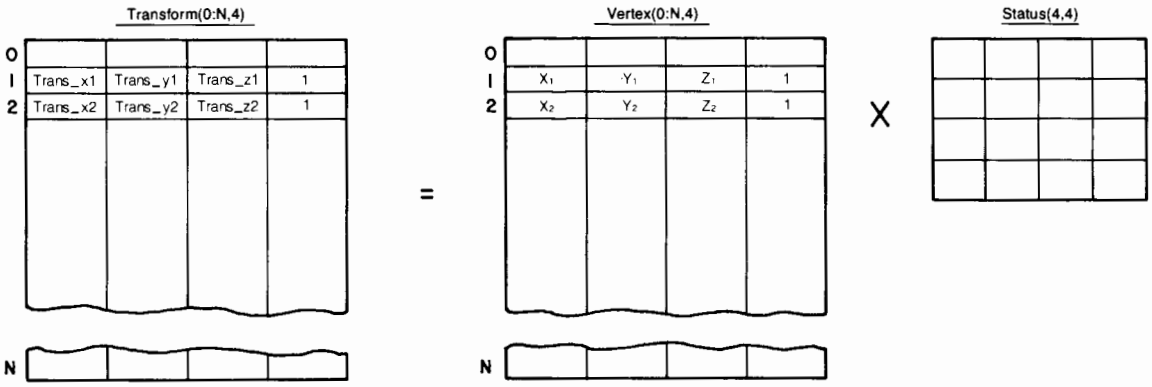


Figure 4.3

Note that Edge(*), Surface(*), and Object(*) can still be used to access the transformed points of an object.

Viewing

The first temporary data base in the display process is used in the clipping routine. All transformed points and edge connecting information from the main data base are put into the following format:

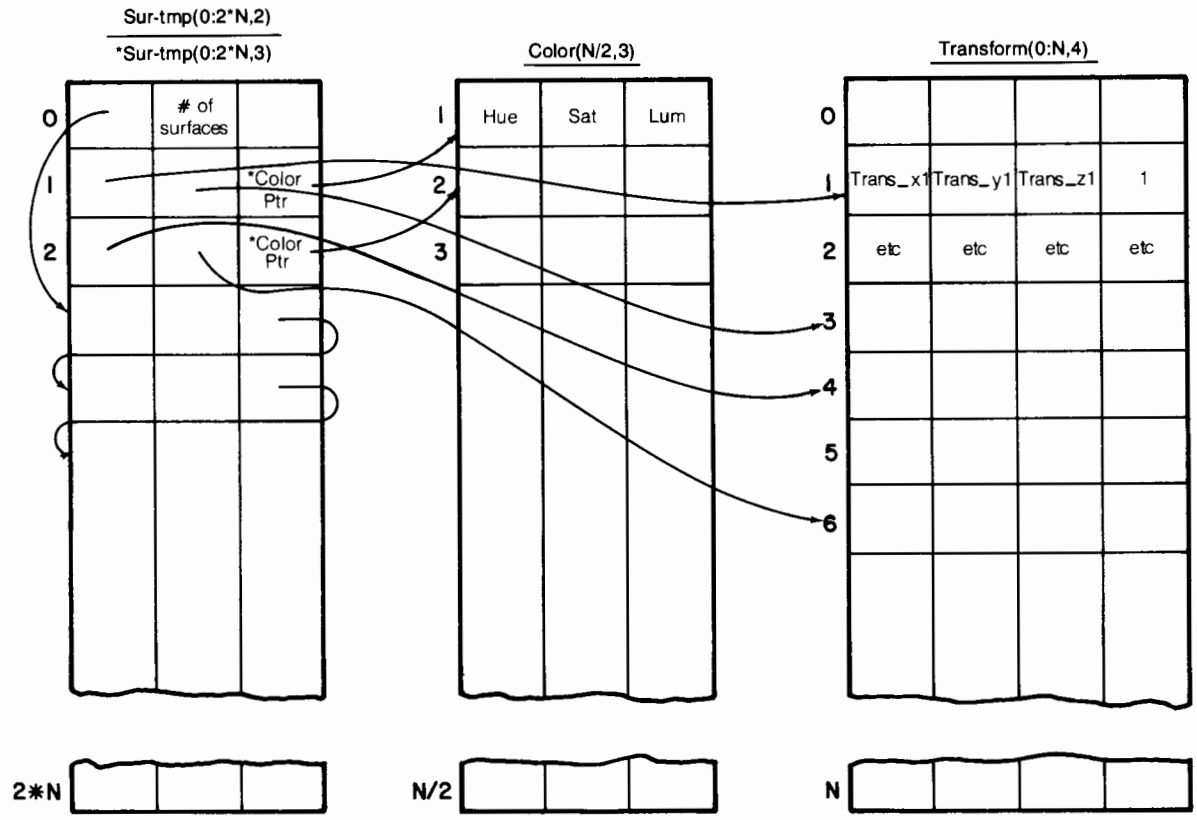


Figure 4.4

Points are stored in Transform(*) as they occur in each surface. Note that circles and arcs are broken up into line segments at this time. Sur_tmp(*) contains pointers to the beginning and ending rows of points in Transform(*) that comprise each surface. The third column of Sur_tmp(*) contains the link to the next used row in Sur_tmp(*). Sur_tmp(0,2) contains the number of surfaces.

The purpose of this data structure, Sur_tmp(*), is to put the information about an object into a form necessary for the surface clipping algorithm. During the clipping process, it usually becomes necessary to add or delete vertices from Transform(*), as not all vertices of a surface may be visible. In this case, Sur_tmp(*) is adjusted accordingly. When using 09845-10080, the color information for each surface must also be preserved.

After clipping, if hidden-surface removal is not desired, the object may be drawn in wire-frame from this temporary data base. If hidden-surface removal is desired, another temporary data base is required.

Output

This last temporary data base is quite different from any other previously discussed. It must contain the endpoints of all line segments and information indicating which surface the line belongs to, as well as the slopes of each line and surface. The two matrices of this data base are Poly(*) and Edge(*). Due to the length of these arrays, each array subscript is described is detailed below. Those marked with "**" are initialized by the Scan routine, while all others must be set before entering the Scan portion.

Edge(N,0:10)

Edge(I,0) – pointer to an edge stored in Edge(*) with the next less Y initial value: last pointer in list = 0; (changed in program to reflect position in the Xsort list).

Edge(I,1) – pointer to one polygon in Poly(*) that this edge belongs to

Edge(I,2) – pointer to 2nd polygon in Poly(*) that this edge belongs to; note that one edge may refer to two polygons

Edge(I,3) – X coordinate of the top endpoint

Edge(I,4) – Y coordinate of the top endpoint

Edge(I,5) – X coordinate of the bottom endpoint

Edge(I,6) – Y coordinate of the bottom endpoint

Edge(I,7) – X coordinate of the top endpoint; (changed in the program to be the X coordinate of the current point on the edge under consideration)

Edge(I,8) – Dxdy, which is (Delta X) / (Delta Y); for horizontal lines, set this value to 99999.

Edge(I,9) – Z coordinate of the top endpoint; (changed in the program to be the Z coordinate of the current point on the edge under consideration)

Edge(I,10) – Dzdy, which is (Delta Z) / (Delta Y); for horizontal lines, set this value to 99999

Poly(2*N,0:5)

**Poly(I,0) – polygon's position in the Inpoly list

Poly(I,1) – initialize to −1; (changed in the program to be this polygon's status flag)

**Poly(I,2) – used in the program as Z coordinate of this polygon at a point in space

Poly(I,3) – Dz/Dx, which is the range in Z with respect to X of a polygon. It is computed by the following formula where (X1,Y1,Z1), (X2,Y2,Z2) and (X3,Y3,Z3) are three different points on the polygon.

$$Dz/Dx = \frac{\begin{vmatrix} Y1 & Z1 \\ Y2 & Z2 \\ Y3 & Z3 \end{vmatrix}}{\begin{vmatrix} X1 & Y1 \\ X2 & Y2 \\ X3 & Y3 \end{vmatrix}}$$

Poly(I,4) – used in the program as a pointer to the current edge of this polygon under consideration

The following array extensions are for 09845-10080

Poly(I,5) – the hue value of this polygon

Poly(I,6) – the saturation value of this polygon

Poly(I,7) – the luminosity value of this polygon

The subprogram Hidden_surface, is responsible for translating the clipping data base into this last data structure. The subprogram Scan then processes the object in this form, one line at a time, and outputs the object with hidden surfaces removed.

Chapter 5
Input Utilities

The Input Utilities are located on the "Input" cartridge. This cartridge includes all the subprograms needed for entering objects manually or with the digitizer.

Manual Entry Subprograms

The following subprograms are used when doing manual data entry. They are One_obj_entry, Color_entry, Initialize, List, Relist, Surface_setup, Check_arrays, Surface_entry, Enter_line, Enter_circle, Enter_arc, Save_line, Save_circle, Save_arc, Edge_ver_entry, Change_point, Database_delete, One_obj_store, Store_color, and Plane_equation.

A description of each of these along with its file name and the definitions for the local variables follows.

One_obj_entry

One_obj_entry(File$,INTEGER Edge(*),N, Object,Object(*), Pic, Surface(*), SHORT Status(*), Vertex(*))

One_obj_entry enters one previously stored 3-D object into the data base.

If One_obj_entry is used while the program is in GRAPHICS mode on the 9845B, the subprogram Ginput will be needed. If Ginput is used, all LINPUTs must be changed to calls to Ginput.

Filename: Obj_en

Local Variable Definitions

Flag	Equals 1 if a file does not exist on the mass storage designated
Type	Designates type of file; 3-D file =3

Color_entry

Color_entry(INTEGER Object(*),Pic,Surface(*),SHORT Color(*))

This subprogram inputs the color parameters for an object and links them to the data base.

Filename: Cl_enC

Local Variable Definitions

File_color$[12]	Name of file containing color parameters
Color	Index into Color(*)
Object	Index into Object(*)
Return	Equals 1 if file is found
Surface	Index into Surface(*)

Initialize

Initialize(INTEGER Edge(*),N,Object(*), Surface(*), SHORT Vertex(*))

Initialize provides for the initialization of the 3-D data base.

Filename: Init

Local Variable Definitions

I	Loop index

List

List(File$,INTEGER List_flag)

This subprogram provides for the listing of the X, Y, and Z coordinates of a 3-D object.

Filename: List

Local Variable Definitions

Printer	1 = CRT, 2= thermal line printer, 3 = no listing

Relist(09845-10060)

Relist(INTEGER Edge(*),Object(*),Pic,Surface(*), Surface_count, SHORT Offset(*), Range(*), Transform(*))

This subprogram is used to list the X,Y, and Z coordinates of an object.

Filename: RlistB

Local Variable Definitions

Next_edge	Pointer to next edge to process
Next_surface	Pointer to next surface to process
Object	Pointer to first Object
X1	Calculated X-coordinate of 1st vertex
X2	Calculated X-coordinate of 2nd vertex
X3	Calculated X-coordinate of 3rd vertex (if circle or arc)
Y1	Calculated Y-coordinate of 1st vertex

Y2	Calculated Y-coordinate of 2nd vertex
Y3	Calculated Y-coordinate of 3rd vertex (if circle or arc)
Z1	Calculated Z-coordinate of 1st vertex
Z2	Calculated Z-coordinate of 2nd vertex
Z3	Calculated Z-coordinate of 3rd vertex (if circle or arc)

Relist(09845-10080)

Relist(INTEGER Edge(*),Flag,Object(*),Pic,Surface(*),Surface_count,SHORT Color(*), Offset(*),Range(*),Transform(*))

This subprogram is used to list the X,Y, and Z coordinates of an object. If the variable Flag is set, it will pause after displaying the coordinates of a surface and allow you to enter color information.

Filename: RlistC

Local Variable Definitions

Next_edge	Pointer to next edge to process
Next_surface	Pointer to next surface to process
Object	Pointer to first Object
X1	Calculated X-coordinate of 1st vertex
X2	Calculated X-coordinate of 2nd vertex
X3	Calculated X-coordinate of 3rd vertex (if circle or arc)
Y1	Calculated Y-coordinate of 1st vertex
Y2	Calculated Y-coordinate of 2nd vertex
Y3	Calculated Y-coordinate of 3rd vertex (if circle or arc)
Z1	Calculated Z-coordinate of 1st vertex
Z2	Calculated Z-coordinate of 2nd vertex
Z3	Calculated Z-coordinate of 3rd vertex (if circle or arc)

Surface_setup

Surface_setup(INTEGER Edge_count,Object,Object(*),Pic,Surface,Surface(*), Surface_count,Surface_flag,REAL A(*),B(*),C(*),D(*))

This subprogram handles the overhead for addition of a new surface when using the manual entry method.

Filename: Surset

Local Variable Definitions

I	Loop index

Check_arrays

Check_arrays(INTEGER Edge(*), Edge_flag, Vertex_flag, SHORT Vertex(*))

This subprogram checks to see if Edge(*) or Vertex(*) are full before an edge is added to the data base.

Filename: Check

Surface_entry

Surface_entry(INTEGER Closed_flag, Edge_count, Edge_index, Surface, Surface(*), Surface_flag, SHORT Firstpt(*), Lastpt(*))

Surface_entry stores the pointer to an edge into the Surface array.

Filename: Surent

Enter_line

Enter_line (INTEGER Edge_count,Line,Surf_flag,SHORT Firstpt(*),Lastpt(*),Offset(*), Range(*),Real_pts(*),Tran_pts(*),REAL A,A(*),B,B(*),C,C(*),D,D(*))

This subprogram provides for the entry of line segments. The X, Y, and Z coordinates of the endpoints are entered using this routine.

Plane_equation is called from Enter_line.

Filename: Line

Enter_circle

Enter_circle(SHORT Offset(*),Range(*), Real_pts (*), Tran_pts (*))

Enter_circle allows entry of a circle to the data base by entering the X, Y, and Z coordinates of three points on the circle.

Filename: Circ

Local Variable Definitions

I	Loop index
A(3,3)	Matrix used to find the plane parameters of a surface
B(3,3)	Matrix used to find the plane parameters of a surface
C(3,3)	Matrix used to find the plane parameters of a surface

Enter_arc

Enter_arc (INTEGER Edge_count, Surface_flag, SHORT Firstpt (*), Lastpt (*), Offset (*), Range (*), Real_pts (*), Trans_pts (*), REAL A, A (*), B, B (*), C, C (*), D, D (*))

Enter_arc provides the utility of entering X,Y and Z coordinates for three points of an arc. These points are the two endpoints and a point on the arc.

Plane_equation is called from Enter_arc.

Filename: Arc

Local Variable Definitions

E(3,3)	Matrix used to determine if three points in space are collinear
F(3,3)	Matrix used to determine if three points in space are collinear
G(3,3)	Matrix used to determine if three points in space are collinear

Save_line

Save_line(INTEGER Edge_type,SHORT Real_pts(*),Status_inv(*))

Save_line prepares line to be be added to data base.

Filename: Sline

Local Variable Definitions

Temp_pts(3,4)	Used to temporarily contain the Real_pts(*)

Save_circle

Save_circle(INTEGER Edge_type,SHORT Real_pts(*),Status_inv(*))

This subprogram prepares the points determining a circle to be added to the data base.

Filename: Scir

Local Variable Definitions

Temp_pts(3,4)	Used to temporarily contain the Real_pts(*)

Save_arc

Save_arc(INTEGER Edge_type,SHORT Real_pts(*),Status_inv(*))

This subprogram prepares for an arc to be added to the data base.

Filename: Sarc

Local Variable Definitions

Temp_pts(3,4)	Used to temporarily contain the Real_pts(*)

Edge_ver_entry

Edge_ver_entry (INTEGER Edge(*),Edge_count,Edge_index,Edge_type,SHORT Real_pts(*),Vertex(*))

This subprogram provides a method of entering edge information into the data base arrays Edge(*) and Vertex(*).

Filename: Edgver

Local Variable Definitions

Flag1	Set to 1 if 1st point to be stored is already in Vertex(*)
Flag2	Set to 1 if 2nd point to be stored is already in Vertex(*)
Flag3	Set to 1 if 3rd point (for arc or circle) is already in Vertex(*)
I	Loop index
P1_index	Points to next free space in Vertex(*)
P2_index	Points to next free space in Vertex(*)
P3_index	Points to next free space in Vertex(*)

Change_point

Change_point (INTEGER Edge(*), Object (*), Pic, Surface (*), Surface_count, SHORT Offset (*), Range (*), Status_inv (*), Vertex (*))

This subprogram changes a single vertex of a surface when entering data from manual mode. Change_point does no testing to see if the vertex you enter is already in the surface description. It is only meant as a quick error correction routine so that you will not have to reenter an entire surface due to a mistake in entering a single vertex.

The only subprogram called from Change_point is Plane_equation.

Filename: Chngpt

Local Variable Definitions

Edge	Pointer to edge being examined for point being changed
Edge1	Pointer to first edge containing point to be changed
Edge2	Pointer to second edge containing point to be changed
I	Loop index
New_vertex	Pointer to next empty location in Vertex(*) to be used if a new vertex must be added
Num_edges	The number of edges containing the point to be changed
Number_of_surf	The number of the surface containing the point to be deleted
Object	Pointer to object
Surface	Pointer to surface
Vertex	Index to point that is to be changed
Vertex1	Pointer to vertex

Vertex2	Pointer to vertex
Vertex3	Pointer to vertex
Real_pts(2,4)	Tran_pts(*) multiplied by current Status(*)
Tran_pts(2,4)	Contains X, Y, and Z value of point to be changed and X, Y, and Z value of new point
X	X-coordinate of new point after it has been multiplied by Status(*)
Y	Y-coordinate of new point after it has been multiplied by Status(*)
Z	Z-coordinate of new point after it has been multiplied by Status(*)
A	Determinant of A(*)
A(3,3)	Matrix used to find plane parameters of a surface
B	Determinant of B(*)
B(3,3)	Matrix used to find plane parameters of a surface
C	Determinant of C(*)
C(3,3)	Matrix used to find plane parameters of a surface
D	Determinant of D(*)
D(3,3)	Matrix used to find plane parameters of a surface

Database_delete

Database_delete (INTEGER Edge (*), Object, Object (*), Pic, Surface (*), Surface_count, SHORT Vertex (*))

This subprogram deletes a surface from the data base.

Filename: Surdel

Local Variable Definitions

Edge	Pointer to current edge of surface that is to be deleted
Edge_type	1 = line, 2 = circle, 3 = arc
Flag1	Set to one if vertex 1 is used by another edge
Flag2	Set to one if vertex 2 is used by another edge
Flag3	For arcs - 1 if vertex 3 used by another edge
I	Loop index
Number_of_surf	The number of surface to be deleted
Obj	Pointer to object to be deleted
Surface	Pointer to surface to be deleted
V1_ptr	Pointer to 1st vertex of edge to be deleted
V2_ptr	Pointer to 2nd vertex of edge to be deleted
V3_ptr	Pointer to 3rd vertex if Edge_type = 2 or 3

One_obj_store

One_obj_store (INTEGER Edge (*), N, Object (*), Pic,Surface (*), SHORT Status (*), Vertex (*))

This subprogram stores a 3-D object on the desired mass storage medium. If you are using pack 09845-10060 and wish to store data while in GRAPHICS mode, replace all LINPUT or INPUT statements with calls to Ginput.

The only subprogram called from One_obj_store is Ginput when using pack 09845-10060 and this subprogram is only needed in the case mentioned above.

Filename: Obj_st

Local Variable Definitions

Answer$[3]	String for YES/NO responses
File$[15]	String for filename designation
Flag	Equals 1 if no file already exists on the mass storage designated

Store_color (09845-10080)

Store_color(INTEGER Object(*), SHORT Color(*))

This subprogram stores an object's color information onto a data file.

Filename: Cl_stC

Local Variable Definitions

Ans$[17]	String used for YES/NO response
File$[12]	String used for filename designation
Return	Equals 1 if no file already exists on the designated mass storage device

Plane_equation

Plane_equation(REAL A,A(*),B,B(*),C,C(*),D,D(*))

This subprogram provides the values A, B, C,and D for the plane equation Ax+By+Cz-D=0.

Filename: Plnequ

Digitizer Entry Subprograms

The following ten subprograms are used only for digitizer entry. These allow you to enter a three-dimensional figure by digitizing in two-dimensional information from three orthogonal views of a mechanical drawing. Refer to Chapter 3 for a more detailed description of the three views needed. See the digitizer input demo, Demo 3, in Chapter Ten to see how the subprograms interact. These subprograms are Input, Initviewpln, Constviewpln, Display, Draw2d, Modvp, Recvpcoords, Convert3d, Indbad, and Findsurf. A description of each along with their file names and local variable definitions follow.

Input

Input (INTEGER Digi, Mneps, Ne, Ne2, No, No2, Ns, Ns2, Nv, Nv2, Redundfctr)

Input is the main subprogram for digitizer input. It provides for the digitizer function key monitoring.

The subprograms called from Input are Convert3d, Display, Draw2d, Dump Initviewpln, Modvp, One_obj_store, and Plot.

Filename: DginpB (09845-10060),DginpC (09845-10080)

Local Variable Definitions

Answer$[3]	Contains YES/NO response
Dsp$[15]	Message to be displayed in digitizer display
Filnam$[10]	File name used when getting or saving 2D information
Plotter$[20]	Plotter identifier
S$[17]	Status string
Add_del	Switch indicating whether modification to a viewplane is to be 0) add surfaces, 1) delete surfaces, or 2) move vertices
Bp	Indicates bit position – determines what digitizer key has been pressed
Bus_address	Bus_address of plotter
Cvterror	Flag returned from Convert3d that indicates if conversion successful or not (0/1)
G	New grid resolution input in response to key "fe"
Hid	Parameter specifying hidden-surface or wire-frame
Key	Contains the key code returned from the digitizer when a key is pressed
N	Number of vertices allowed
Ne2a	Number of edges allocated in Edge2d(*)
No2a	Number of surfaces allocated in Object2d(*)
Ns2a	Number of surface elements (edges) allocated in Surf2d(*)
Num_surf_ex	Number of surfaces user expects for object
Nv2a	Number of vertices allocated in Vertex2d(*)

Pic	Pointer to head of list in Object(*)
Pn	Indicates mass storage file containing 2D data
R	Loop index
Rv	Return variable indicating if Filenam$ already exists
S	Contains digitizer status when a serial poll is performed
Select_code	Select code of plotter
Status(4,4)	Matrix of concatenated transformations
Transform(0:N,4)	Array of vertices multiplied by current Status(*)
Xvp	X coordinate of viewpoint
Yvp	Y coordinate of viewpoint
Zvp	Z coordinate of viewpoint

Initviewpln

Initviewpln(INTEGER Digi,Ne2,No2,Ns2,Nv2)

Initviewpln provides for the initialization of the data base and viewplanes for the digitizer input subprograms.

Constviewpln is called by Initviewpln.

Filename: DginiB (09845-10060),DginiC(09845-10080)

Local Variable Definitions

Cper	Flag that indicates an error in the cutting plane intersection point
Row	Loop index
S	Status returned from serial poll of digitizer
View	Indicates what plane of the 2-D arrays is being initialized

Constviewpln

Constviewpln(INTEGER Error,Digi)

Constviewpln draws the viewplanes on the CRT.

Filename: DgcvwB (09845-10060), DgcvwC(09845-10080)

Display

Display(A$,INTEGER Select)

This subprogram provides for displaying characters on the digitizer display.

Filename: Digdsp

Local Variable Definitions

Alpha$[72]	Contains the list of characters from which to choose the correct display code from A(*)
D$[240]	Contains the list of commands to be sent to the digitizer in order to display the message contained in A$
A(72)	Contains the codes used to form the alphanumeric characters on the digitizer display
I	Loop index
P	Position of character of interest in the character set defined by Alpha$

Draw2d

Draw2d(INTEGER Digi)

This subprograms draws the three 2-D orthogonal views on the screen.

The subprograms called by Draw2d are Constviewpln and Recvpcoords.

Filename: Digdrw

Local Variable Definitions

Cedg	Points to current edge when drawing a 2-D surface
Edg	Used to retrieve the coordinates of the endpoints of a particular edge in a surface
Error	Indicates error in cutting plane description if not equal to 0
Objel	Points to beginning of surface description in Surf2d(*)
Plane	Indicates the viewplane currently being drawn
Xpln	X coordinate of a 2-D vertex in the located and scaled plane coordinate system
Ypln	Y coordinate of a 2-D vertex in the located and scaled plane coordinate system

Modvp

Modvp(S$,INTEGER Add_del,Digi,Mneps,Redundfctr)

This subprogram is used for the input and modification of the three orthogonal views.

Modvp calls the subprogram Recvpcoords.

Filename: DgmodB (09845-10060),DgmodC (09845-10080)

Local Variable Definitions

Ann$[15]	Annotation returned from the digitizer with a digitized point if desired
Annot	Numeric value of annotation returned from the digitizer
Beepdur	Beep duration – long beep for new point, short beep for old
Beepfreq	Beep frequency – high beep for new point, low beep for old
Beginsurf	Pointer of beginning of a possible surface to be deleted
Cedg	Current edge pointer used to follow a linked list surface description
Currsurf	Current surface pointer; points to surface currently being entered
De	Loop index; points to element of delete edge array
Deletedg(Mneps,2)	Array containing the edges of a surface being deleted
Deletptr	Points to last used element of Deletedg(*)
Deletsurf(0:Redundfctr)	Array containing a list of edges that form the surface to be deleted
Ds	Loop index; points to each surface to see if it is to be deleted
Edg	Pointer to edge when following a linked list description of a surface
Edg2fptr	Free pointer for the 2-D edge array
Edgptr	Used to point to an edge in the 2-D edge array
First	Flag indicating if a point is the first point of a surface being entered; 1 = 1st point, 0 = intermediate point
Key	Key code returned from digitizer
Kydn	Flag indicating a key has been pressed
Lstvtx	Points to last vertex entered when entering 2-D surface information
Nosrf	Number of possible surfaces to be deleted
Obj2fptr	Free pointer for the 2-D object array
Objel	Pointer used to follow through the 2-D surfaces of an object
Plane	1=XY plane; 2=YZ plane; 3=XZ plane
S	Status returned from digitizer on serial poll
Surf	Surface element of the 2-D object array
Surf2fptr	Free pointer into 2-D surface array
Surfel	Pointer used to search the 2-D surface array for unreferenced edges

Temp	Temporary storage
Vptrfirst	Pointer to a vertex that begins a surface being added or deleted
Vtx	Pointer used to search vertex array for any unreferenced vertices
Vtx2fptr	Free pointer for vertex array
Vtxptr	Points to vertex
Vtxptr1	Points to vertex
X	X coordinate of a vertex in the object coordinate system
Xp	X coordinate of location of digitizer cursor before a point is digitized
Xpln	X coordinate of a vertex in its local plane coordinate system
Y	Y coordinate of a vertex in the object coordinate system
Yp	Y coordinate of location of digitizer cursor before a point is digitized
Ypln	Y coordinate of a vertex in its local plane coordinate system
Z	Z coordinate of a vertex in the object coordinate system

Recvpcoords

Recvpcoords(INTEGER Plane,Vtxptr,REAL Xvp,Yvp)

This subprogram is needed to recover original viewplane coordinates from the 2-D vertex array coordinates.

Filename: Digrvc

Convert3d

Convert3d(INTEGER Digi,Error,Ne,No,Ns,Nv,Redundfctr)

This subprogram converts the three arrays of two-dimensional data entered from three orthogonal views into one array of three-dimensional data.

The subprograms called from Convert3d are Draw2d, Findsurf,and Indbad.

Filename: DgcvtB (09845-10060), DgcvtC (09845-10080)

Badedgfg	Flag used to indicate inconsistent edge information
Badsrffg	Flag used to indicate inconsistent surface information
Badvtxfg	Flag used to indicate inconsistent vertex information
C	Subscript indicating column number in Commonedg(*)
Ce	Number of edges is Commonedg(*)

Com1	Loop index
Com2	Loop index
Commonedg(0:Redundfctr,2)	Contains pointers to vertices that are connected in the primary plane
Commonpt	The number of vertices in the Secondary plane that have a primary coordinate equal to the primary coordinate of a point in the Primary plane
Commonvtx(Redundfctr)	Contains the vertices whose primary coordinate in the Secondary plane equal the primary coordinates in the Primary plane
Cpt	Loop index
Csurf2el	Current 2-D surface element used to follow a surface description
E	Loop index
Edg	Loop index
Edgetype	Edgetype is set to 1 for line segment
Edgfptr	Free pointer for the 3-D edge array
Edglink(Ne,4)	Array containing pointers referencing each 3-D edge to its representation in each of the three orthogonal views
Edgno	Number of edges in a 2-D surface
Foundanedgfg	Flag indicating that at least one 3-D edge found for a particular 2-D edge
Foundedgsec	Flag indicating that an edge in the secondary plane was found that corresponded with the edge in the Primary plane
Foundptfg	Flag indicating that a 3-D vertex was found for a particular point in the Primary plane
Foundsurffg	Flag indicates 3-D surface found
Fsrffg	Passed back from Findsurf - indicates that a consistent surface description has been found
Linkel	Element of the vertex link array
Objfptr	Free pointer for the 3-D Object array
P	Loop index
Primcoord	Primary coordinate
Primedg	Primary edge
Primpln	Primary plane
Primpt	Primary point
R	Loop index
Seccord	Secondary coordinate
Secpln	Secondary plane
Secpt	Secondary point

Sedg	Loop index
Strtedg	Starting edge of a surface used to find all 3-D edges that are represented by the 1st edge of a 2-D surface
Sumused	Sum of "used" conversion flags
Surf	Loop index
Surf2el	Pointer to 2-D surface element (edge)
Surffptr	Free pointer for the 3-D surface array
Tedg	Tertiary edge
Temp	Temporary
Tertcoord	Tertiary coordinate
Tertpln	Tertiary plane
Tertpt	Tertiary point
Tsurfel	Element of the temporary surface array used to hold surface description of a 3-D surface during the conversion process
V	Loop index
Vtxfptr	Free pointer for the 3-D Vertex array
Vtxlink(Nv,3)	Contains pointers from each 3-D vertex to its representation in each of the three orthogonal views

Indbad

Indbad(INTEGER Badplane,Plane,Vtx_edg_srf,Vtxedgptr)

This subprogram is used to indicate a vertex that cannot be located in another plane while converting from the 2-D (three orthogonal views) to a three-dimensional figure.

Indbad calls Recvpcoords.

Filename: Digibd

Local Variable Definitions

Angle	Angle in degrees; used to draw a circle around a vertex
Csurf2el	Current 2-D surface element used to follow a surface description
Endpt	Indicates which endpoint of an edge is being projected into the inconsistent plane
Surf2el	2-D surface element used to follow a 2-D surface description
Xpln	X coordinate of a vertex in local plane coordinates
Ypln	Y coordinate of a vertex in local plane coordinates

Findsurf

Findsurf(INTEGER Edgno,Firstedg,Fsrffg,Prevedg)

Findsurf finds all possible 3-D surfaces that are represented by a 2-D surface. Since there are a variable number of edges in the 2-D surface and a variable number of possible 3-D edges represented by each 2-D edge of the 2-D surface, this recursive subprogram tries all possible combinations in an effort to find the 3-D surfaces.

Filename: Digfsr

Local Variable Definitions

Edg	Loop index

Chapter 6
Manipulation Utilities

The Manipulation Utilities are located on the "Manipulation and Display" cartridge. Included among these are all subprograms needed for transformations.

Manipulation Subprograms

The subprograms included in this section are Crt_tra, Translate, Rotate_x, Rotate_y, Rotate_z, Scale, Center, Curve_setup, Circle_by_3_pts, Arc_gen, and Multiply.

Crt_tra (09845 -10060)

Crt_tra(INTEGER Edge(*),Hid,N,Object(*),Pic,Surface(*),Sur_tmp(*),Trans,
SHORT Status (*),Transform(*),Vertex(*),REAL Xvp,Yvp,Zvp)

This subprogram takes care of the overhead involved in transforming an object. It provides for user interaction and calls the subprograms necessary to perform the desired transformations.

The subprograms Center, Ginput, Parse_response, Plot, Rotate_x,Rotate_y, Rotate_z, Scale, Translate, and Unplot are called from Crt_tra.

NOTE

If Ginput is omitted so responses are no longer made while in GRAPHICS mode, all calls to Ginput must be replaced by an appropriate INPUT or LINPUT statement.

Filename: CrtB

Local Variable Definitions

Axis$[1]	Axis of rotation
Angle$[10]	Angle of rotation
Dx_dy_dz$[80]	Change in X, Y, and Z for translation
Responses$(10)	Contains responses returned from parsing the response string returned from Ginput
X_scale$[10]	X scaling factor response string
Y_scale$[10]	Y scaling factor response string

Z_scale$[10]	Z scaling factor response string
Num_of_res	Number of responses returned from parsing the response string returned from Ginput
Angle	Angle of rotation
Dx	Change in X direction for translation
Dy	Change in Y direction for translation
Dz	Change in Z direction for translation
Old_x_mid	X coordinate of previous "center" of object
Old_y_mid	Y coordinate of previous "center" of object
Old_z_mid	Z coordinate of privious "center" of object
Rx	X coordinate of "center" of object used for rotating
Ry	Y coordinate of "center" of object used for rotating
Rz	Z coordinate of "center" of object used for rotating
X_mid	X coordinate of current "center"
X_scale	X scaling factor
Y_mid	Y coordinate of current "center"
Y_scale	Y scaling factor
Z_mid	Z coordinate of current "center"
Z_scale	Z scaling factor

Crt_tra (09845-10080)

Crt_tra(INTEGER Edge(*),Hid,N,Object(*),Pic,Surface(*),Sur_tmp(*),Trans,
SHORT Color(*),Status(*),Transform(*),Vertex(*),REAL Xvp,Yvp,Zvp)

This subprogram takes care of the overhead involved in transforming an object. It provides for user interaction and calls the subprograms necessary to perform the desired transformations.

The subprograms Center, Plot, Rotate_x,Rotate_y, Rotate_z, Scale, Translate, and Unplot are called from Crt_tra.

Filename: CrtC

Local Variable Definitions

Axis$[1]	Axis of rotation
Angle	Angle of rotation
Dx	Change in X direction for translation
Dy	Change in Y direction for translation
Dz	Change in Z direction for translation
Old_x_mid	X coordinate of previous "center" of object
Old_y_mid	Y coordinate of previous "center" of object
Old_z_mid	Z coordinate of previous "center" of object

Rx	X coordinate of "center" of object used for rotating
Ry	Y coordinate of "center" of object used for rotating
Rz	Z coordinate of "center" of object used for rotating
X_mid	X coordinate of current "center"
X_scale	X scaling factor
Y_mid	Y coordinate of current "center"
Y_scale	Y scaling factor
Z_mid	Z coordinate of current "center"
Z_scale	Z scaling factor

Translate

Translate(SHORT Dx,Dy,Dz,Status(*))

Translate provides the utility to move an object from its current position to a new position by moving it the desired amount along the X, Y and Z axes.

Filename: Trans

Local Variable Definitions

T(4,4)	Basic matrix used to translate an object
Temp(4,4)	Temporary used when multiplying matrices

Rotate_x

Rotate_x(SHORT Angle,Rx,Ry,Rz,Status(*))

This subprogram computes the matrix for a rotation about the point (Rx,Ry,Rz) of "Angle" degrees with respect to the X-axis. It then concatenates it with the object's current Status(*).

In DEMO2 the "center" of the object is used as the point about which rotation is performed.

Filename: Rotx

Local Variable Definitions

R(4,4)	Basic rotation matrix
Temp1(4,4)	Temporary used in multiplying matrices
Temp2(4,4)	Temporary used in multiplying matrices
Trano(4,4)	Matrix used in translating from origin back to (Rx,Ry,Rz)
Tranr(4,4)	Matrix used in translating to the origin

Rotate_y

Rotate_y(SHORT Angle,Rx,Ry,Rz,Status(*))

This subprogram computes the matrix for a rotation about the point (Rx,Ry,Rz) of "Angle" degrees with respect to the Y-axis. It then concatenates it with the object's current Status(*).

In DEMO2 the "center" of the object is used as the point about which rotation is performed.

Filemane: Roty

Local Variable Definitions

R(4,4)	Basic rotation matrix
Temp1(4,4)	Temporary used in multiplying matrices
Temp2(4,4)	Temporary used in multiplying matrices
Trano(4,4)	Matrix used in translating from origin back to (Rx,Ry,Rz)
Tranr(4,4)	Matrix used in translating to the origin

Rotate_z

Rotate_z(SHORT Angle,Rx,Ry,Rz,Status(*))

This subprogram computes the matrix for a rotation about the point (Rx,Ry,Rz) of "Angle" degrees with respect to the Z-axis. It then concatenates it with the object's current Status(*).

In DEMO2 the "center" of the object is used as the point about which rotation is performed.

Filename: Rotz

Local Variable Definitions

R(4,4)	Basic rotation matrix
Temp1(4,4)	Temporary used in multiplying matrices
Temp2(4,4)	Temporary used in multiplying matrices
Trano(4,4)	Matrix used in translating from origin back to (Rx,Ry,Rz)
Tranr(4,4)	Matrix used in translating to the origin

Scale

Scale(SHORT Status(*),X_scale,Y_scale,Z_scale)

Scale provides the utility of enlarging or diminishing the size of a three-dimensional object.

Filename: Scale

Local Variable Definitions

S(4,4)	Basic scaling matrix
Temp(4,4)	Temporary used when multiplying matrices

Center

Center(INTEGER Edge(*),N,SHORT Status(*),Transform(*),Vertex(*),X_mid,
Y_mid,Z_mid)

This subprogram finds the approximate center point of an object. This means the center of a rectangular solid whose corners are the maximum and minimum values in the X, Y and Z directions. The point determined is not necessarily the center of mass. The maximum and minimum values of the coordinates of an object are found by 1) testing the endpoints of line segments, 2) generating points at 0, 90, 180 and 270 degrees for a circle, and 3) generating several lines for an arc. The subprograms called by Centr are Arc_gen, Curve_setup, and Multiply.

Filename: Centr

Local Variable Definitions

Angle	Loop index
I	Loop index
Index	Loop index
Subscript	The number of vertices to be tested for max and min values for circles and arcs
Curve_array(26,4)	Temporary array for storage of values to be tested in cases of circles and arcs
Curve_trans(26,4)	Temporary array used for matrix multiplication
Radius	Radius of circle
Xa	X-coordinate of first endpoint of arc
Xb	X-coordinate of second endpoint of arc
Xc	X-coordinate of last endpoint of arc
X_center	X-coordinate of center point of arc
X_max	Maximum value of object in X direction
X_min	Minimum value of object in X direction
Ya	Y-coordinate of first endpoint of arc
Yb	Y-coordinate of second endpoint of arc
Yc	Y-coordinate of last endpoint of arc
Y_center	Y-coordinate of center point of arc
Y_max	Maximum value of object in Y direction
Y_min	Minimum value of object in Y direction
Z_value	Z distance from X-Y plane to the parallel plane the arc was rotated into
Z_max	Maximum value of object in Z direction
Z_min	Minimum value of object in Z direction
Rotation(4,4)	Rotation information used to rotate arc/circle into a plane parallel to X-Y plane
Rot_mat(4,4)	Inverse of the Rotation matrix

Curve_setup

Curve_setup (INTEGER Edge (*), I, SHORT Radius, Vertex (*), Xa, Xb, Xc, X_center, Ya, Yb, Yc, Y_center, Z_value, REAL Rotation (*))

This subprogram allows you to rotate a circle or arc in space into a plane parallel to the X-Y plane. This, in turn, allows you to compute the radius and center of the circle by calling Circle_by_3_pts. Therefore, the subprogram Circle_by_3_pts must be in memory.

Filename: Crvset

Local Variable Definitions

Index	Loop index
A	Determinant of matrix A
A(3,3)	Matrix used in determining coefficients for equation of plane
B	Determinant of matrix B
B(3,3)	Matrix used in determining coefficients for equation of plane
C	Determinant of matrix C
C(3,3)	Matrix used in determining coefficients for equation of plane
D	Determinant of matrix D
D(3,3)	Matrix used in determining coefficients for equation of plane
Point_a(4)	Coordinates of first point of circle or arc
Point_a_prime(4)	Temporary storage during computation
Point_b(4)	Coordinates of second point of circle or arc
Point_b_prime(4)	Temporary storage during computation
Point_c(4)	Coordinates of third point of circle or arc
Point_c_prime(4)	Temporary storage during computation
Theta_xy	Angle between the plane of the circle and the X-Y plane
Theta_yz	Angle between the plane of the circle and the Y-Z plane
Za	Z-coordinate of first point of circle or arc
Zb	Z-coordinate of second point of circle or arc
Zc	Z-coordinate of third point of circle or arc
Rot_y(4,4)	Matrix to rotate about the Y-axis
Rot_z(4,4)	Matrix to rotate about the Z-axis

Circle_by_3_pts

Circle_by_3_pts (SHORT Radius, Xa, Xb, Xc, X_center, Ya, Yb, Yc, Y_center)

This subprogram, when given three points, determines the radius and the center of the circle they determine. Refer to the chapter on Formulas for a more detailed description of the process used.

Filename: C_by_3

Local Variable Definitions

Angle_cab	The measure of <CAB in degrees
Avg_side	Length of a+b+c divided by 2
Side_a	Length of side a
Side_b	Length of side b
Side_c	Length of side c
Slope	Slope of line segment and of perpendicular bisector
Slope1	Slope of line segment c and of perpendicular bisector
Slope2	Slope of line segment a and of perpendicular bisector
X1	X coordinate of first endpoint
X2	X coordinate of second endpoint
X_prime	X coordinate of midpoint of a line
Y1	Y coordinate of first endpoint
Y2	Y coordinate of second endpoint
Y_prime	Y coordinate of midpoint of a line
Y_intercept	Y coordinate where perpendicular bisector of a line crosses the Y-axis

Arc_gen

Arc_gen (INTEGER Subscript, SHORT Curve_array (*), Radius, Xa, Xb, Xc, X_center, Ya, Yb, Yc, Y_center, Z_value)

This subprogram provides information necessary for arc generation. It first finds the two angles that each endpoint makes with the origin, then arranges the angles so the arc will be drawn in a clockwise fashion.

Filename: Arcgn

Local Variable Definitions

Num_vertices	The number of vertices generated to draw the arc
Offset	Degrees added to determine the correct range
Angle	Loop index
Range	Number of degrees from the beginning vertex of the arc to the ending vertex of the arc

Stepsize	The stepsize to be used while drawing the arc
Theta	Temporary storage for computed angle
Theta1	Angle the first endpoint of an arc makes with the origin
Theta2	Angle the second endpoint of an arc makes with the origin
Theta3	Angle the last endpoint of an arc makes with the origin
X	Temporary X coordinate used in computing angles
Y	Temporary Y coordinate used in computing angles

Multiply

Multiply (INTEGER Flag, N, SHORT Status (*), Transform (*), Vertex (*))

This subprogram multiplies vertices by the current Status(*) to find the transformed points.

Filename: Mult

Local Variable Definitions

Free_ptrs(0:N)	Keeps the linked free pointer list
Index	Loop index
Last_element	Last element in Vertex(*)
Next_free_ele	Points to next free space in Vertex(*)

Chapter 7
Viewing Utilities

The Viewing Utilities are located on the "Manipulation and Display" cartridge. They deal with translating points from world coordinates to screen coordinates, choosing the viewpoint from which the object is viewed, clipping polygons when necessary, and providing perspective to the object being viewed.

Viewing Subprograms

The six subprograms included in this section are Viewpoint, Viewcoord, Perspective, Clip, Level, and Close_poly. For more detail concerning the algorithms used, see Chapter 3.

Viewpoint (09845-10060)

Viewpoint(INTEGER View,REAL Xvp,Yvp,Zvp)

This subprogram provides the setup for choosing a new viewpoint. If View=1, the object is viewed from the front, 2 means side, 3 means top, 4 means isometric, and 5 for user defined.

If the viewpoint is to be changed while the screen is in GRAPHICS mode, Ginput and Parse _response are called from Viewpoint, so these two subprograms must also be in memory.

NOTE

If Ginput is not used, you must replace all calls to Ginput with appropriate INPUT or LINPUT statements.

Filename: VwptB

Local Variable Definitions

Num _of _res	Number of responses returned from parsing the response string returned from Ginput
Responses$(10)	Contains responses returned from parsing the response string returned from Ginput
Xyz _vp$[80]	X, Y, and Z viewpoint response string

Viewpoint (09845-10080)

Viewpoint(INTEGER View,REAL Xvp,Yvp,Zvp)

This subprogram provides the setup for choosing a new viewpoint. If View=1, the object is viewed from the front, 2 means side, 3 means top, 4 means isometric, and 5 for user defined.

Filename: VwptC

Viewcoord

Viewcoord(INTEGER Hid,N,Sur_tmp(*),SHORT Transform(*),REAL Vcx,Vcy,Vsx,Vsy, Xvp,Yvp,Zvp)

The subprogram, Viewcoord, allows you to change the point from which the three dimensional object is viewed.

The subprograms called from Viewcoord are Clip, Multiply, and Perspective.

Filename: Viewcr

Local Variable Definitions

I	Index
Denom1	Square root of (Xvp squared + Yvp squared)
Dist	Distance from the viewpoint to the origin of the object coordinate system
Distscr	Distance to the screen from the eye
F	Zmax+(Zmax-Zmin)/10
Scrsize	Half the width of the screen
T1(4,4)	Matrix to translate the origin of the world coordinate system to the viewpoint (Xvp,Yvp,Zvp)
T2(4,4)	Matrix to rotates the "prime" coordinate system around X' axis by −90 degrees.
T3(4,4)	Matrix to rotate the "prime" coordinate system around Y' so (0,0,Zvp) lies on the Z' axis
T4(4,4)	Matrix to rotate the "prime" coordinate system about the X' axis so that the origin of the world coordinate system will lie on the Z' axis
T5(4,4)	Matrix to reverse the sense of the Z' axis to produce a left-handed coordinate system.
P(4,4)	Perspective matrix that provides homogeneous coordinates
S(4,4)	Matrix to scale and translate the X and Y coordinates of each vertex into the coordinate system in which they will be displayed

T(4,4)	Temporary matrix used when calling Multiply
Temp(4,4)	Temporary matrix used during multiplication
Temp _vertex(0:N,4)	Temporary vertex array used when calling Multiply
V(4,4)	Is the product of matrices T1,T2,T3,T4 and T5

Perspective

Perspective(INTEGER N,SHORT Transform(*))

This subprogram is used to provide the perspective distortion for three-dimensional figures. This is done by dividing each coordinate of a point by the associated homogeneous coordinate for that point.

Filename: Perspt

Local Variable Definitions

I	Loop index
Temp(0:N,4)	Temporary array used to hold vertices while division for perspective is performed

Clip

Clip(INTEGER N,Sur_tmp(*),SHORT Transform(*))

This subprogram implements a recursive polygon clipping algorithm designed by Ivan Sutherland and Gary Hodgeman. The algorithm can be found in SIGGRAPH '79. A flowchart of this algorithm can be found in Chapter Four of this manual.

Clip calls the subprograms Close _poly and Level.

Filename: ClipB (09845-10060), ClipC(09845-10080)

Local Variable Definitions

Count	Number of surfaces in Transform(*)
First _flag(6)	Each subscript refers to a level of recursion. An element equals 0 if no points have been passed to this level before, or 1 if the level has been passed at least one point before.
Level	Level of recursion
Surface	Loop index
Surface _count	Number of clipped surfaces
Vertex	Loop index
First(6,4)	In this array, row indicates the level of recursion and elements contained in each row are the X,Y,Z and W coordinate of the first point passed to that level of recursion.
Q(0:N,4)	Output points of surface clipper

Save(6,4)	In this array, row indicates the level of recursion and the elements contained in each row are the X,Y,Z and W coordinate of the last point passed to that level of recursion.
Surface _clip(0:N,2)	Temporary storage for clipped surfaces
Temp _tran(0:N,4)	Temporary storage for Transform(*)

Level

Level (INTEGER First _flag (*), Level, SHORT First (*), Px, Py, Pz, Pw, Q (*), Save (*))

This subprogram is called by Clip to process all surface points at a given level. There are six levels. These correspond to the six limiting planes of the viewing pyramid which are top, bottom, left, right, hither, and yon. See the description of the clipping algorithm found in Chapter 3.

Level calls itself.

Filename: Level

Local Variable Definitions

Alpha	Fraction of the line left after clipping
Ix	X coordinate of intersection point
Iy	Y coordinate of intersection point
Iz	Z coordinate of intersection point
Iw	Homogeneous coordinate of intersection point

Close

Close_poly (INTEGER First_flag(*),Level,SHORT First(*),Q(*),Save(*))

This subprogram is called by the subprogram Clip. Its purpose is to close polygons that have been clipped.

Close _poly calls the subprogram Level.

Filename: Close

Local Variable Definitions

Alpha	The clipped fraction of the line
Ix	The X coordinate of the intersection of line SF and the limiting plane
Iy	The Y coordinate of the intersection of line SF and the limiting plane
Iz	The Z coordinate of the intersection of line SF and the limiting plane
Iw	The homogeneous coordinate of the intersection of the line SF and the limiting plane

Chapter 8
Output Utilities

The Output Utilities are located on the "Manipulation and Display" cartridge. They include the subprograms needed to plot an object as a wire-frame or with hidden surfaces removed. Also included is a subprogram that allows for plotting an object on a peripheral plotting device.

Output Subprograms

These utilities include Plot, Unplot, Hidden_surface, Scan, and Plotter_setup. A description of each follows.

Plot

(09845-10060)
Plot(INTEGER Edge(*),Hid,N,Object(*),Pic,Surface(*),Sur_tmp(*),SHORT Status(*),
Transform(*),Vertex(*),REAL Xvp,Yvp,Zvp)

(09845-10080)
Plot(INTEGER Edge(*),Hid,N,Object(*),Pic,Surface(*),Sur_tmp(*),SHORT Color(*),
Status(*),Transform(*),Vertex(*),REAL Xvp,Yvp,Zvp)

This subprogram plots a wire-frame three-dimensional object and provides the setup for the hidden-surface removal subprogram. Subprograms called by Plot are Arc_gen, Curve_setup, Hidden_surface, Multiply, and Viewcoord.

Filename: PlotB(09845-10060),PlotC(09845-10080)

Local Variable Definitions

Angle	Loop index
Count	Index into Temp_tran(*)
Edge	Pointer to current edge being processed
Edge_next	Index to next edge in Edge(*)
I	Loop index
Object	Index into Object(*)
Subscript	Number of elements in Curve_array(*)
Surface	Index into Surface(*)
Surface_count	Index into Sur_tmp(*)

Surface_flag	Set to 0 if first edge of a surface
Vertex1	Index into Transform(*) where surface begins
Vertex2	Index into Transform(*) where surface ends
Curve_array(26,4)	Array of vertices for line segments composing a circle or arc; generated in plane parallel to X-Y plane
Curve_trans(26,4)	Circle's or arc's endpoints in 3-D
Old_x2	Temporary variable to hold old X coordinate of 2nd endpoint
Old_y2	Temporary variable to hold old Y coordinate of 2nd endpoint
Old_z2	Temporary variable to hold old Z coordinate of 2nd endpoint
Radius	Radius of circle or arc
Temp_tran(0:N,4)	Temporary used to store transformed points
Temp_x1	Temporary variable to hold old X coordinate of 1st endpoint
Temp_y1	Temporary variable to hold old Y coordinate of 1st endpoint
Temp_z1	Temporary variable to hold old Z coordinate of 1st endpoint
Xa	X-coordinate of 1st point of circle or arc
Xb	X-coordinate of 2nd point of circle or arc
Xc	X-coordinate of 3rd point of circle or arc
X_center	X-coordinate of center point that is calculated in Circle_by_3_pts
X1	Temporary X coordinate
X2	Temporary X coordinate
Ya	Y-coordinate of 1st point of circle or arc
Yb	Y-coordinate of 2nd point of circle or arc
Yc	Y-coordinate of 3rd point of circle or arc
Y_center	Y-coordinate of center point that is calculated in Circle_by_3_pts
Y1	Temporary Y coordinate
Y2	Temporary Y coordinate
Z_value	Z distance from X-Y plane to the plane the circle or arc was rotated into
Z1	Temporary Z coordinate
Z2	Temporary Z coordinate
Rotation(4,4)	Rotation information used to rotate arc or circle into a plane parallel to the X-Y plane

Rot_mat(4,4)	Inverse of Rotation(*) used to rotate arc or circle back to original orientation
Vcx	X-coordinate of center point of screen in screen coordinates
Vcy	Y-coordinate of center point of screen in screen coordinates
Vsx	Half the X range of the screen in screen coordinates
Vsy	Half the Y range of the screen in screen coordinates

Unplot

Unplot(INTEGER Sur_tmp(*),SHORT Transform(*))

Unplot erases a wire-frame plot, when GCLEAR is not desired.

Filename: Unplot

Local Variable Definitions

I	Loop index
Surface	Loop index
Vertex1	Points to vertex of segment to be erased
Vertex2	Points to vertex of segment to be erased

Hidden_surface (09845-10060)

Hidden_surface(INTEGER N,Sur_tmp(*),SHORT Transform(*))

This subprogram sets up the data base for the hidden-surface algorithm. More detail about the hidden surface algorithm used here can be found in Chapter Three of the manual.

The subprogram Scan is called by Hidden_surface.

Filename: HidsrB

Local Variable Definitions

Edge_hid	Index denoting next available space in Edge_hid(*)
Edgesort(N)	Contains indices to Edge_hid(*) in sorted order (from largest to smallest) - sorted on initial y values for each edge.
Elist	Pointer to the first edge stored in Edge(*) by its Y max value
First_vertex	Pointer to first vertex of a surface
I	Loop index
J	Loop index
J1	Temporary used for subscript when sorting Edge_hid(*)
J2	Temporary used for subscript when sorting Edge_hid(*)

Last_vertex	Pointer to last vertex of a surface
M1	Subscript used when calculating Dzdx for a polygon
M2	Subscript used when calculating Dzdx for a polygon
Point_count	Contains the count for points used in calculating Dzdx
Poly	Number of polygons in Poly_hid(*)
Surface	Loop index
Swap	Indicates the number of swaps that have taken place while sorting Edge_hid(*)
Temp	Temporary used when sorting Edge_hid(*)
Vertex	Loop index
Denom	Temporary used in calculating the change in Z with respect to X for a plane
Dxdy	Change in X with respect to Y for an edge of a polygon (set to 99999 if Yinit=Yfinal)
Dzdy	Change in Z with respect to Y for an edge of a polygon (set to 99999 if Yinit=Yfinal)
Dzdx	Change in Z with respect to X for a polygon
Edge_hid(N,0:10)	List of edge-defining vertices
Numer	Temporary used in calculating the change in Z with respect to X for a plane
Poly_hid(2*N,0:4)	List of surface-defining information
Sum	Value returned when calculating numerator or denominator for Dzdx
Temp(3,3)	Temporary array used in calculation of Dzdx
Xinit	The X value of the endpoint of an edge that has the largest Y value.
Xfinal	The X value of the endpoint of an edge that has the smallest Y value.
Yinit	The Y value of the endpoint of an edge that has the largest Y value
Yfinal	The Y value of the endpoint of an edge that has the smallest Y value
Zinit	The Z value of the point along an edge that has the largest Y value

Hidden_surface (09845-10080)

Hidden_surface(INTEGER N,Sur_tmp(*),SHORT Color(*),Transform(*))

This subprogram sets up the data base for the hidden-surface algorithm. More detail about the hidden surface algorithm used here can be found in Chapter Three of the manual.

If you use the 9872 as a color output device, pens must be put into stalls 1 through 4 in the following order: black, red, green, blue. The program will then convert the area color parameters of each surface to a pen color.

If you are making overhead slides, be sure to slow down the pen by inserting the following statements after the plotter is declared: PRINTER IS Select_code,Bus_address

```
PRINT "VS10;"
PRINTER IS 16
```

The subprogram Scan is called by Hidden_surface.

Filename: HidsrC

Local Variable Definitions

Bus_address	Bus address of 9872
Edge_hid	Index denoting next available space in Edge_hid(*)
Edgesort(N)	Contains indices to Edge_hid(*) in sorted order (from largest to smallest); sorted on initial y values for each edge.
Elist	Pointer to the first edge stored in Edge(*) by its Y maximum value
First_vertex	Pointer to first vertex of a surface
I	Loop index
J	Loop index
J1	Temporary used for subscript when sorting Edge_hid(*)
J2	Temporary used for subscript when sorting Edge_hid(*)
Last_vertex	Pointer to last vertex of a surface
M1	Subscript used when calculating Dzdx for a polygon
M2	Subscript used when calculating Dzdx for a polygon
Pen	Loop index
Plot	Equals 0 for plot on CRT; equals 1 to 4 depending on current pen selection for 9872 plot
Point_count	Contains the count for points used in calculating Dzdx
Poly	Number of polygons in Poly_hid(*)
Select_code	Select code of 9872
Surface	Loop index
Swap	Indicates the number of swaps that have taken place while sorting Edge_hid(*)
Temp	Temporary used when sorting Edge_hid(*)
Vertex	Loop index
Denom	Temporary used in calculating the change in Z with respect to X for a plane

Dxdy	Change in X with respect to Y for an edge of a polygon (set to 99999 if Yinit=Yfinal)
Dzdy	Change in Z with respect to Y for an edge of a polygon (set to 99999 if Yinit=Yfinal)
Dzdx	Change in Z with respect to X for a polygon
Edge_hid(N,0:10)	List of edge-defining vertices
Numer	Temporary used in calculating the change in Z with respect to X for a plane
Poly_hid(2*N,0:7)	List of surface-defining information
Sum	Value returned when calculating numerator or denominator for Dzdx
Temp(3,3)	Temporary array used in calculation of Dzdx
Xinit	The X value of the endpoint of an edge that has the largest Y value
Xfinal	The X value of the endpoint of an edge that has the smallest Y value
Yinit	The Y value of the endpoint of an edge that has the largest Y value
Yfinal	The Y value of the endpoint of an edge that has the smallest value
Zinit	The Z value of the endpoint of an edge that has the largest Y value

Scan (09845-10060)

Scan(INTEGER Edgecount,Elist,Polycount,SHORT Edge(*),Poly(*))

This subprogram is used to remove hidden surfaces, one scan line at a time. The outline of each visible surface is plotted.

Filename: ScanB

Local Variable Definitions

E	Pointer to new edge being processed
I	Loop index
Index	Loop index
Inpend	Pointer to end of Inpoly list
Inpoly	Pointer to the head of list of active polygons sorted by Z values
Insert_flag	Flag used to indicate when the head of list insertions are allowed
Old_edge	Edge pointer used to determine if last edge was horizontal
Old_xend	Previous end of the Xsort list
Outcount	Number of dots in dot buffer

Output(1:10,1:2)	Dot buffer
P	Pointer to a polygon being sorted into Inpoly list
Polytmp	Temporary pointer to Inpoly during update of list
Pptr	Pointer to Inpoly list during update of list
S	Subscript used to indicate pointer to 1st polygon and 2nd (if there is one) that edge being processed belongs to
T	Temporary pointer to present edge in Xsort list
T1	Temporary pointer to next edge in Xsort list
Temp_xend	Temporary pointer to end of Xsort list
Temp_xsort	Temporary pointer to head of Xsort list
Xend	Pointer to end of Xsort list
Xptr	Pointer into Xsort list
Xsort	Pointer to head of list of all edges sorted by X value
Y	Y value of point during output
Yval	Y value of current scan line
Dx	Distance from current sample point to next sample point
Oldxlsp	Saves the value of the Xlsp
V	X-coordinate of new edge being sorted by X value
Xlsp	X-coordinate of left sample point
Xrsp	X-coordinate of right sample point

Scan (09845-10080)

Scan(INTEGER Edgecount,Elist,Plot,Polycount,SHORT Edge(*),Poly(*))

This subprogram is used to remove hidden surfaces, one scan line at a time. Color shading is used to display a surface.

Filename: ScanC

Local Variable Definitions

Buffer_count	Number of line segments buffered on a scan line
E	Pointer to new edge being processed
I	Loop index
Inpend	Pointer to end of Inpoly list
Inpoly	Pointr to head of list of active polygons sorted by Z values
Insert_flag	Flag used to indicate when the head of list insertions are allowed
Old_xend	Previous end of the Xsort list
P	Pointer to a polygon being sorted into Inpoly list
Polytmp	Temporary pointer to Inpoly during update of list
Pptr	Pointer to Inpoly list during update of list

S	Subscript used to indicate pointer to 1st polygon and 2nd (if there is one) that edge being processed belongs to
T	Temporary pointer to present edge in Xsort list
T1	Temporary pointer to next edge in Xsort list
Temp_xend	Temporary pointer to end of Xsort list
Temp_xsort	Temporary pointer to head of Xsort list
Xend	Pointer to end of Xsort list
Xptr	Pointer into Xsort list
Xsort	Pointer to head of list of all edges sorted by X value
Yval	Y value of current scan line
Buffer(100,2)	Array of X left and X right sample points for a scan line
Dx	Distance from current sample point to next sample point
V	X-coordinate of new edge being sorted by X value
X_draw	X-coordinate of point to draw to
X_move	X-coordinate of point to move to
Xlsp	X-coordinate of left sample point
Xrsp	X-coordinate of right sample point

Plotter_setup (09845-10060)

Plotter_setup(Plotter$,INTEGER,Bus_address,Select_code)

This program provides the setup for plotting to a plotter other than the CRT.

If used from GRAPHICS mode, Plotter_setup must call Ginput.

Filename: PltstB

Local Variable Definitions

Answer$[3]	Contains YES/NO response
Bus_address$[2]	Bus address of HP-IB plotter in string form
Select_code$[2]	Select code of plotter in string form
Plotter	1= CRT, 2 = other plotter

Plotter_setup (09845-10080)

Plotter_setup (Plotter$,INTEGER,Bus_address,Select_code)

This program provides the setup for plotting to a plotter other than the CRT.

Filename: PltstC

Local Variable Definitions

Answer$[3]	Contains YES/NO response
Plotter	1= CRT, 2 = other plotter

Chapter 9
Additional Utilities

The Additional Utilities are also found on the "Manipulation and Display" cartridge. They include one program, five subprograms and two binary programs. The program Strip can be used to remove comments from a program to save memory. The subprograms Dump, Ginput and Parse_response, and Letter and Draw_char provide for dumping graphics to a printer, entering responses while in GRAPHICS mode and lettering with different fonts while in graphics, respectively. The two binary programs mentioned above are used in conjunction with the subprograms Dump and Ginput. The binary Dmpg#b provides the special utility of dumping graphics to a specified HP-IB printer and the binary called Binary provides the special utility of entering data while in GRAPHICS mode. Ginput, Parse_response and the two binaries are for use on the 9845B with pack 09845-10060.

BASIC Utilities

Strip

This utility is a main program by itself. It is used to strip out all comment lines from a program. Since most of the subprograms in this pack are heavily commented, you may wish to delete these remarks to save memory. Note that you must have already created the destination file.

Filename: Strip

Local Variable Definitions

Destination$[10]	New file containing no comments
Line$[160]	Used to hold one line of program code
Source$[10]	Old file containing commented program

Dump (09845-10060)

Dump

This subprogram dumps the 9845 graphics raster to the internal thermal line printer or other Hewlett-Packard dot matrix printer from GRAPHICS mode.

When used from GRAPHICS mode, Dump calls Ginput. In any case Dump requires the binary, Dmpg#b.

NOTE

If you do not wish to use Ginput, replace all calls to Ginput with an analogous INPUT or LINPUT statement.

Filename: DumpB

Local Variable Definitions

Answer$[3]	Answer to YES/NO question
Bus_address$[3]	Response string for bus address
Printer$[2]	Response string for printer choice
Select_code$[2]	Response string for select code
Bus_address	Bus address
Printer	Printer choice,1=internal printer,2=other
Select_code	Select code of printing device

Dump (09845-10080)

Dump

This subprogram dumps the 9845 graphics raster to the internal thermal line printer or other Hewlett-Packard dot matrix printer.

Filename: DumpC

Local Variable Definitions

Answer$[3]	Answer to YES/NO question
Bus_address	Bus_address
Printer	Printer choice,1=internal printer,2=other
Select_code	Select code of printing device

Ginput (09845-10060)

Ginput(Gprompt$,Var$)

This subprogam allows the user to input data while in GRAPHICS mode. Since the information is returned in string form, it is up to the user to parse the string for numbers or expressions. It requires that the binary program called Binary be present in memory. See section on binary programs for more information.

NOTE

Ginput must be removed and analogous INPUT or LINPUT statements used in place of the CALL statements if you wish to enter data from alphanumeric mode as opposed to GRAPHICS mode.

Filename: Ginput

Local Variable Definitions

Kbd$[80]	Buffer holding input information; used in conjunction with ON KBD statement
Save$[80]	Temporary string used when removing control characters form input lines
C1	ASCII value of control key
Cpos	Current cursor position in dot units
I	Loop index
Insflag	Equals 1 while in character inserting mode
Ipos	Vertical position where input line begins
J	Loop index
Lpos	Current position of cursor in letter units
Numbr	ASCII value of pressed key
Ppos	Vertical position where prompt line begins
Spos	Horizontal position where prompt line begins

Parse_response (09845-10060)

Parse_response(Responses$(*),Str$,INTEGER Num_of_res)

This recursive subprogram parses the response string returned by the Ginput subprogram. You can have up to nine, one character responses.

NOTE

Parse_response is only needed if the subprogram, Ginput, is used.

Filename: Parse

Local Variable Definitions

Comma	If non-zero, then a comma was encountered in the string
Length	Length of the string being processed

Letter and Draw_char

These two subprograms are used to provide the user with four lettering fonts: Stick, Script, Roman and Gothic. The subprogram Letter does the initial setup for the lettering, while Draw_char does the actual drawing. The lettering fonts are stored on the four data files "stick", "script", "roman", and "gothic". Note that the Stick and Roman fonts include foreign characters. For pack 09845-10060, the subprogram "Ginput" and the binary program "Binary" must also be in memory, since Ginput is called from Letter.

Letter (INTEGER Basic_element (*), Char_index, Char_table (*), Old_font, Spacing(*))

Filename: LetrB (09845-10060) , LetrC (09845-10080)

Local Variable Definitions

A$[80]	String to be labeled
Font$[1]	Type of font in string form
Size$[1]	Size of letters in string form
Char_val	ASCII value of character to be drawn
First_compound	Pointer to first compound element in Char_table(*)
Font	Equals 1 for stick, 2 for script, 3 for roman, 4 for gothic
I	Loop index
Index	Pointer to first compound element in the Char_index(*) table
Penc	Pen control parameter
Size	Equals 1 for small, 2 for medium, 3 for large
X	Used in positioning a character
X1	Used in positioning a character
X2	Used in positioning a character
Y	Used in positioning a character

Draw_char (INTEGER Basic_element (*), Char, Char_index (*), Char_table (*), First_compound, Index, Penc)

Filename: Drawch

Local Variable Definitions

Delta_x	X increment of next point to draw to
Delta_y	Y increment of next point to draw to
Ele	Value of character to be drawn
I	Loop index

Binary Utilities

Using Binaries

Two binary programs are provided for pack 09845-10060. For those of you who have never used a binary, there are a few simple rules.

To load a binary program into memory:

1. Type: LOAD BIN "(name of binary)"
2. Press: EXECUTE

From then on you can only get other programs into memory that were saved in data form, or else you will destroy the binary. Binaries may be repeatedly loaded in without affecting any type of program already in memory. Please note that binaries cannot be separated once in memory.

To store a program WITH a binary in it:

1. Type: STORE "(name of program)"
2. Press: EXECUTE

To save a program WITHOUT the binary in it:

1. Type: SAVE "(name of program)"
2. Press: EXECUTE

Once a binary has been stored with a program, use the normal LOAD command to load the program along with the binary back into memory.
Binary programs may be copied like any other file. On a file catalog their program type is BPRG.

Dump Graphics Binary (09845-10060)

This 9845B binary is used to dump the graphics raster to a dot matrix printer. It is located on file "Dmpg#b" and contains only one keyword. It has the following syntax:

DUMP GRAPHICS #Select_code[[[, <Bus_address>]<; Lower>]<, Upper>]

Select_code is the select code of the printer. Bus_address is the HP-IB bus address of the printer . Lower is the last row of dots to be dumped. Upper is the first row of dots to be dumped.

Gprint Binary (09845-10060)

This 9845B binary is used to implement a "GINPUT" command. It is located on file "Binary" and contains the three keywords, GPRINT, GERASE, and LCURSOR.

GPRINT will print a string on the graphics raster and has the following syntax:

GPRINT <X>,<Y>,<string variable>

X and Y are the coordinates of the lower left corner of the first character to be printed (in dot units). The string variable is the string to be printed.

GERASE will erase a block of dots on the graphics raster and has the following syntax.

GERASE <X>,<Y>,<delta X>,<delta Y>

X and Y are the coordinates of the lower left corner of the area to be erased (in dot units). Delta X and delta Y define the length and width of the block to be erased (also in dot units).

LCURSOR is used to position the cursor on the graphics raster and has the following syntax:

LCURSOR <X>,<Y>

All characters printed using the GPRINT keyword fit in a block 7 by 12 dots. So, when positioning the cursor beneath a character, X refers to the middle dot of the character and Y is the row of dots the cursor is to appear on.

Chapter 10
Programming Aids

Example Driver Programs

The programs described in the following pages provide examples for utilizing the Three-Dimensional Graphics Utilities. A list of User Instructions has been provided for each demo driver. It is important that you realize the demos were not designed to solve any specific application but to familiarize you with the utilities and how they interact. These driver programs demonstrate 1) Manual Entry, 2) Object Transformations, 3) Digitizer Entry, and for pack 09845-10080, 4) Surface Color Entry. The drivers are saved as DRIV1B, DRIV2B and DRIV3B for 09845-10060, and as DRIV1C, DRIV2C, DRIV3C, and DRIV4C for 09845-10080. The driver programs can be located on the Manipulation and Display cartridge.

Manual Entry

This driver provides for the use of the alphanumeric keyboard as a means of entering three-dimensional objects. Each object is made up of surfaces that are comprised of circles, arcs or line segments. Circles may be considered surfaces in themselves. If an object that contains a curved surface is desired, the curved surface must be approximated using line segments. For example, to enter a cone, the following figure could be entered.

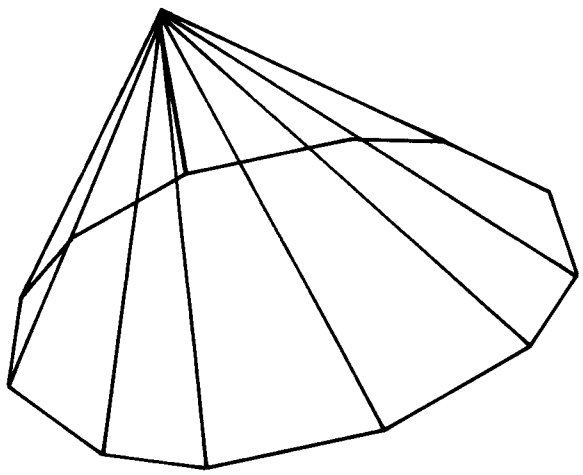


Figure 10.1

In this figure, trianglar surfaces are used to approximate the curved surface of a cone.

09845-10060

For pack 09845-10060, the subprograms that must be in memory are: One_obj_entry, Initialize, Multiply, List, Relist, Surface_setup, Check_arrays, Enter_line, Plane_equation, Save_line, Edge_ver_entry, Surface_entry, Enter_circle, Save_circle, Enter_arc, Save_arc, Database_delete,Change_point, and One_obj_store.
After getting DRIV1B, link the subprogram files: Obj_en, Init, Mult, List, RlistB, Surset, Check, Line, Plnequ, Sline, Edgver, Surent, Circ, Scir, Arc, Sarc, Surdel, Chngpt, and Obj_st. Store on file "DEMO1". Use the KEYS file named "Keys3D" along with DEMO1.

09845-10080

For pack 09845-10080, the subprograms that must be in memory are: One_obj_entry, Initialize, Multiply, List, Relist, Surface_setup, Check_arrays, Enter_line, Plane_equation, Save_line, Edge_ver_entry, Surface_entry, Enter_circle, Save_circle, Enter_arc, Save_arc, Database_delete, Change_point, and One_obj_store.
After getting DRIV1C, link the subprogram files: Obj_en, Init, Mult, List, RlistC, Surset, Check, Line, Plnequ, Sline, Edgver, Surent, Circ, Scir, Arc, Sarc, Surdel, Chngpt, and Obj_st. Store on file "DEMO1". Use the KEYS file named "Keys3D" when using DEMO1.

Demo 1 User Instructions

1. To load in the demo and begin execution:
   a. Type: LOAD "DEMO1"
   b. Press: EXECUTE
   c. Press: RUN

2. When "ENTER MINIMUM X VALUE,MAXIMUM X VALUE" is displayed:
   a. Enter: minimum x value of data range, maximum x value of data range (Default is -100,100)
   b. Press: CONT

3. When "ENTER MINIMUM Y VALUE,MAXIMUM Y VALUE" is displayed:
   a. Enter: minimum y value of data range, maximum y value of data range (Default is -100,100)
   b. Press: CONT

4. When "ENTER MINIMUM Z VALUE,MAXIMUM Z VALUE" is displayed:
   a. Enter: minimum z value of data range, maximum z value of data range (Default is -100,100)
   b. Press: CONT

5. When "DO YOU WISH TO USE A PRIVIOUSLY STORED FILE?" is displayed:
   a. Enter: YES for yes
   b. Enter: NO for no
   c. Press: CONT

6. If you entered "YES" then answer the prompts to get the file in memory.
7. The special function keys will be displayed.
8. To enter a surface:
   a. Press: K0 or K24
   b. The number of the surface being entered will be displayed.
   c. Enter the surface desired using K1,K2,K3. NOTE: A surface can be made up of line segments, line segments in conjunction with arcs, two or more arcs, or it can be a circle.
   d. Go to 8b.

NOTE

You **must** press this key (K0 or K24) before defining any new surface except for circles!

---

9. To enter a line:
   a. Press: K1 or K25
   b. Enter: coordinates of first endpoint
   c. Enter: coordinates of second endpoint
   d. The coordinates of the line segment will be displayed
   e. The program will stay in line drawing mode until another special function key is pressed.
   f. Goto 9c.

10. To enter a circle by 3 points.
    a. Press: K2 or K26 (Do not press K0 before pressing K2)
    b. Enter: coordinates of point 1
    c. Enter: coordinates of point 2
    d. Enter: coordinates of point 3
    e. If 2 or 3 points are coincidental, or if all 3 points are collinear, an error message will be displayed.
    f. The program will stay in circle drawing mode until another special function key is pressed.
    g. Go to 10b.

11. To enter an arc by 3 points:
    a. Press: K3 or K27
    b. Enter: coordinates of first endpoint
    c. Enter: coordinates of second point on arc
    d. Enter: coordinates of last endpoint
    e. If 2 or 3 points are coincidental, or if all 3 points are collinear, an error message will be displayed.
    f. The program will stay in arc drawing mode until another special function key is pressed.
    g. Go to 11b.

12. To delete a surface:
    a. Press: K5 or K29
    b. Enter: number of surface to be deleted
    c. The program will stay in surface deleting mode until another special function key is pressed.
    d. Go to 12b.

13. To change a point:
    a. Press: K6 or K30
    b. Enter: number of surface containing the point
    c. Enter: coordinates of point to be changed
    d. Enter: new coordinates of point
    e. The program will stay in change point mode until another special function key is pressed.
    f. Go to 13b.

NOTE

Change_point is only an error correction routine to be used when a point has been entered incorrectly. It cannot be used to "redesign" a surface. If a surface needs massive changes you should delete it and enter a new surface.

---

14. To store the object:
    a. Press: K7 or K31
    b. Enter: file name by following the prompts

Filename: DRIV1B (09845-10060)

Local Variable Definitions

Answer$[3]	Contains YES/NO response
File$[12]	Name of file to be used
Closed_flag	Equals 1 if surface is now closed
Edge(0:250,5)	Linked list of pointers to vertices that comprise edges; also includes edge type
Edge_count	Number of edges in the surface currently being defined
Edge_flag	Equals 1 if Edge array is full
Edge_index	Pointer to stored edge in Edge array
Edge_type	Equals 1 for line, 2 for circle, 3 for arc
Line	Equals 0 if first line of sequence to be drawn; equals 1 if already in line drawing mode
List_flag	0 if a list is not desired; 1 if a list is desired
N	Number used in dimensioning data base arrays
Object	Pointer to last surface pointer of object in Object array
Object(0:125,3)	Linked list of pointers to surfaces that comprise objects
Pic	Pointer to first surface in object array
Surface	Pointer to first edge of surface to be defined
Surface(0:500,3)	Linked list of pointers to edges that comprise surfaces
Surface_count	Current number of surfaces in an object
Surface_flag	Equals 1 if surface has been declared
Vertex_flag	Equals 1 if vertex array is full
Firstpt(3)	Coordinates of first point in surface
Lastpt(3)	Coordinates of last point in surface
Offset(3)	Absolute values of X,Y and Z values
Range(3)	Ranges of X,Y and Z values
Real_pts(3,4)	Coordinates of endpoints that will be stored in the data base
Status(4,4)	Matrix of concatenated transformations
Status_inv(4,4)	Inverse of Status matrix
Trans_pts(3,4)	Coordinates of endpoints in user units
Transform(0:250,4)	Array of vertices multiplied by current Status array
Vertex(0:250,4)	Linked list of all edge-defining vertices
Xmax	Maximum X value in user units
Xmin	Minimum X value in user units
Ymax	Maximum Y value in user units
Ymin	Minimum Y value in user units

Zmax	Maximun Z value in user units
Zmin	Minimum Z value in user units
A	Determinant of A(*)
A(3,3)	Matrix used to determine plane that surface is in
B	Determinant of B(*)
B(3,3)	Matrix used to determine plane that surface is in
C	Determinant of C(*)
C(3,3)	Matrix used to determine plane that surface is in
D	Determinant of D(*)
D(3,3)	Matrix used to determine plane that surface is in

Filename: DRIV1C (09845-10080)

Local Variable Definitions

Answer$[3]	Contains YES/NO response
File$[12]	Name of file to be used
Closed_flag	Equals 1 if surface is now closed
Edge(0:250,5)	Linked list of pointers to vertices that comprise edges; also includes edge type
Edge_count	Number of edges in the surface currently being defined
Edge_flag	Equals 1 if Edge array is full
Edge_index	Pointer to stored edge in Edge array
Edge_type	Equals 1 for line, 2 for circle, 3 for arc
Line	Equals 0 if first line of sequence to be drawn; equals 1 if already in line drawing mode
List_flag	0 if a list is not desired; 1 if a list is desired
N	Number used in dimensioning data base arrays
Object	Pointer to last element used in Object array
Object(0:125,3)	Linked list of pointers to surfaces that comprise objects
Pic	Pointer to first surface in object array
Surface	Pointer to first edge of surface to be defined
Surface(0:500,3)	Linked list of pointers to edges that comprise surfaces
Surface_count	Current number of surfaces in an object
Surface_flag	Equals 1 if surface has been declared
Vertex_flag	Equals 1 if vertex array is full
Color(125,3)	Color parameters for the current object
Firstpt(3)	Coordinates of first point in surface
Lastpt(3)	Coordinates of last point in surface
Offset(3)	Absolute values of X,Y and Z values

Range(3)	Ranges of X,Y and Z values
Real_pts(3,4)	Coordinates of endpoints that will be stored in the data base
Status(4,4)	Matrix of concatenated transformations
Status_inv(4,4)	Inverse of Status matrix
Trans_pts(3,4)	Coordinates of endpoints in user units
Transform(0:250,4)	Array of vertices multiplied by current Status array
Vertex(0:250,4)	Linked list of all edge-defining vertices
Xmax	Maximum X value in user units
Xmin	Minimum X value in user units
Ymax	Maximum Y value in user units
Ymin	Minimum Y value in user units
Zmax	Maximun Z value in user units
Zmin	Minimum Z value in user units
A	Determinant of A(*)
A(3,3)	Matrix used to determine plane that surface is in
B	Determinant of B(*)
B(3,3)	Matrix used to determine plane that surface is in
C	Determinant of C(*)
C(3,3)	Matrix used to determine plane that surface is in
D	Determinant of D(*)
D(3,3)	Matrix used to determine plane that surface is in

Object Transformations

This driver demonstrates translation, rotation, and scaling of an object. It also allows for choosing the viewpoint from which to observe the object. Once a particular view is chosen that view of the object can be drawn on a plotter or dumped to a printer. Also, hidden surfaces of the object can be removed and the result plotted to the CRT (or the 9872 plotter if using 09845-10080).

09845-10060

The following subprograms and binaries must be present in addition to DRIV2B:

One_obj_entry, Ginput, Parse_response, Viewpoint, Plot, Multiply, Curve_setup, Circle_by_3_pts, Arc_gen, Viewcoord, Clip, Level, Close_poly, Perspective, Crt_tra, Center, Rotate_x, Rotate_y, Rotate_z, Unplot, Translate, Scale, Dump, Plotter_setup, Hidden_surface, Scan, One_obj_store, Binary, and Dmpg#b.

After loading the binaries called Binary and Dmpg#b and getting DRIV2B into memory, link the subprogram files: Obj_en, Ginput, Parse, VwptB, PlotB, Mult, Crvset, C_by_3, Arcgn, Viewcr, ClipB, Level, Close, Perspt, CrtB, Centr, Rotx, Roty, Rotz, Unplot, Trans, Scale, DumpB, PltstB, HidsrB, ScanB, Obj_st. Then store on file "DEMO2". Use KEYS file "KeysTR" when using DEMO2.

09845-10080

The following subprograms must be present in addition to DRIV2C: One_obj_entry, Color_entry, Viewpoint, Plot, Multiply, Curve_setup, Circle_by_3_pts, Arc_gen, Viewcoord, Clip, Level, Close_poly, Perspective, Crt_tra, Center, Rotate_x, Rotate_y, Rotate_z, Unplot, Translate, Scale, Dump, Plotter_setup, Hidden_surface, Scan, and One_obj_store.

After getting DRIV2C, link the subprogram files: Obj_en, Cl_enC, VwptC, PlotC, Mult, Crvset, C_by_3, Arcgn, Viewcr, ClipC, Level, Close, Perspt, CrtC, Centr, Rotx, Roty, Rotz, Unplot, Trans, Scale, DumpC, PltstC, HidsrC, ScanC, Obj_st. Then store on file "DEMO2". Use KEYS file "KeysTR" when using DEMO2.

If you will be using the 9872 for a color hard copy of an object with hidden surfaces removed, put the pens into stalls one through four in the following order: black, red, green, blue.

Demo 2 User Instructions

1. To load in the demo and begin execution:
   a. Type: LOAD "DEMO2"
   b. Press: EXECUTE
   c. Press: RUN

2. When "ENTER FILE NAME: MASS STORAGE" is displayed:
   a. Enter: name of file to be transformed
   b. Press: CONT

3. If using 09845-10080, when "ENTER COLOR FILE NAME : MASS STORAGE" is displayed:
   a. Enter: name of corresponding color file
   b. Press: CONT

4. The special function keys and "INPUT X,Y,Z OF NEW VIEWPOINT (DEFAULT 300, -300,300)" will be displayed.
   a. Enter: values for new viewpoint
   b. Press: CONT

5. The object will be displayed from the viewpoint requested. (The point you choose for the viewpoint determines the line of sight from the viewer to the origin of the object's coordinate system. As a result, a "bad" viewpoint may prevent any portion of the object from being plotted.)

6. To translate the object:
   a. Press: K0 or K24
   b. When "ENTER X, Y AND Z INCREMENTS FOR TRANSLATING" is displayed:
      1) Enter: values for X,Y and Z
      2) Press: CONT

7. To rotate the object: (Rotation takes place about the approximate center. See Chapter 3 for more detail.)
   a. Press: K1 or K25
   b. When "ABOUT WHICH AXIS? ENTER X,Y OR Z" is displayed:
      1) Enter: axis you wish figure rotated about
      2) Press: CONT
   c. When "ENTER ROTATION ANGLE" is displayed:
      1) Enter: rotation in degrees (See the Section in chapter 3 on rotation).
      2) Press: CONT
   d. The object will be rotated.

8. To scale the object:
   a. Press: K2 or K26
   b. When "ENTER SCALE IN X DIRECTION" is displayed:
      1) Enter: X scale factor
      2) Press: CONT
   c. When "ENTER SCALE IN Y DIRECTION" is displayed:
      1) Enter: Y scale factor
      2) Press: CONT
   d. When "ENTER SCALE IN Z DIRECTION" is displayed:
      1) Enter: Z scale factor
      2) Press: CONT

NOTE

Curved surfaces must be scaled the same in X,Y, and Z direction since ellipses are not provided for in the data base.

---

9. To choose a new viewpoint:
   a. Press: K3 or K27
   b. When"CHOOSE VIEWPOINT: FRONT(1), END(2), TOP(3), ISOMETRIC(4), OTHER(5)" is displayed:
      1) Enter: the number corresponding to the viewpoint desired
      2) Press: CONT

NOTE

FRONT – Indicates you are looking down the negative Y-axis toward the origin. Unless the figure is centered about the origin, you will be viewing the figure from an angle.

END – Indicates you are looking down the positive X-axis toward the origin. Again, unless the figure is centered about the origin, you will be viewing the end (or side) of the figure at an angle.

TOP – Indicates you are looking down the positive Z-axis toward the origin. As before, unless the figure is centered around the origin, you will be observing it from an angle.

---

   c. If "OTHER" is chosen:
      1) When "INPUT X,Y,Z OF NEW VIEWPOINT (DEFAULT 300,-300,300) is displayed:
         a) Enter: values for new viewpoint
         b) Press: CONT

10. To plot the object on the CRT or another plotting device:
    a. Press: K4 or K28
    b. Answer questions about device to be plotted to.
    c. The object will be plotted.

11. To remove hidden surfaces or to return to wire-frame mode:
    a. Press: K5 or K29
    b. When "DO YOU WISH TO REMOVE HIDDEN SURFACES? (Y,N)" is displayed:
       1) Enter: desired response
       2) Press: CONT

    c. If you are using 09845-10080, answer prompts to enter the desired plotter. Plots on the 9872 will take four times as long as those on the CRT.

12. To dump the object to a printer:
    a. Press: K6 or K30
    b. Answer questions about device to be dumped to.
    c. The object will be dumped.

13. To store the transformed object:
    a. Press: K7 or K31
    b. Answer prompts to enter the file name.

Filename: DRIV2B (09845-10060)

Local Variable Definitions

Answer$[3]	Contains YES/NO response
File$[20]	Name of file to be used
Plotter$[20]	Plotter identification
Bus_address	Bus address for HP-IB device
Edge(0:250,5)	Linked list of pointers to vertices that comprise edges; also includes edge type
Hid	Equals 0 for no hidden-surface removal, 1 for hidden surfaces to be removed
N	Number used in dimensioning data base arrays
Object(0:125,3)	Linked list of pointers to edges that comprise objects
Old_hid	Contains last value of Hid
Pic	Pointer to head of list in Object(*)
Select_code	Select code of plotter
Surface(0:500,3)	Linked list of pointers to edges that comprise surfaces
Sur_tmp(0:500,2)	Temporary list of pointers to vertices that comprise surfaces; also contains pointers to color information
Trans	Equals 1 for rotation, 2 for translation, 3 for scaling
View	Equals 1 for front, 2 for side, 3 for top, 4 for isometric, 5 for user defined
Status(4,4)	Concatenated transformations
Transform(0:250,4)	Array of vertices multiplied by current Status array
Vertex(0:250,4)	Linked list of vertices
Xvp	X value of viewpoint
Yvp	Y value of viewpoint
Zvp	Z value of viewpoint

Filename: DRIV2C (09845-10080)

Local Variable Definitions

Answer$[3]	Contains YES/NO response
File$[20]	Name of file to be used
Plotter$[20]	Plotter identification
Bus_address	Bus address for HP-IB device
Edge(0:250,5)	Linked list of pointers to vertices that comprise edges; also includes edge type
Hid	Equals 0 for no hidden-surface removal, 1 for hidden surfaces to be removed
N	Number used in dimensioning data base arrays
Object(0:125,3)	Linked list of pointers to edges that comprise objects
Old_hid	Contains last value of Hid
Pic	Pointer to head of list in Object(*)
Select_code	Select code of plotter
Surface(0:500,3)	Linked list of pointers to edges that comprise surfaces
Sur_tmp(0:500,2)	Temporary list of pointers to vertices that comprise surfaces; also contains pointers to color information
Trans	Equals 1 for rotation, 2 for translation, 3 for scaling
View	Equals 1 for front, 2 for side, 3 for top, 4 for isometric, 5 for user defined
Color(125,3)	Color parameters for the current object
Status(4,4)	Concatenated transformations
Transform(0:250,4)	Array of vertices multiplied by current Status array
Vertex(0:250,4)	Linked list of vertices
Xvp	X value of viewpoint
Yvp	Y value of viewpoint
Zvp	Z value of viewpoint

Digitizer Entry

This program demonstrates the use of the digitizer in entering three-dimensional objects. An object is entered from a mechanical drawing that provides three orthogonal views. Each surface must be composed of line segments. Once the object is entered, it can be stored, drawn on a plotter, or dumped to a printer. Both the 09845-10060 and 09845-10080 packs require the I/O ROM for this demo.

09845-10060

The subprograms and binaries required are: Input, Initviewpln, Constviewpln, Display, Draw2d, Recvpcoords, Modvp, Convert3d, Indbad, Findsurf, Dump, Viewpoint, Plotter_setup, Plot, Multiply, Curve_setup, Circle_by_3_pts, Arc_gen, Viewcoord, Parse_response, Clip, Level, Close_poly, Perspective, One_obj_store, Dmpg#b, and Binary.

After loading the binaries, Dmpg#b and Binary, and getting DRIV3B, link the subprogram files: DginpB, DginiB, DgcvwB, Digdsp, Digdrw, Digrvc, DgmodB, DgcvtB, Digibd, Digfsr, DumpB, VwptB, PltstB, PlotB, Mult, Crvset, C_by_3, Arcgn, Viewcr, Parse, ClipB, Level, Close, Perspt and Obj_st. Store on file "DEMO3".

NOTE

Remember to replace all CALL Ginput statements in the subprograms Plotter_setup, Dump and Viewcoord with analogous LINPUT statements. For example:

```
CALL Ginput ("ENTER PRINTER SELECT CODE, "Select_
code$)
```

becomes

```
LINPUT "ENTER PRINTER SELECT CODE," Select_code$
```

09845-10080

The subprograms needed in memory are: Input, Initviewpln, Constviewpln, Display, Draw2d, Recvpcoords, Modvp, Convert3d, Indbad, Findsurf, Dump, Viewpoint, Plotter_setup, Plot, Multiply, Curve_setup, Circle_by_3_pts, Arc_gen, Viewcoord, Clip, Level,Close_poly, Perspective, and One_obj_store.

After getting DRIV3C link the subprogram files: DginpC, DginiC, DgcvwC, Digdsp, Digdrw, Digrvc, DgmodC, DgcvtC, Digibd, Digfsr, DumpC, VwptC, PltstC, PlotC, Mult, Crvset, C_by_3, Arcgn, Viewcr, ClipC, Level, Close, Perspt and Obj_st. Store on file "DEMO3".

Demo 3 User Instructions

1. To load in the demo and begin execution:
   a. Type: LOAD "DEMO3"
   b. Press: EXECUTE
   c. Press: RUN

2. When "ENTER DIGITIZER SELECT CODE AND BUS ADDRESS(706)" is displayed:
   a. Enter: select code and bus address. (Default is set to 706)
   b. Press: CONT

3. The data base will be initialized at this point. "INITIALIZING DATA BASE" is displayed on the CRT.
4. When "DIGITIZE TWO POINTS ON HORIZONTAL LINE OF DRAWING" is displayed:
   a. Using the digitizer (9874), digitize a point on any horizontal line of the drawing. (A BEEP will indicate the point has been digitized.)
   b. Digitize a second point on the horizontal line. (A BEEP will indicate the point has been digitized.)

NOTE

The greater the distance between the points, the more accurate your horizontal line will be.

---

5. When "DIGITIZE LOWER LEFT AND UPPER RIGHT CORNERS OF DRAWING" is displayed:
   a. Digitize the lower left corner of the rectangle enclosing the orthographic views. (Do NOT expect a BEEP at this point.) In figure 10.2 the point described is labeled LL.
   b. Digitize the upper right corner of the rectangle enclosing the orthographic views. (A BEEP will indicate that both points have been digitized.) This point is indicated by UR in figure 10.2.

6. When "DIGITIZE INTERSECTION OF CUTTING PLANES" is displayed:
   a. Digitize the point at which the horizontal line and vertical line that separate the three orthogonal views meet. (A BEEP will indicate the point has been digitized.) In figure 10.2, the point labeled CP designates the intersection of the cutting planes.

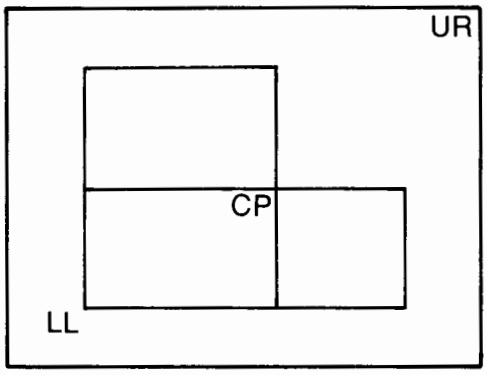


Figure 10.2: Standard "3-view" arrangement

7. IF an error has occurred during the digitizing of the boundaries, "ERROR MADE DURING ENTRY - PLEASE REENTER POINTS INDICATED" will appear on the CRT and the program will go to step 4.
8. When "Select function" is displayed by the digitizer, use the digitizer special function keys to choose desired action. You should create a key overlay for your convenience. The labels should be as follows:

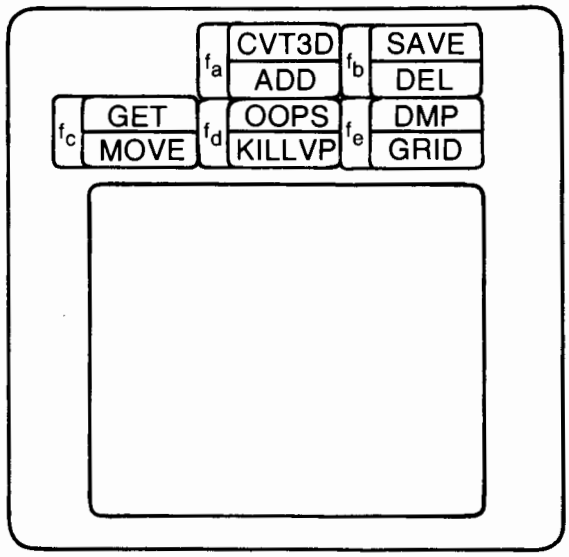


Figure 10.3: Digitizer Key Overlay

9. To add surfaces using orthographic views:
   a. Press: fa
   b. When "Add SURFACES" appears on the digitizer display:
      1) Position the puck in the desired plane and digitize vertices of a surface.(A beep will sound as each vertex point has been digitized. A high-pitched short beep denotes a point that has not been digitized before. A low-pitched short beep denotes a point that has been digitized before. A long low-pitched beep denotes a surface is being closed at that point.)

2) The program will stay in add surface mode until another digitizer special function key is pressed.
3) Go to 9b.1.

10. To delete surfaces:
    a. Press: fb
    b. When "DELETE SURFACES" appears on the digitizer display:
        1) Position the puck in the desired plane and digitize vertices of the surface to be deleted.
        2) The program will stay in delete surface mode until another digitizer special function key is pressed.
        3) Go to 10b.1.

11. To move vertices:
    a. Press: fc
    b. When "CHANGE POINTS" appears on the digitizer display:
        1) Position the puck on the vertex to be moved and digitize that point. (If the point you digitized has not been entered before, you will receive a low-pitched BEEP and NO action will be taken.)
        2) Position the puck at the new location for the vertex and digitize that point.
        3) Go to 11b.1.

12. To delete an entire view plane:
    a. Press: fd
    b. When "ENTER PLANE NO." appears:
        1) Enter: the number of the plane you wish to delete
        2) Press: ENTER
        3) Go to 8.

13. To change the error tolerance so as to make entering an object easier:
    a. Press: fe
    b. When "Grid IS <current error tolerance>" appears on the digitizer display:
        1) Enter: new tolerance desired (using digitizer keypad)
        2) Press: ENTER
        3) Go to 8.

14. To save the two-dimensional (three orthographic view) information:
    a. Press: sfb (PREFIX fb)
    b. "DO YOU WANT TO CHANGE MASS STORAGE DEVICE? (Y/N) will appear on the CRT.
        1) If your response is yes "SPECIFY MASS STORAGE DEVICE"

c. When "FILE no.? [Sa]" appears on the digitizer display:

1) Enter: the number you desire (Your file will be stored on the mass storage device as "P" followed by the number you entered. If you entered a 555, for example, a file named "P555" will be created on your storage device.) If you wish to save information on a file that has already been created, enter the negative of the file number when the file number is requested.

2) Press: ENTER

3) Go to 8.

15. To convert the three orthogonal views to a 3-D representation:

a. Press: sfa (PREFIX fa)

b. Vertex, Edge, and Surface information will be computed at this point.

c. If inconsistencies are found, enter a modification mode (fa,fb,fc,fd, or fe). Once modifications are made, go to 15a.

d. If no inconsistencies are found, "ENTER THE NUMBER OF SURFACES THE OBJECT HAS" will appear on the CRT.

1) ENTER: number of surfaces you expect for the object

e. If the number you enter does not agree with the number of surfaces generated by the program, "NUMBER OF SURFACES GENERATED ( )<> NUMBER OF SURFACES EXPECTED ( )" will appear on the CRT.

f. "DO YOU WISH TO DISPLAY THE 3-D OBJECT? (Y/N)" will appear on the CRT.

1) If your response is yes, the object will be displayed using a default viewpoint of (300,−300,300), no clipping and no hidden surface.

2) "DO YOU WANT TO DUMP THE GRAPHICS CRT? (Y/N" will be displayed next. Answer the questions that follow.

g. "DO YOU WANT TO STORE THE 3-D OBJECT? (Y/N) will appear on the CRT.

1) If your response is yes, then answer prompts to input file name when "ENTER FILE NAME:MASS STORAGE" appears on the CRT.

2) Go to 8.

h. "DO YOU WANT TO ENTER A DIFFERENT FIGURE? (Y/N)" will be displayed on the CRT.

1) If your response is "yes", the program exits so you can enter an entirely new object.

2) If your response is "no", you may clear the current viewplanes or leave them as is.

16. To get two-dimensional (three orthographic view) information that has been previously stored:
    a. Press: sfc (PREFIX fc)
    b. "DO YOU WANT TO CHANGE MASS STORAGE DEVICE? (Y/N)" will appear on the CRT.
        1) If your response is yes, "SPECIFY MASS STORAGE DEVICE" will appear.
    c. When "FILE no.? [get]" appears on the digitizer display:
        1) Enter: the number you desire
        2) Press: ENTER
        3) Go to 8.
17. To delete all vertices of the current surface being entered:
    a. Press: sfd (PREFIX fd)
    b. Continue in the mode in which you were currently working
18. To dump the current graphics display:
    a. Press: sfe (PREFIX fe)
    b. Go to 8.

Filename: DRIV3B (09845-10060), DRIV3C (09845-10080)

COMMOM Variable Definitions

Vertex(0:100,4)	Linked list of all edge-defining vertices
Edge(0:100,5)	Linked list of pointers to vertices that comprise edges; also includes edge type
Surface(0:200,3)	Linked list of pointers to edges that comprise surfaces
Object(0:50,3)	Linked list of pointers to surfaces that comprise objects
Vertex2d(3,0:100,5)	Contains vertex information entered from three orthogonal views
Edge2d(3,0:100,5)	Contains edge information entered from three orthogonal views
Surf2d(3,0:200,3)	Contains surface information entered from three orthogonal views
Object2d(3,0:50,3)	Contains pointers to surfaces for object entered from three orthogonal views
Xcp	X value of cutting plane intersection of mechanical drawing
Ycp	Y value of cutting plane intersection of mechanical drawing
Digiratio	Ratio of the width to the height of the area of the digitizer; defined by lower left and upper right corners of input document

Plotwidth	Plotter width in GDU's
Plotheight	Plotter height in GDU's
Planeratio	Ratio of height of plane 1 to height of plane 2
Grid	Number of User Units by which at least one of the coordinates of 2 vertices must differ in order for them to be considered distinct vertices
Possedg(0:10,30)	Array containing all possible 3-D edges that are represented by the 2-D edges of a 2-D surface
Tempsurf(30)	Temporary storage for surface information

Filename: DRIV3B (09845-10060), DRIV3C (09845-10080)

Local Variable Definitions

Digi	Digitizer select code and bus address
Mneps	Maximum number of edges allowed per surface
Ne	Number of edges allowed
Ne2	Number of 2-D edges allowed
No	Number of objects allowed
No2	Number of 2-D objects allowed
Ns	Number of surface elements allowed
Ns2	Number of 2-D surface elements allowed
Nv	Number of vertices allowed
Nv2	Number of 2-D vertices allowed
Redundfctr	Maximum number of 2-D or 3-D edges that may be represented by a single 2-D edge

Handling Errors Detected

NOTE

Before attempting error correction, check to see what the error tolerance is. The default error tolerance is 2. Setting the tolerance to a larger number will eliminate most errors caused by careless digitizing.

When converting orthographic views to 3-D object, there are three phases in the conversion process. These are:

a. computation of 3-D vertices
b. computation of 3-D edges
c. computation of 3-D surfaces

Any of these phases may find inconsistencies in the orthographic views that have been entered. Such errors will cause the three views to be redrawn with dashed lines indicating the nature and location of the error or errors. All errors found during a phase will be designated before control is returned to the digitizer keypad. Note that the next phase in the sequence will be entered only if the last phase was completed without detecting any inconsistencies.

The following is an example of what will result if errors are found in the first phase.(computation of 3-D vertices)

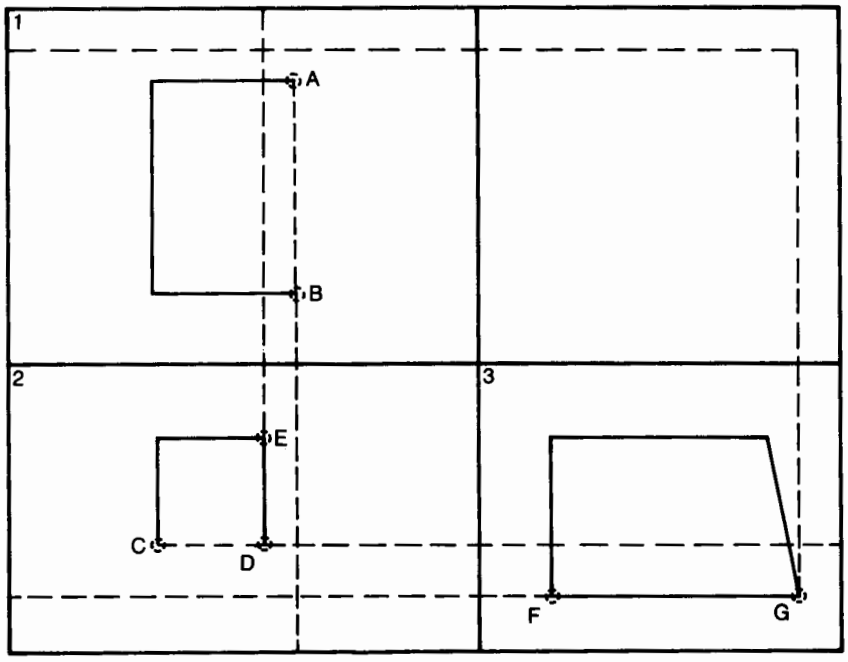


Figure 10.4: Lines indicating inconsistent vertices

The dashed lines indicate where the vertex conversion algorithm has found inconsistencies in the input data. The vertices in question are encircled. The circles enclosing these vertices have a radius equal to the defined grid spacing.(This grid spacing can be altered using PREFIX fe. Refer to the User Instructions for DEMO3.)

This feedback alone is of little use unless you have some knowledge of the algorithm used. Basicly, each vertex is projected into the other two view planes in order to determine its representation in each of them. The vertices are projected as follows:

1. FROM plane1 TO plane 2 TO plane 3
2. FROM plane2 TO plane 3 TO plane 1
3. FROM plane3 TO plane 1 TO plane 2

In the example shown above, you see that points A and B of plane 1 have no corresponding points in plane 2 when they are projected. Points C and D have no corresponding points when projected into plane 3. Point E, however, has a corresponding point in plane 2 but not in plane 1. Point F has no corresponding point in plane 2. Point G has no corresponding point in either plane 2 or plane 1.

Below is an example of what you can expect if inconsistencies are found during phase two.(computation of 3-D edges)

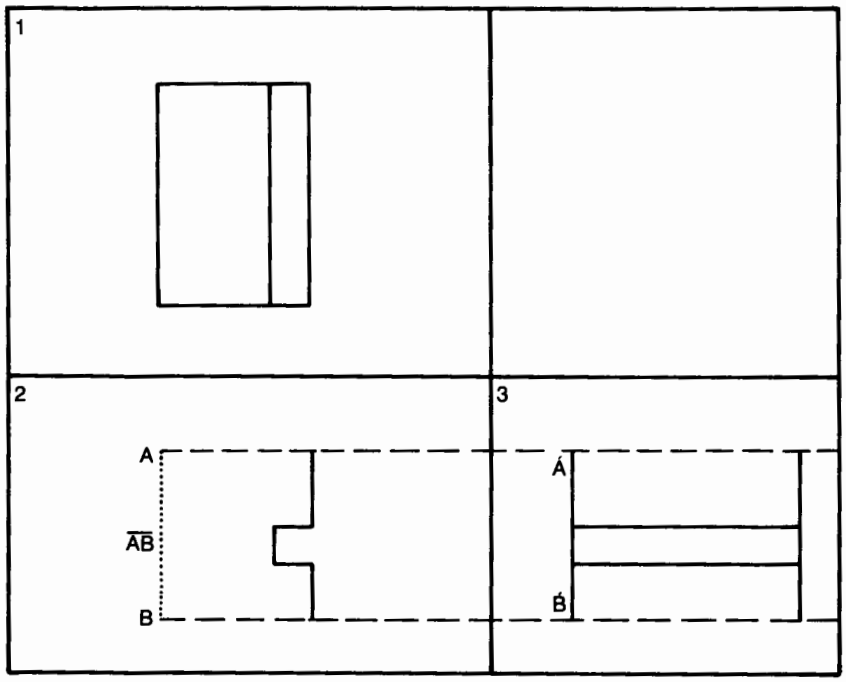


Figure 10.5: Lines indicating inconsistent edges

Observe, if an edge is indicated by a dotted segment, it means that the edge so indicated connects two vertices that are not connected in one of the other view planes. Edge $\overline{AB}$, for example, indicates a connection between points A and B in view plane 2. Points A and B in plane 2 correspond to points A′ and B′ in view plane 3. The dashed lines indicate that the points A′ and B′ were never connected as an edge in view plane 3. (For example, assume you entered only three surfaces in this third view plane - the top, middle, and bottom rectangular surfaces.)

As with vertices, the projection sequence is:

1. FROM plane1 TO plane2 TO plane3
2. FROM plane2 TO plane3 TO plane1
3. FROM plane3 TO plane1 TO plane2

Lastly, during phase three, it is possible to detect a surface represented in one of the view planes for which a collection of 3-D edges that form a closed path cannot be found. If such a 3-D surface is not found for a 2-D surface of a view, that 2-D surface is considered inconsistent and is indicated by redrawing that surface using broken line segments.

Surface Color Entry

09845-10080

This driver allows you to take a previously entered object and add color information to its surfaces. The color parameters are then stored on a data file to be used with the 9845C version of the Object Transformation Demo. There are no special function keys used. The subprograms that need to be in memory are: One_obj_entry, Multiply, Relist (9845C version) and Store_color. After getting DRIV4C link the subprogram files Obj_en, Mult, RlistC and C1_stC. Store the program on file "DEMO4".

NOTE

An object's color file must be reentered every time an object's surface information is altered.

Demo 4 User Instructions

1. To load in the demo and begin execution:
   a. Type: LOAD "DEMO4"
   b. Press: EXECUTE
   c. Press: RUN
2. When "ENTER FILE NAME MASS STORAGE" is displayed:
   a. Enter: file name of 3-D object
   b. Press: CONT
   c. If the file cannot be found, the program will return to 2.
3. The name of the demo and column headings will be displayed on the CRT.
4. The coordinates and connecting information for the objects first surface will be displayed on the CRT.
5. When Hue, Saturation and Luminosity FOR SURFACE 1 = ?" is displayed:
   a. Enter: the hue, saturation and luminosity values for surface 1
   b. Press: CONT
   c. If you enter a value outside the range 0 to 1, the program will return to 5.
6. The program will repeat steps 4 and 5 until all surfaces have been assigned color parameters.
7. When "ENTER NAME OF COLOR FILE: MASS STORAGE" is displayed:
   a. Enter: file name for color information
   b. Press: CONT
   c. If the file already exists, "(file) ALREADY EXISTS. USE IT ANYWAY?" will be displayed.
      1) To use the same file name:
         a) Enter: YES
         b) Press: CONT
      2) To use a different file name:
         a) Enter: NO
         b) Press: CONT
         c) The program will return to step 7.
8. The color information will be stored.

Local Variable Definitions for DRIV4C

File$[12]	Object's file name
Edge(0:250,5)	Array containing edge connecting information
Flag	Set to 1 before calling Relist to indicate color entry
N	Number used in dimensioning data arrays
Object	Pointer to last element used in Object array
Object(0:50,3)	Linked list of pointers to surfaces that comprise an object
Pic	Pointer to head of Object(*)'s linked list.
Surface (0:500,3)	Linked list of pointers to edges that comprise surfaces
Surface_count	Actual number of surfaces in an object
Color (12.5,3)	Array containing color parameters for AREA COLOR statement
Offset(3)	Absolute values of X, Y and Z axes minimum values.
Range(3)	Ranges of X, Y and Z values
Status (4,4)	Matrix of concatenated transformations
Transform(0:250,4)	Array of vertices multiplied by current Status array
Vertex(0:250,4)	Linked list of all edge-defining vertices

Programming Tips

Before beginning to program, it would be wise to study the subprogram definitions and how they interact via the four demo programs. The demos are not included to provide a "turnkey" solution to any application, but they do provide an example of the correct calling sequence needed to perform a certain action, such as rotation or clipping. The following paragraphs include tips on 1) memory size, 2) mass storage, 3) key definitions, 4) preparing data to be stored, 5) using the 9874 Digitizer and 6) using advanced shading techniques with the 9845C.

With a memory of 187K bytes, you can have a fairly large program and data base in memory at one time. For example, "DEMO1" contains about 1300 lines of code, and has a data base of 250 vertices and edges. Keep in mind however, the larger the data base, the longer it takes to do pictorial processing. You can change the number of vertices for DEMO1 and DEMO2 by altering the array declaration statements in the demo drivers and setting the value of the variable N to the desired number of vertices. For DEMO 3, you must change the COMMON variable declarations and appropriate variables.

It is easy to see how an application program could become very large. Just considering the demo programs, DEMO2 takes up 377 records and DEMO3 takes 444 on a mass storage device. Because of tape access time, it is suggested you strongly consider using a flexible disc. Also, remember to use the comment stripper utility to remove unnecessary remarks.

There are two ways to define the special function keys. The first alternative is that of using a keys file. This provides immediate interrupts only while in INPUT and PAUSE statements. However, there is a fault with this method of key definition on the 9845B. If you are in graphics mode, pressing a special function key will cause CRT to momentarily blink out of graphics mode. The reason being that since the keys are also typing aids, the definition of the key will be momentarily displayed in the alpha mode. The second alternative is that of defining the special function keys with the ON KEY# statement in the program. This way will not blink you out and back into graphics mode, but has other drawbacks. The keys are not recognized in an INPUT or PAUSE statement, and are only defined in their own environment. This means a key defined this way in the main program is undefined in any subprogram.

It is advisable to display special function keys on the CRT when possible. This alleviates the problem of lost overlays. In the demo programs for the 9845B, special function key definitions are "LABELED" or "GPRINTED" onto the graphics raster, and "PRINTED" onto the alpha raster. The 9845C, however, has the convenient command "LABEL KEYS" that displays key definitions along the bottom of the CRT.

In most cases, when an (X,Y,Z) coordinate is entered, it is not in the form it will be stored. This pack uses two arrays, Range(*) and Offset(*), to convert from user units to the range $-100,100$ for all three X, Y and Z axes. This is to insure all coordinates are in the same range for scaling purposes of the active plotter. In the digitizer entry program, this range is set up as the default. If the coordinates are entered manually, this conversion must be done programmatically. Also, since this pack allows transformations of objects, you must apply the inverse of its Status(*) to a point before storing it. If this multiplication was not performed, it would be difficult to edit an object after transforming it.

The 9845B and 9845C may appear to interact the same with the 9874 Digitizer to a user, but the programming techniques are different. The 9845B views the digitizer as another plotter and most interactiveness was programmed through the I/O ROM. The 9845C sees the digitizer as a "GRAPHICS INPUT DEVICE" and uses new commands to talk to it as such.

The first difference comes when trying to moniter the digitizer keys. The 9845B merely sits in a loop waiting for the status word to change. It is then a simple matter of checking bit 2 for a digitized point or bit 7 for a pressed key, to see what was pressed. The 9845C, however, clears out the pressed digitizer button or key bit as soon as its pressed. This is due to the new "ON GKEY" command. Therefore, it is necessary to set up on "ON GKEY" interrupt that branches to another routine. Here the "CURSOR X,Y, Status$" command is executed, then Status$ is analyzed to determine what key was last pressed.

The second difference is not quite so complicated. In order to get the marker of your CRT to follow the movements of the cursor of the 9874, the 9845B stays in a small loop of four statements. After the digitizer is declared on, the CURSOR statement interrogates the input device to find the X,Y location of the cursor. Then the CRT is declared on and the POINTER statement is executed to position the CRT's marker to the corresponding X,Y location. The 9845C has the command "TRACK <plotter select code:> IS ON/OFF" which sets this "Track" mode for you.

The last major difference is that the 9845C may have multiple plotters on at one time. When programming with the 9845B it is sufficient to just turn the current plotter on, and the previously active plotter would be turned off. For the 9845C you must be sure to programmatically turn plotters off.

Instead of assigning each surface of an object its own color, you may wish to generate a more realistic picture. Normally an object is the same color all over. One way you can distinguish separate surfaces is by varying the intensity of each surface by the amount of light it would receive. If you pick the light source to be the same as your viewpoint, this calculation can be greatly simplified.

This computation would have to be done in the subprogram Hidden_surface before Scan is called. In order to assign each surface by its average intensity, you simply need to calculate $(COS\theta)^2$, where θ is the angle between the light source vector and the normal vector to the surface. If this value is normalized from 0 to 1, it may be used directly as the third parameter in the AREA COLOR statement.

Glossary

clipping	a technique used to select those parts of a picture that lie on the screen or in cases of three dimensions, within the viewing pyramid
coordinate system	a method of addressing points in a plane or in space
concatenation of matrix transformations	the product of two or more matrix transformations resulting in one matrix that represents a sequence of transformations
digitizer	a device used to enter coodinate data
edge	as used in this pack, an edge can be a line segment, a circle, or an arc
eye coordinate system	has its origin fixed at the viewpoint and its Z-axis pointed in the direction of the view
hidden-line removal	removing lines of an object (scene) that are hidden from view by opaque objects
hidden-surface removal	removing parts of an object that are invisible because they face away from the viewer or are obscured by other parts of the object
homogeneous coordinate	a scale factor used as the fourth component of a vector describing a point in three-dimensional space
homogeneous point	a vector [X,Y,Z,W] that represents a point (X,Y,Z)
matrix	an M by N array of elements
object coordinate system (world coordinate system)	the coordinate system in which an object is defined
perspective	conveys depth information by making distant object smaller and foreshortening objects that are near
raster-scan display	the display is determined by the intensity of each dot in a rectangular matrix of dots that covers the entire screen
rotation	the movement of an object about a fixed point
scaling	increasing or diminishing the size of an object
scan line algorithm	involves using a test line (scan line) to determine all pixels (dots) of a polygon that need to be turned on for a raster display
screen coordinate system	the coordinate system made use of for the display on the screen
surface	as used in this pack, a surface can be a closed polygon made up of line segments, a circle, or an arc with one or more line or arc segments used to close the surface
vertex	a point where two or more edges meet
viewbox	a truncated viewing pyramid mapped into the clipping coordinate system that describes the region in which an object must lie in order to be visible

viewing pyramid	designates a region of the eye coordinate system in which an object must lie in order to be visible
viewpoint	the origin of the eye coordinate system
viewport	a rectangle on the display screen where the contents of the "window" is to be displayed
world coordinate system (object coordinate system)	the coordinate system in which an object is defined
window	a rectangle of the object coordinate system that is of interest to the viewer
windowing	specifying a rectangle that surrounds the information desired in the display
wire-frame	a diplay technique that shows all edges of an object whether hidden to the viewer or not

Bibliography

1 Eastman, J.F.: "An Efficient Scan Conversion and Hidden Surface Removal Algorithm", Computer Graphics, Vol. 1, pp. 215-220, 1975.

2 Gilio, W.K.: Interactive Computer Graphics, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

3 Newman, W.M. and R.F. Sproull: Principles of Interactive Graphics, Second Edition, McGraw-Hill, New York, 1979.

4 Sutherland, I.E. and G.W. Hodgman: Reentrant Polygon Clipping, "CACM Communication of ACM, Vol. 17, January, 1974

5 Gusecke, Mitchell, Spencer, Hill: Technical Drawings, Sixth Edition, Macmillan Publishing Co. Inc., New York, 1974.

Part No. 09845-10061
Rev. C

Printed in U.S.A.
Third Edition (Aug. 1981)