THREE DIMENSIONAL MICROCOMPUTER GRAPHICS

BASIC VERSION

by

Bruce A. Artwick

August 1977

SubLogic Company
P.O. Box 3442
Culver City, California 90230

First Edition
First Printing

Printed in U.S.A.

Serial Number _____

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

The use of computer graphics in science, engineering, art and many other fields has been increasing over the past fifteen years. Display device costs continue to drop as many manufacturers enter the display systems market with new display hardware and software products. Software packages which allow the user to project three dimensional scenes however are not very common, and programs which allow a viewer to observe a three dimensional scene from any location and angle are nearly nonexistent. This can be attributed to the fact that 3D graphics programs are very complex, rely heavily on complicated mathematical theories and computations and are very hard to write, debug and test. In addition, intricate software/hardware interactions are involved. Very few people simultaneously know enough about the mathematics of computer graphics, hardware, software design, and now microprocessors, to effectively write 3D graphics software. SubLogic has therefore put great effort into developing a simple-to-use 3D graphics package for microprocessor based systems. SubLogic software and hardware products take 3D graphics out of the university and industrial environments and makes 3D graphics available to everyone.

One of the biggest problems in developing any graphics software is thoroughly testing it. A 3D graphics program is

actually an implementation of a large, conditional mathematical expression. A 3 dimensional space with a range of + or - 32000 units in the X,Y, and Z directions has 2.6 x 10^{14} (64000 cubed) possible coordinate points. There are therefore 6.9 x 10^{28} possible straight lines. With such an enormous number of program input combinations, combinatorial program testing is out of the question. To insure that the program is reliable (the equation is stable), program segments are combinatorially tested. Continuous development and testing of 3D graphics programs at SubLogic promises to increase program performance and reliability.

Two SubLogic graphics packages currently exist: a general purpose, low speed BASIC language version and an optimized 6800 assembly language version. Both programs perform the task of converting an array of straight "three dimensional space lines" into an array of "two dimensional screen lines". Figure 1 gives the performance characteristics of the SubLogic 3D Microcomputer Graphics Packages. Other 3D graphics packages are being developed. 8080, 6502 and Z80 versions are set for introduction in the first quarter of 1978.

package characteristics	sublogic 3D microcomputer graphics packages	
	universal basic version	6800 assembly language version
Program Number	3DGBU.V03	3DG68.V03
Program Language	Minimal Set BASIC (SWTP 4K or equiv.)	6800 Optimized Assembly Language
Projection Method	3D to 2D wire frame perspective transformation with 3D clipping	3D to 2D wire frame perspective transformation with 3D clipping
Viewing Capabilities	X,Y,Z range: floating pt. range of BASIC. 3 axis freedom: 0 to 359 degrees	X,Y,Z range: ± 32767 units 3 axis freedom: 0 to 359 degrees
"World" Size	Floating point range of BASIC	1912 cubic miles using one foot units
Special Features	Variable viewing window (telephoto to wide angle) Universal machine compatibility due to the use of BASIC	Variable viewing window Optimized clipping High rate dynamic capabilities
Package Contents	3D to 2D space to screen converter subroutine Listing in BASIC Usage information Application notes Interfacing information Test procedures Algorithm description	3D to 2D space to screen converter subroutine Object listing Mikbug/Kansas cassette Usage information Application notes Interfacing information Test procedures Algorithm Description
Cost	$ 22.00	$ 28.00

Figure 1. SubLogic 3D Graphics Package characteristics

THE 3D TO 2D CONVERTER CONCEPT

Most 3D graphics users are primarily interested in putting 3D graphics to use in their own special application. To these users, the process used to perform the transformations and projections is considered to be of secondary importance. The SubLogic 3D to 2D converter subroutine (the heart of the 3D Microcomputer Graphics Package) was therefore designed to be very easy to use without any graphics programming knowledge or experience. The user simply sets up an input array (an array of 3D lines in a pre-set format) in processor memory and executes the 3D to 2D conversion subroutine. The converter transforms the 3D scene into a 2D screen image. A device dependent interface program then sends the scene to the display device.

Eight BASIC variables are used to store 3D to 2D converter subroutine control information such as viewer's position, direction of view and screen width. Figure 2 interrelates the input array, 3D to 2D converter and interface program.

Appendix 2 describes how the 3D to 2D converter subroutine performs its task. The rest of this section is devoted to the details of how to use the 3D to 2D converter.

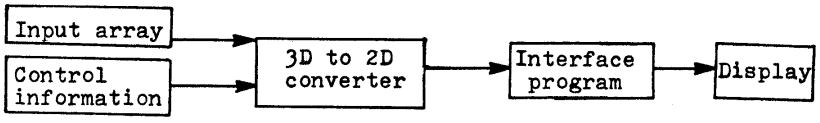
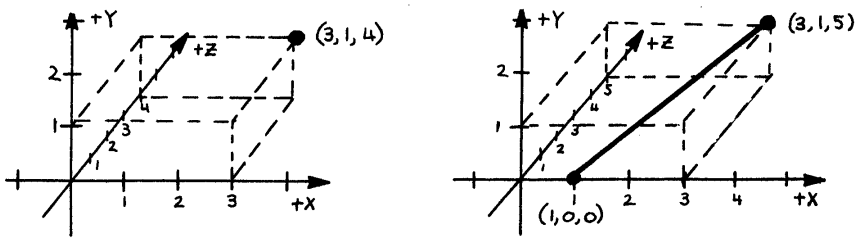


Figure 2. The 3D to 2D converter program and its arrays

SPACE AND SCREEN COORDINATES

The first concept which must be well understood is that of space and screen coordinates. It should be noted that the word "screen" used throughout this discussion also implies plotters and other display devices.

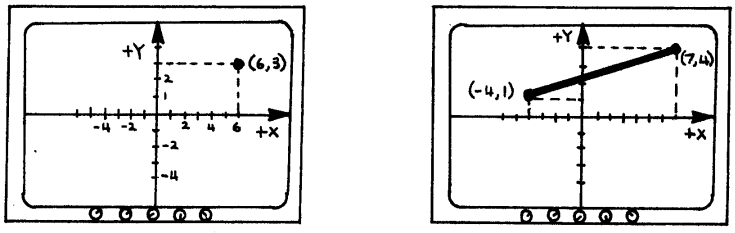
3D SPACE COORDINATES. Every point in 3 dimensional space has an X,Y,Z space coordinate associated with it as figure 3a shows. A straight line is represented by its start and end points as shown in figure 3b.



a) A point in 3D space b) A line in 3D space

Figure 3. Three dimensional space coordinates

2D SCREEN COORDINATES. Every point appearing on the screen has a 2D screen coordinate associated with it. Figure 4a shows a point on a screen, and a screen line is represented by a screen start and end point as shown in figure 4b.



a) A screen point b) A screen line

Figure 4. Two dimensional screen coordinates

3D SPACE AND 2D SCREEN COORDINATE RELATIONSHIPS

The X,Y,Z space coordinate axes directions were chosen to correspond in a graph axis fashion with the screen coordinates. As figure 5 shows, the X and Y space coordinate axes viewed through a screen match the X and Y screen axes with the Z axis representing "depth into the screen". This X,Y axis match-up applies when the viewer's viewing direction is 0 degrees pitch, 0 degrees heading and 0 degrees bank.

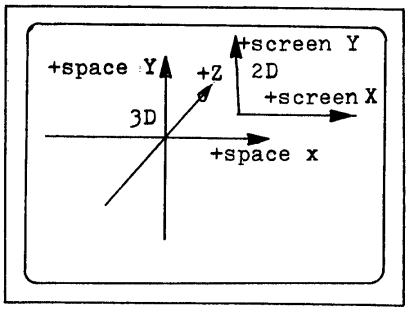
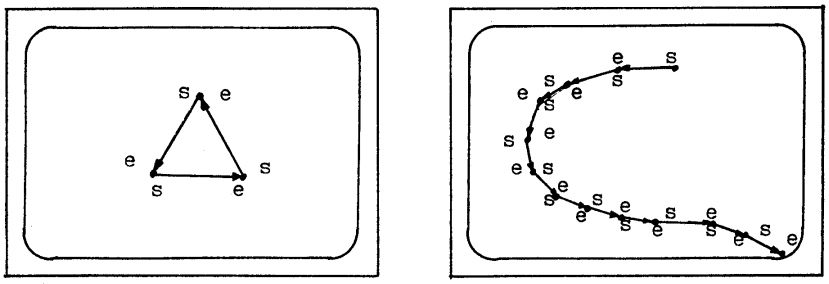


Figure 5. 3D and 2D coordinate alignment

OBJECT CONSTRUCTION IN SPACE COORDINATES

The SubLogic 3D to 2D converter converts 3D space coordinates to corresponding 2D screen coordinates. Straight lines in space are represented by two points in space; a start point and an end point. Wire frame objects and outlines can be constructed using many straight lines as shown in figure 6. Curves can be represented by a long string of short, straight

lines. Very realistic curves can be generated using large numbers of line segments but projection speed suffers as each line must be projected seperately.



s= start point
e= end point

a) Line drawing

b) Curve construction

Figure 6. Object construction in 3D space

OBSERVING 3D SPACE ON A DISPLAY DEVICE

A few 3D to 2D converter control parameters are needed before a 3D to 2D conversion can be performed. The location and direction from which one wants to view the 3D scene are needed. First the viewer's location in space must be specified. Figure 7 illustrates what is meant by viewer's location. An X,Y,Z viewer location must be submitted as the BASIC variables X(3), Y(3), and Z(3). Movement of the viewer's location is called translation.

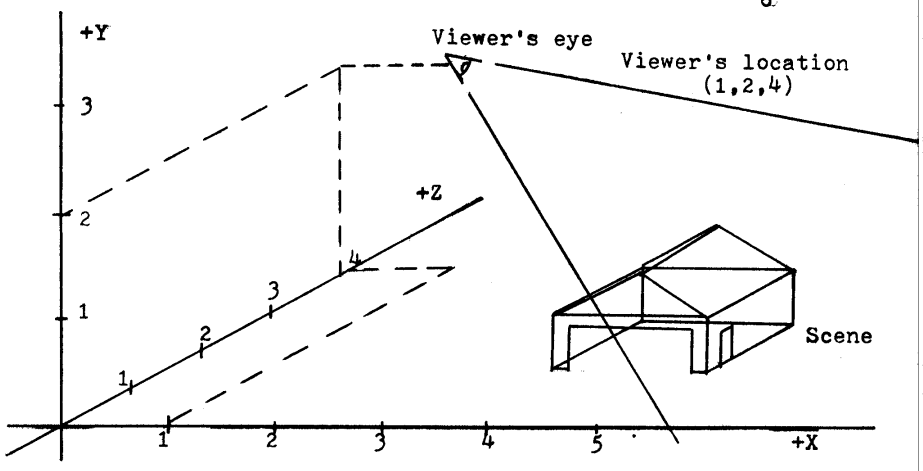


Figure 7. Viewer's location in space

Viewer's direction must also be specified as figure 8 illustrates. A pitch, bank and heading must be specified. Change in the viewer's direction is called rotation.

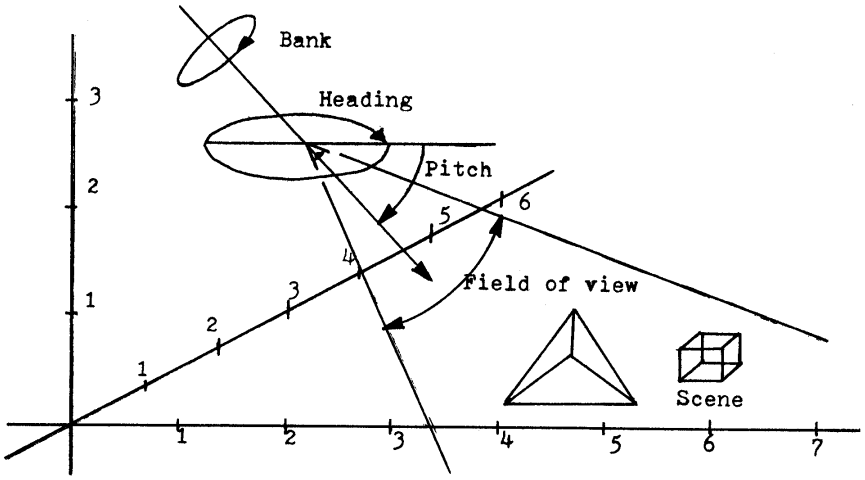
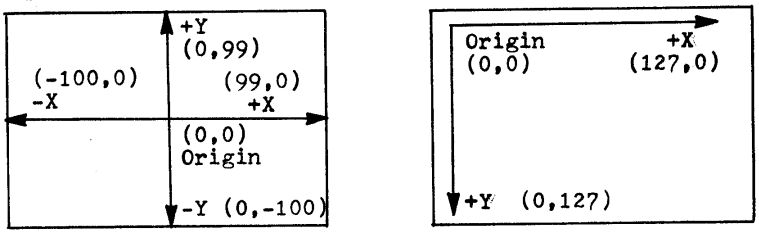


Figure 8. Viewer's direction and field of view

The field of view must also be specified. This parameter is similar to a camera's field of view as figure 8 shows. Only a limited field of view can fit onto the viewing screen and the viewer must decide whether a wide angle or telephoto view is desired.

The 3D to 2D converter subroutine generates 2D screen start and end points which represent lines to be plotted on a display device. Screen coordinate numbering systems vary widely between different display devices. An example of this is shown in figure 9.



a) A 200 x 200, origin at center format

b) A 128 x 128, origin at upper left format

Figure 9. Variations in screen coordinates

The 3D to 2D converter subroutine needs a screen width parameter to enable the generation of user device compatible coordinates. Regardless of screen width, the screen origin (screen coordinate x=0,y=0) is assumed to be at the screen center. Many display deviced, however, do not have the origin in the center of the screen. Origins in the upper left hand screen corner (see figure 10) are very popular. For this type of display, an array of output points in the

origin-at-center format can easily be transformed by adding a constant value to each X and Y point. A seperate, user written subroutine is required to do this.

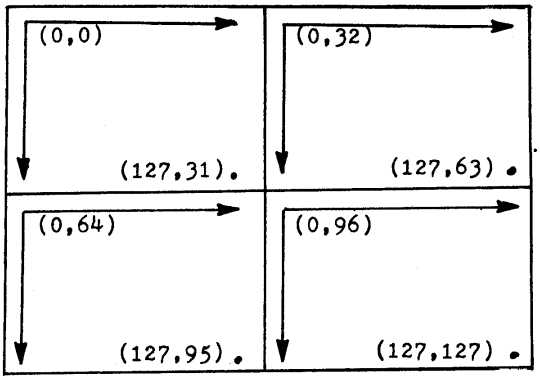


Figure 10. A popular display device's unusual screen addressing scheme

ORDER OF TRANSFORMATIONS

The order in which transformations are considered is of prime importance. The image projected on the screen will be different if different orders of translation and rotation are applied. For example, if the viewer's location in space (translation) is considered before his viewing direction (rotation), a different projection than if location had been considered after direction would result. Figure 11 shows these two orders of projection.

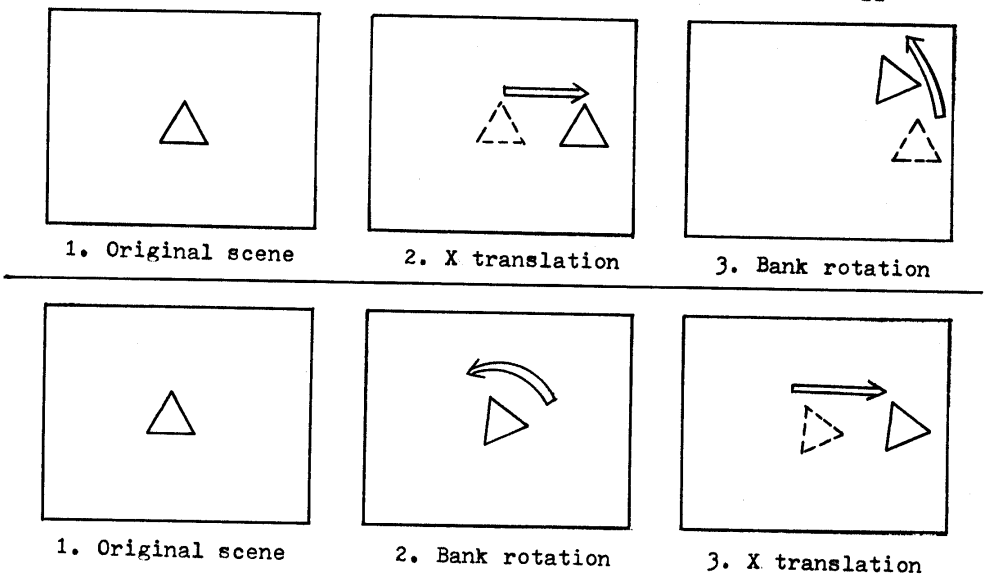


Figure 11. The difference transformation order makes

The SubLogic graphics package performs transformations in the following order:

1. X,Y,Z translation
2. Heading (rotation about the Y axis)
3. Pitch (the angle of view to the X,Z plane)
4. Bank

Figure 12 shows the sense of direction of each of the transforms. It should be noted that the transform senses are dependent on one another. A positive change in X will cause an object to move to the right if the viewer is at a 0 degree bank angle. If the viewer is in a 90 degree bank however, the cube will appear to move up instead.

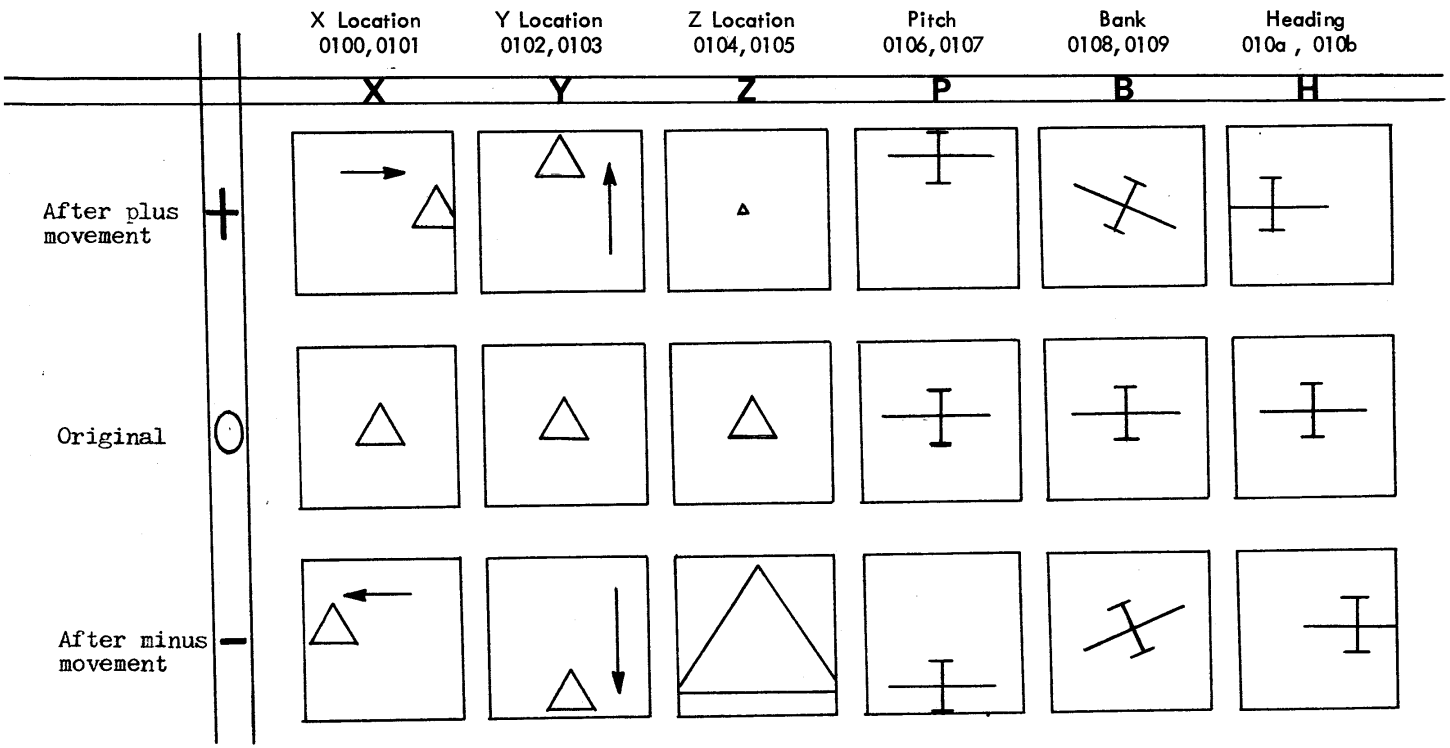


Figure 12. Direction of movement conventions

DETAILED CONTROL PARAMETER DEFINITIONS

A few of the eight control parameters have already been mentioned. This detailed definition section will describe them all. The program's feedback parameter will also be defined.

X(3), Y(3), Z(3)- - - These three floating point BASIC variables are the viewer's X, Y and Z location in space. X and Z movement usually represent north and west movement while Y is the viewer's altitude.

P - - - This is a floating point variable specifying viewer's pitch; the angle of inclination from which he looks at the scene.

B - - - A floating point variable representing viewer's bank; the angle at which a viewer's head is tilted sideways when viewing the scene.

H - - - Floating point variable representing heading. Heading is the direction the viewer is facing (North for example) while standing on the XZ plane.

V - - - Floating point value representing the tangent of the half field of view. V=1 represents a 45 degree half field or 90 degree full field of view (an unnatural-looking wide angle) while .3 represents a narrow telephoto view.

W - - - This floating point value represents half the screen width minus one. If a screen is 128 dots wide, this value should be (128/2) -1 = 63. This variable affects the scaling of the final output screen points. If W=63, the four corners of the screen will be (63,63), (-63,63), (63,-63),(-63,-63). This format is designed for screens with the origin at the center of the screen as described earlier.

P2 - - - Feedback Parameter - - - This parameter is not submitted by the user. It is generated by the program. P2 is set to 1 if the 3D space line just processed by the 3D to 2D converter is visible and on the screen. P2 is cleared to zero if the line is off the screen. P2 should be used in the display driving program to decide which lines to send to the display device.

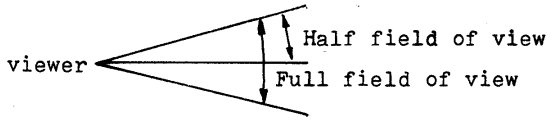


Figure 13. Field of view definition

USING THE 3DGBU.V03 3D GRAPHICS PROGRAM

The BASIC 3D graphics program will now be presented. Descriptions of all the subroutines will be given followed by three test and interface programs which take the user from simple "terminal type-in" formats to array handling and putting a scene on a display device.

The best way to approach the following sections is to first, carefully read over "The Basic Program" section and read the subroutine description sheets. Next, read the Test 1 section, fire up your computer system, and run the test program following the step-by-step procedure. Test 1 is very helpful in familiarizing the user with the program variables. Experiment with it. Array handling can then be tried and finally a display driving program can be implemented. This will require some programming on the user's part but examples and interface requirements are given.

Advanced methods of display system control and array handling are covered in the appendix sections.

THE BASIC PROGRAM

The BASIC 3D to 2D conversion program 3DGBU.V03 is capable of performing slow but extremely accurate 2D perspective projections of 3D space. In addition, the range of movement and size of "the world" can be very large due to BASIC's 9.9×10^{99} (depending on the BASIC version) range. No integer overflow problems, which exist on the assembly language versions, exist in the BASIC program.

The BASIC program consists of 3 parts:

1. The 3D to 2D converter subroutine
2. The transformation matrix generator
3. The interface program

The 3D to 2D Converter Subroutine

The 3D to 2D converter takes a single 3D line consisting of two 3D space coordinate points, X(1),Y(1),Z(1), X(5),Y(5),Z(5), and converts it into a 2D screen line consisting of two 2D screen coordinate points. The projection flag, P2, is set to one if the line is to be displayed, and is set to zero if the line is off the screen and not to be displayed.

The 3D to 2D converter consists of the Matrix Multiplier, Clipping and Projection sections shown in figures 14-17. The 3D to 2D converter subroutine's call is, "GOSUB 8500".

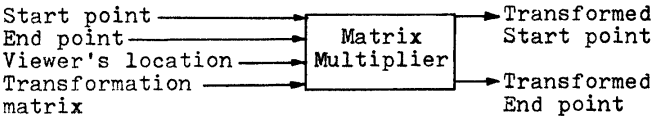
Title: Matrix Multiplier 8500

Purpose: This inline program section adds the viewer's location to the 3D start and end points and multiplies them by the matrix (T matrix) resulting in rotation.

Input: x(1),y(1),z(1),x(3),y(3),z(3),x(5),y(5),z(5), T1,T2,T3,T4,T5,T6,T7,T8,T9

Output: x(1),y(1),z(1),x(5),y(5),z(5) transformed

Operation:
The viewer's location values are added to the start and end points by executing the for-next loop twice. The points are also multiplied by the transformation matrix which is expanded into a three equation form. Execution goes on to the 8600 (Clipping) section when matrix multiplication is finished.



Subroutines Called:
None

Temporary Storage:
G,K,S

```
8500 FOR A=1 TO 5 STEP 4
8510  G=X(A)+X(3)
8520  S=Y(A)+Y(3)
8530  K=Z(A)+Z(3)
8540  X(A)=G*T1+S*T4+K*T7
8550  Y(A)=G*T2+S*T5+K*T8
8560  Z(A)=G*T3+S*T6+K*T9
8570 NEXT A
```

Translation offsets added

Rotation matrix multiplied

Translate and rotate the start and the end point (loop twice)

Figure 14. The Matrix Multiplier section

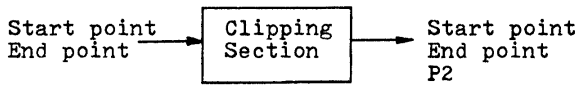
Title: Clipping Section 8600

Purpose: This section of inline code determines if a line is on or off the screen and accordingly displays, clips and displays, or eliminates it.

Input: x(1),y(1),z(1),x(5),y(5),z(5)

Output: x(1),y(1),z(1),x(5),y(5),z(5),P2

Operation:
This section first generates a code for the start and end point based on where they are in relation to the viewing pyramid. From these codes, the decision to clip, project or eliminate is made. A seperate section (the 8700 block) decides which way to clip a line if clipping is called for and performs the appropriate "push" mathematics. A projection code is set to 1 if a line is on the screen and cleared to 0 if it is not.



Subroutines Called:
None

Temporary Storage:
A,C(1-8),S

```
8600 FOR A=1 TO 5 STEP 4
8610  C(A)=0
8612  C(A+1)=0
8614  C(A+2)=0
8616  C(A+3)=0
8618 IF X(A) < -Z(A) THEN C(A)=1
8620 IF X(A) > Z(A) THEN C(A+1)=1
8622 IF Y(A) < -Z(A) THEN C(A+2)=1
8624 IF Y(A) > Z(A) THEN C(A+3)=1
8626 NEXT A
```

Code the start and end points

(continued)

Figure 15. The Clipping section

Clipping Section continued

```
8632 FOR A=1 TO 4 STEP 1                    ┐ Off screen
8634 IF C(A)=0 THEN GOTO 8638               │ line check
8636 IF C(A)=C(A+4) THEN GOTO 8668          ┘
8638 NEXT A
8644 FOR A=1 TO 4 STEP 1                    ┐ Off screen start
8646 IF C(A)=1 THEN GOTO 8676               │ point check
8648 NEXT A                                 ┘
8654 FOR A=5 TO 8 STEP 1                    ┐ Off screen end
8656 IF C(A)=1 THEN GOTO 8686               │ point check
8658 NEXT A                                 ┘
8662  P2=1                                  ┐ Set project code
8664 GOTO 8800                              ┘
8668  P2=0                                  ┐ Clr project code
8670 RETURN                                 ┘
8676  A=1                                   ┐ Start point clip
8678  S=5                                   ┘
8680 GOTO 8694
8686  A=5                                   ┐ End point clip
8688  S=1                                   ┘
```

On/off screen checking

```
8694 IF C(A)=1 THEN GOTO 8728               ┐
8696 IF C(A+1)=1 THEN GOTO 8714             │ "Push" direction
8698 IF C(A+2)=1 THEN GOTO 8742             │ decision
8700 IF C(A+3)=1 THEN GOTO 8756             │
8706 GOTO 8662                              ┘
8714  K=(Z(A)-X(A))/(X(S)-X(A)-Z(S)+Z(A))  ┐
8716  X(A)=K*(Z(S)-Z(A))+Z(A)              │ Push point left
8718  Y(A)=K*(Y(S)-Y(A))+Y(A)              │
8720  Z(A)=X(A)                            │
8722 GOTO 8600                              ┘
8728  K=(Z(A)+X(A))/(X(A)-X(S)-Z(S)+Z(A))  ┐
8730  X(A)=K*(Z(A)-Z(S))-Z(A)              │ Push point right
8732  Y(A)=K*(Y(S)-Y(A))+Y(A)              │
8734  Z(A)=-X(A)                           │
8736 GOTO 8600                              ┘
8742  K=(Z(A)+Y(A))/(Y(A)-Y(S)-Z(S)+Z(A))  ┐
8744  X(A)=K*(X(S)-X(A))+X(A)              │ Push point up
8746  Y(A)=K*(Z(A)-Z(S))-Z(A)              │
8748  Z(A)=-Y(A)                           │
8750 GOTO 8600                              ┘
8756  K=(Z(A)-Y(A))/(Y(S)-Y(A)-Z(S)+Z(A))  ┐
8758  X(A)=K*(X(S)-X(A))+X(A)              │ Push point down
8760  Y(A)=K*(Z(S)-Z(A))+Z(A)              │
8762  Z(A)=Y(A)                            │
8764 GOTO 8600                              ┘
```

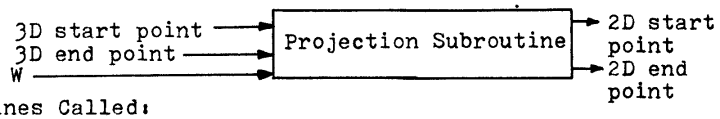
Title: Projection Subroutine 8800

Purpose: This subroutine takes the transformed and clipped 3D coordinates and performs a perspective 2D projection.

Input: x(1),y(1),z(1),x(5),y(5),z(5), W

Output: x(2),y(2),x(4),y(4)

Operation:
This subroutine takes the start and end points of a line and uses the x/z and y/z principle to perform a perspective projection. Protection statements prevent divide by zero conditions.



Subroutines Called:
None

Temporary Storage:
None

```
8800 IF Z(1)=0 THEN Z(1)=.001   ] Pyramid base crash divide
8845 IF Z(5)=0 THEN Z(5)=.001   ] by zero protection
8855  X(2)=X(1)/Z(1)*W         ]
8860  Y(2)=Y(1)/Z(1)*W         ] Start point projection
8865  X(4)=X(5)/Z(5)*W         ]
8870  Y(4)=Y(5)/Z(5)*W         ] End point projection
8875 RETURN
```

Figure 16. The Projection section

The Transformation Matrix Generator

This subroutine (shown in figure 17) generates the 3x3 transformation matrix required for rotation. This matrix need only be created once for each viewing direction. All lines in the scene will use the same matrix. The call for the generator is, "GOSUB 8200".

The Sine/Cosine subroutine is used in conjunction with the Matrix Generator and is shown in figure 18.

The Interface Program

The interface program handles the input and output from the 3D to 2D converter and sends the results to the display device. This subroutine is machine dependent and will be different for different display devices. The basic idea of the program is to sequentially perform the following:

1. Get the screen width and field of view
2. Get the viewer's position and direction of view
3. Call the transformation matrix generator subroutine
4. Feed an array of 3D input points to the 3D to 2D converter, one at a time
5. Send the resulting screen start and end points to the display device to be displayed.

The following Test 1 and Test 2 programs are examples of interface programs and will describe them in more detail.

Title: Matrix Generator Subroutine 8200

Purpose: This subroutine generates the 3 x 3 transformation matrix.

Input: P,B,H,V

Output: T1,T2,T3,T4,T5,T6,T7,T8,T9

Operation:
This subroutine takes the pitch, bank, heading and field of view information and creates the predefined transformation matrix.

P,B,H,V → [Matrix Generator] → T1,T2,· · ·,T9

Subroutines Called:
Sine 8300
Cosine 8310

Temporary Storage:
F,N,R1,R2,R3,R4,R5

```
8200  F=P                          ┐
8203 GOSUB 8300                    │ Sine (Pitch) Calculation
8204  R1=N                         ┘
8206  F=P                          ┐
8209 GOSUB 8310                    │ Cosine (Pitch)
8212  R2=N                         ┘
8215  F=B                          ┐
8218 GOSUB 8300                    │ Sin (Bank)
8221  R3=N                         ┘
8222  F=B                          ┐
8224 GOSUB 8310                    │ Cos (Bank)
8227  R4=N                         ┘
8230  F=H                          ┐
8233 GOSUB 8300                    │ Sin (Heading)
8236  R5=N                         ┘
8237  F=H                          ┐
8239 GOSUB 8310                    │ Cos (Heading)
8245  T1=N*R4+R5*R1*R3            ┐
8248  T2=-N*R3+R5*R1*R4           │
8251  T3=R5*R2*V                  │
8253  T4=R2*R3                    │ Transformation
8256  T5=R2*R4                    │ matrix generation
8259  T6=-R1*V                    │
8262  T7=-R5*R4+N*R1*R3           │
8265  T8=R5*R3+N*R1*R4            │
8269  T9=R2*N*V                   ┘
8272 RETURN                        ┘ Return
```

Figure 17. The Matrix Generator Subroutine

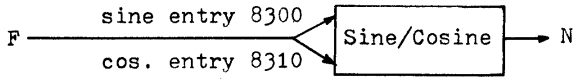
Title: Sine/Cosine Subroutine 8300

Purpose: This subroutine creates a sine or a cosine of a value in degrees so 3DGBU.V03 will run with simple BASIC interpreters without trig functions.

Input: F (angle in degrees)

Output: N (sine or cosine of F)

Operation:
This subroutine takes the F arguement and generates a cosine by expanding a cosine series. Sine is generated by initially shifting the angle 90 degrees. Sines result from subroutine entry at 8300. Cosines result from subroutine entry at 8310.



Subroutines Called:
None

Temporary Storage:
A,S,M

```
8300  F=F-90                                  ] Sine entry phase shift
8305 IF F<=-180 THEN F=F+360                  ]
8310 IF F<0 THEN F=-F                         ]
8315  S=F                                     ] Swing angle into
8320 IF S>=90 THEN F=180-F                    ] quadrant 1
8330  N=F*.01745329                           ]
8340  A=N*N                                   ]
8350  M=A*A                                   ] Cosine series
8360  N=1-A/2+M/24-A*M/720+M*M/40320          ] expansion
8370 IF S>=90 THEN N=-N                       ] Quadrant sign correction
8380 RETURN                                   ] Return
```

Figure 18. The Sine/Cosine Subroutine

USING THE TEST 1 PROGRAM

Test 1 is a simple interface program designed to familiarize the user with the 3D program. The following procedure should be followed to run this test.

1. First, load BASIC into your microcomputer. Any simple 4K basic will do since no trig functions are used in the program.

2. Enter the following subroutines into the computer:
   - Matrix Multiplier
   - Matrix Generator
   - Sine/Cosine Subroutine
   - Clipping Section
   - Projection Subroutine
   - Test 1 Program

3. Check all statements to make sure no errors were made in entry. Double check the 8700 block as this is an error-prone section.

4. Run the program. The first statement executed should be 8000.

The computer should prompt you to enter your first variable. First you will be asked for the "screen width W". Suppose you are working with a graphics terminal which is layed out as a 200x200 dot matrix. The screen is 200 dots wide so enter 200 (followed by a carriage return).

Next, the "field of view V" will be requested. Large values of about 1 represent wide angle views while small values of about .2 represent telephoto views. Enter 1.0 to put the program in a wide angle display mode.

The "viewer's location X,Y,Z" will be asked for. Assume that you are at a point X=Y=Z=0 in 3D space. Enter 0,0,0.

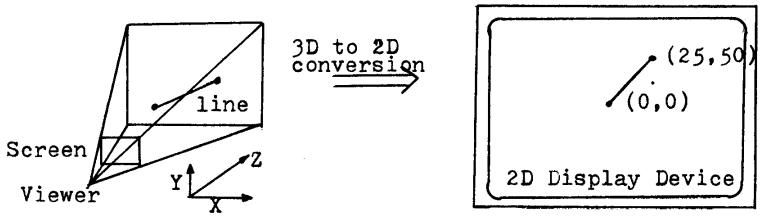
Viewer's direction will now be asked for. Pitch, bank and heading must be submitted so enter 0,0,0 representing a "head-on" view.

As you press carriage return after entering the viewer's direction you will notice a definite delay before the program asks for the next values. The reason is that the program jumped to the matrix generator subroutine at 8200 and calculated the transformation matrix.

The program will now ask for a line in 3D space. First, the start point will be requested. Assume the line begins at 0,0,100 (100 feet straight in front of you). Enter 0,0,100. Now assume the line ends at 25,50,100 (25 feet to the right, 50 feet above and 100 feet in front of you). Enter 25,50,100.

A delay will occur as the program transforms the 3D line to a 2D line and prints the values of the 2D screen points on the terminal. The results should be, 0,0 and 24.75,49.5. Answers will vary by a few thousandths of a percent depending on the BASIC version used but rounding to the nearest integer will always give the proper screen coordinate. If the results are totally wrong, go back and check the BASIC program. An error in entering it may have occured.

The following has been done:



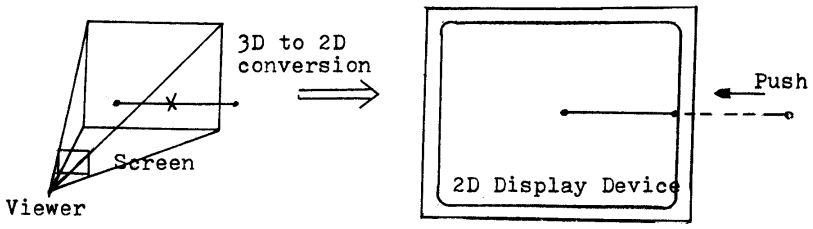
This was a very simple projection. No clipping was performed. The program will now ask for another line to project. Give it the following line:

Start point = 0,0,50
End point = 100,0,50

This represents a line which is off the screen tc the right. The clipping subroutine will clip it and project the following screen points:

Start point = 0,0
End point = 99,0

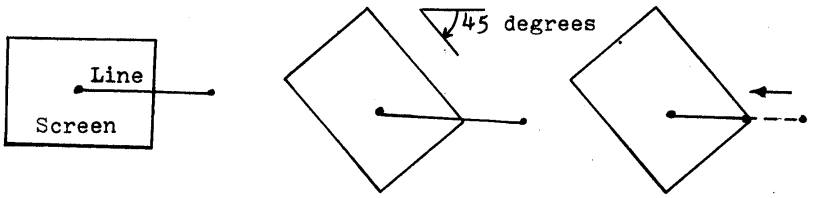
The following has been done:



Now, exit the program (using control C on a SWTP BASIC) and run it again following the procedures already described, but this time, when viewer's direction is asked for, enter 0,45,0. This has introduced a-45 degree bank angle. Now enter the line which had to be clipped earlier:

```
Start point = 0,0,50
End point   = 100,0,50
```

This is what should happen:



a) Original view b) 45 degree bank c) clipped line

Observe the results on the terminal:

```
Start point = 0,0
End point   = 99,99
```

The proper line has been generated.

Now, test the line elimination capability. When a new start point is asked for, enter:

```
Start point = 0,0,-50 (50 feet behind the viewer)
End point = 20,-25,-30  (20 feet right, 25 down,
                30 behind the viewer)
```

The program will print "line off screen" because the line is entirely behind the viewer and is off the screen.

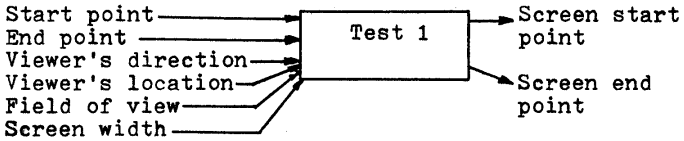
Title: Test 1 8000

Purpose: This test is designed to test the operation of the 3D to 2D converter and familiarize the user with its use.

Input: Typed by user. x(1),y(1),z(1),x(3),y(3),z(3),x(5), y(5),z(5),P,B H,V,W

Output: Printed on terminal. x(2),y(2),x(4),y(4)

Operation:
The Test 1 program asks the user for viewer information and the start and end points of one line. The screen points corresponding to the line are calculated and printed on the terminal.



Subroutines Called:

3D to 2D Converter	8500
Matrix Generator	8200

Temporary Storage
None

```
8000 DIM X(5),Y(5),Z(5),C(8)                    ] Initialize arrays
8005 PRINT "SCREEN WIDTH W"                     ]
8010 INPUT W                                    ]
8015  W=W/2 -1                                  ]
8020 PRINT "FIELD OF VIEW V"                    ] Input viewer's
8025 INPUT V                                    ] information
8030 PRINT "VIEWER'S LOCATION X,Y,Z"            ]
8035 INPUT X(3),Y(3),Z(3)                       ]
8040 PRINT "DIRECTION OF VIEW P,B,H"            ]
8045 INPUT P,B,H                                ] Compute transfor-
8050 GOSUB 8200                                 ] mation matrix
8055 PRINT "START POINT X,Y,Z"                  ]
8060 INPUT X(1),Y(1),Z(1)                       ] Input line
8065 PRINT "END POINT X,Y,Z"                    ] information
8070 INPUT X(5),Y(5),Z(5)                       ]
8075 GOSUB 8500                                 ] 3D to 2D convert
8080 IF P2=0 THEN GOTO 8095                     ]
8082 PRINT "SCREEN START PT = ";X(2),Y(2)       ]
8083 PRINT "SCREEN END PT = ";X(4),Y(4)         ] Print the results
8090 GOTO 8055                                  ]
8095 PRINT "LINE IS OFF SCREEN"                 ]
8099 GOTO 8055                                  ]
```

Figure 19. The Test 1 Program

ARRAY HANDLING IN THE INTERFACE PROGRAM

The Test 1 program was fine for generating a single line projection, but images are formed from many lines or an array of lines. This is where the input array concept comes in. A list of lines can be placed in a BASIC array and a loop can be set up to feed the lines to the 3D to 2D converter, one at a time. The Q, R, and S values correspond to X, Y, and Z start and end points. The following list, which should be entered at locations 1000 to 1035 represents a pyramid.

```
1000  Q(1)=100   ]
1001  R(1)=0     ] pt 1   ]
1002  S(1)=1100  ]        ] line 1
1003  Q(2)=500   ]
1004  R(2)=0     ] pt 2   ]
1005  S(2)=1100  ]        ]
1006  Q(3)=500   ]
1007  R(3)=0     ] pt 3   ]
1008  S(3)=1100  ]        ] 2
1009  Q(4)=300   ]
1010  R(4)=0     ] pt 4   ]
1011  S(4)=1400  ]        ]
1012  Q(5)=300   ]
1013  R(5)=0     ] pt 5   ]
1014  S(5)=1400  ]        ] 3
1015  Q(6)=100   ]
1016  R(6)=0     ] pt 6   ]
1017  S(6)=1100  ]        ]

1018  Q(7)=100   ]
1019  R(7)=0     ] pt 7   ]
1020  S(7)=1100  ]        ] line 4
1021  Q(8)=300   ]
1022  R(8)=400   ] 8      ]
1023  S(8)=1200  ]        ]
1024  Q(9)=300   ]
1025  R(9)=0     ] 9      ]
1026  S(9)=1400  ]        ] 5
1027  Q(10)=300  ]
1028  R(10)=400  ] 10     ]
1029  S(10)=1200 ]        ]
1030  Q(11)=300  ]
1031  R(11)=400  ] 11     ]
1032  S(11)=1200 ]        ] 6
1033  Q(12)=500  ]
1034  R(12)=0    ] 12     ]
1035  S(12)=1100 ]        ]
```

Figure 20. An input array for a pyramid

Test 2, an array handling modification of Test 1, can now be entered and run. Remember to enter the pyramid values also. The program will ask for viewer's information as it did before but line start and end points will be read sequentially out of the input array. All six screen lines will be printed on the terminal.

Many-lined images can be drawn using larger Q,R,S array sizes but BASIC uses up memory fast. Large amounts of memory will be needed to generate large, complex scenes. This is one of BASIC's disadvantages.

Title: Test 2 201

Purpose: This test is designed to illustrate the use of an input array of start and end points of lines.

Input: Typed by user: x(3),y(3),z(3),P,B,H,V,W
Input array: Q,R,S

Output: Printed on terminal. x(2),y(2),x(4),y(4)

Operation:
Same as Test 1 except an input array is fed to the 3D to 2D converter.

Subroutines Called:
3D to 2D Converter
Matrix Generator

Temporary Storage:
A1

```
 0201 DIM Q(12),R(12),S(12)                    ┐
 8000 DIM X(5),Y(5),Z(5),C(8)                  ┘ Initialize arrays
 8005 PRINT "SCREEN WIDTH W"                   ┐
 8010 INPUT W                                  │
 8015  W=W/2-1                                 │
 8020 PRINT "FIELD OF VIEW V"                  │ Get viewer's
 8025 INPUT V                                  │ information and
 8030 PRINT "VIEWER'S LOCATION X,Y,Z"          │ calculate the
 8035 INPUT X(3),Y(3),Z(3)                     │ transformation
 8040 PRINT "DIRECTION OF VIEW P,B,H"          │ matrix
 8045 INPUT P,B,H                              │
 8050 GOSUB 8200                               ┘
 8055 FOR A1=1 TO 11 STEP 2                    ┐
 8060  X(1)=Q(A1)                              │
 8063  Y(1)=R(A1)                              │
 8066  Z(1)=S(A1)                              │ "Feed lines to the
 8069  X(5)=Q(A1+1)                           │  3D to 2D converter"
 8072  Y(5)=R(A1+1)                           │  loop
 8075  Z(5)=S(A1+1)                           │
 8078 GOSUB 8500                               │
 8080 IF P2=0 THEN GOTO 8095                   │
 8082 PRINT "SCREEN START PT = ";X(2),Y(2)     │
 8083 PRINT "SCREEN END PT = ";X(4),Y(4)       │
 8084 GOTO 8098                                │
 8095 PRINT "LINE IS OFF THE SCREEN"           │
 8098 NEXT A1                                  ┘
 8099 GOTO 8030                                ┐ Next view
```

Figure 21. The Test 2 Program

A SIMPLE DISPLAY DRIVING PROGRAM

Instead of printing screen start and end points on the terminal, the start and end points can be sent to a display device to create the final result; a real display. Two system prerequisites must be met before this can be done.

1. You must have some sort of display device to drive.

2. BASIC must be capable of controlling the display divice directly, or BASIC callable assembly language subroutines which are capable of display device control must exist.

Note that any BASIC having a PEEK and POKE instruction can manipulate memory and a display device just as assembly language can. A number of 4K BASICs, unfortunately, have no provision for communication to assembly language programs and don't possess PEEK and POKE instructions either. Running a display device other than the console terminal (which could indeed be a graphics terminal) is very difficult and involves programming tricks.

The Display Interface Program is a modification of the Test 2 program and has screen driving capabilities. It was written for a Graphics One terminal using Southwest Technical Products' 8K BASIC and a SWTP 6800 CPU. Array handling is performed as it was in Test 2 but instead of printing results it jumps to the 9000 Display Control subroutine. The subroutine sheet describes its operation but basically the display control subroutine "POKEs start and end points at the display terminal".

The Graphics One terminal (the one we use at SubLogic) has a built-in vector generator. If your display device doesn't have vector drawing abilities, the Vector Drawing Subroutine (8900) can be used to calculate all the picture elements between a start and end point. This subroutine involves no trig, multiplies or divides and is thus easy to translate into fast assembly language.

```
0201 DIM Q(12),R(12),S(12)                    Initialize arrays
8000 DIM X(5),Y(5),Z(5),C(8)
8005 PRINT "SCREEN WIDTH W"
8010 INPUT W
8015  W=W/2-1
8020 PRINT "FIELD OF VIEW V"                  Get viewer's
8025 INPUT V                                  information and
8030 PRINT "VIEWER'S LOCATION X,Y,Z"          create the
8035 INPUT X(3),Y(3),Z(3)                     transformation
8040 PRINT "DIRECTION OF VIEW P,B,H"          matrix
8045 INPUT P,B,H
8050 GOSUB 8200
8055 FOR A1=1 TO 11 STEP 2
8060  X(1)=Q(A1)
8063  Y(1)=R(A1)
8066  Z(1)=S(A1)                              Array handling
8069  X(5)=Q(A1+1)                            loop sends lines
8072  Y(5)=R(A1+1)                            to the terminal
8075  Z(5)=S(A1+1)                            communication
8078 GOSUB 8500                               section (block
8081 GOSUB 9000                               9000)
8084 NEXT A1
8085 GOTO 8030                                Get new viewer data
9000 IF P2=0 THEN RETURN
9010  X(2)=X(2)+100                           Check if display flag
9011  Y(2)=100-Y(2)                           P2 is set. Calculate
9012  X(4)=X(4)+100                           corrected screen points.
9014  Y(4)=100-Y(4)
9500 POKE( 32792,19)                          Clear the PIA  19,17
9505 POKE( 32792,17)                          Send vector code 89
9510 POKE( 32793,89)
9515 GOSUB 9550                               Wait for echo
9520 POKE( 32793,X(2))
9522 GOSUB 9550
9525 POKE( 32793,Y(2))                        Send screen start and
9527 GOSUB 9550                               end point to graphics
9529 POKE( 32793,X(4))                        terminal.
9531 GOSUB 9550
9535 POKE( 32793,Y(4))
9540 RETURN                                   Return
9550  B9=PEEK (32792)                         "Wait for graphics term-
9555 IF B9=0 THEN GOTO 9550                   inal echo"subroutine
9560 RETURN
```

Figure 22. The Display Interface program

```
8900 REM LINE GENERATOR - - - - - - - -
8903 REM A SUM TRACKING ALGORITHM IS USED TO
8906 REM GENERATE ALL PIXELS BETWEEN TWO POINTS.
8909 REM THIS PROGRAM WILL ASK FOR A START AND END
8912 REM POINT "X1,Y1 AND X2,Y2" AND WILL PRINT
8915 REM THE PIXELS. ONLY INTEGERS ARE ALLOWED.
8918 REM LINE GENERATOR - - - - - - - -
8921 PRINT "ENTER THE SCREEN POINT X1,Y1"
8924 INPUT X1,Y1
8927 PRINT "ENTER THE SCREEN POINT X2,Y2"
8930 INPUT X2,Y2
8933  S=0
8936  M=1
8939  N=1
8942  D=X2-X1
8945 IF D<0 THEN M=-1
8948 IF D<0 THEN D=-D
8951 IF D=0 THEN S=-1
8954  E=Y2-Y1
8957 IF E<0 THEN N=-1
8960 IF E<0 THEN E=-E
8963 PRINT "PIXEL = ";X1,Y1
8966 IF X1=X2 THEN GOTO 8990
8969 IF S<0 THEN GOTO 8981
8972  X1=X1+M
8975  S=S-E
8978 GOTO 8963
8981  Y1=Y1+N
8984  S=S+D
8987 GOTO 8963
8990 IF Y1=Y2 THEN GOTO 8921
8993 GOTO 8969
8996 REM PROGRAM END
```

Figure 23. A line between points drawing program

CUSTOMIZED INTERFACE PROGRAMS

The display interface program of figure 22 will need modification to project images on a display device other than a Graphics One terminal. The 9000 block of the program is where the screen lines are sent to the terminal and where the modification must be made. If your display device has a protocol which allows vectors to be drawn, simply scale and bias the screen start and end point values with addition and multiplication to meet the device's format. This is what was done in statements 9010-14 of the display interface program (figure 22).

When writing interface software make sure you don't interfere with the 3D to 2D conversion software by using already used variables. The following variables should be avoided: X array, Y array, Z array, C array, T1-T9, R1-R5, A, B, F, G, H, K, M, N, P, P2 ,S, V, and W. You can, of course, modify X(3), Y(3), Z(3), P, E, H, V and W for the purpose of changing viewer's position, direction of view and field of vision. Just be careful not to use them as temporary storage. If strange things start happening with the display program after writing a new interface program, violation of this rule is the first thing to check.

CONCLUSION

Three dimensional graphics displays, games and simulations are hard to develop and implement. Unlike many simpler software packages (versions of BASIC, simple games, Star Trek, etc.) this 3D graphics package was not intended to be a load-and-go program package. It was intended to be a versatile package of subroutines which, after a lot of system dependent interface work, results in a very useful and advanced piece of customized display software. Getting 3D graphics up and running may not be easy, but when you finally achieve your end result, it will undoubtedly be the most impressive piece of software you own.

We at SubLogic are interested in hearing about your work in 3D graphics. We are constantly trying to improve, debug and speed-up our 3D to 2D converters and the more feedback from the field we get, the better our products will get.

Once again, thank you for ordering the SubLogic BASIC Microcomputer Graphics Package.

APPENDIX

SECTION

APPENDIX 1- Memory Map

BASIC Block	
8000	User's Interface Program
8100	
8200	Transformation Matrix Generator
8300	Sine/Cosine Subroutine
8400	
8500	Matrix Multiplier
8600	Clipping Section
8700	
8800	Projection Section
8900	Line Drawing Program
9000	Display Device Driver Subroutine
9100	

Viewer's Information

Variable	Definition
X(3)	Viewer's X location in space
Y(3)	Viewer's Y location in space
Z(3)	Viewer's Z location in space
P	Viewer's Pitch (inclination)
B	Viewer's Bank (roll)
H	Viewer's Heading (North,South...)
V	Field of View (wide angle, telephoto)
W	Screen Width (from center to edge)

APPENDIX 2- GRAPHIC PRINCIPLES

A translation, rotation, clipping and projection algorithm must be applied to each line submitted to the 3D to 2D converter program. The details of the four step process will now be discussed.

Point Translation

The viewer's location in space is always considered to be at 0,0,0 in 3 coordinate space. When the program user specifies a location other than 0,0,0 the points in the data base are translated and the viewer remains at 0,0,0. In other words, the whole world moves and the viewer remains stationary. Each individual point in the data base has an X, Y, and Z translational value added to it. Figure a2-1 illustrates translation.

Point Rotation

As with translation, it's the world which rotates around the viewer when point rotation is performed. Through geometric principles, a 3 x 3 matrix, which when multiplied by a 3 element vector (a 3D space coordinate) rotates it about the origin, was derived and is shown in figure a2-2. This matrix need only be created once per each viewing direction since it applies to all points for that view. Sines and cosines of the pitch, bank and heading (the

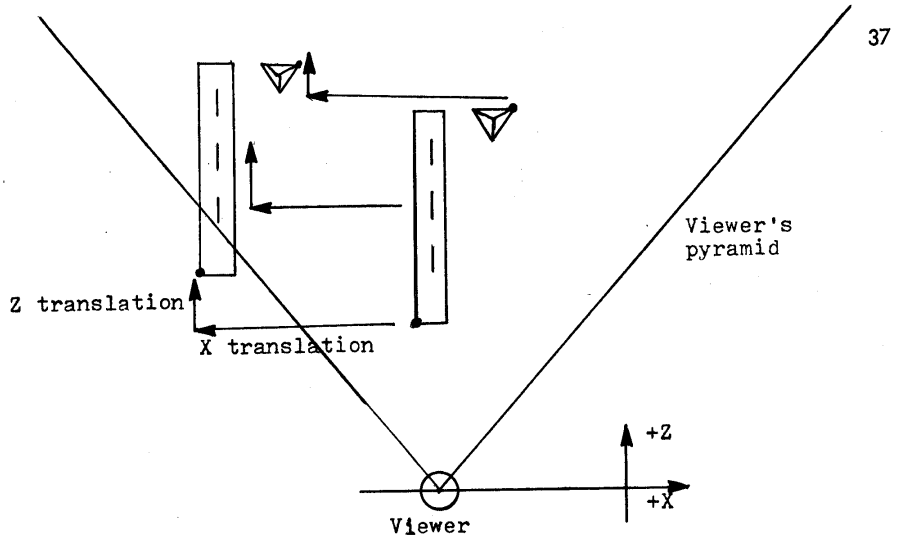


Figure A2-1. Point translation of a data base

$$[X'\ Y'\ Z'] = [X\ \ Y\ \ Z] \begin{bmatrix} \text{Cos H Cos B} + \text{Sin H Sin P Sin B} & -\text{Cos H Sin B} + \text{Sin H Sin P Cos B} & \text{Sin H Cos P} \\ \text{Cos P Sin B} & \text{Cos P Cos B} & -\text{Sin P} \\ -\text{Sin H Cos B} + \text{Cos H Sin P Sin B} & \text{Sin H Sin B} + \text{Cos H Sin P Cos B} & \text{Cos H Cos P} \end{bmatrix}$$

Where $[X'\ Y'\ Z']$ = Transformed (rotated) point,
$[X\ \ Y\ \ Z]$ = Original point,
P = Pitch
B = Bank
H = Heading

Figure A2-2. The rotational matrix being multiplied by the original 3D space coordinate point (row vector) yielding the rotated point

direction of view) must be computed to generate this matrix. In the interest of speed, lookup tables are used in the 3D Microcomputer Graphics Package (assembly versions only).

Line Clipping and Coding

The operation which takes the longest in the 3D graphics program is that of clipping and eliminating lines that fall off or partially off the screen. Figure A2-3 illustrates a line in need of clipping. The mathematics of clipping a line and pushing end points to screen boundaries are quite simple but deciding which way to push them is what takes up the time.

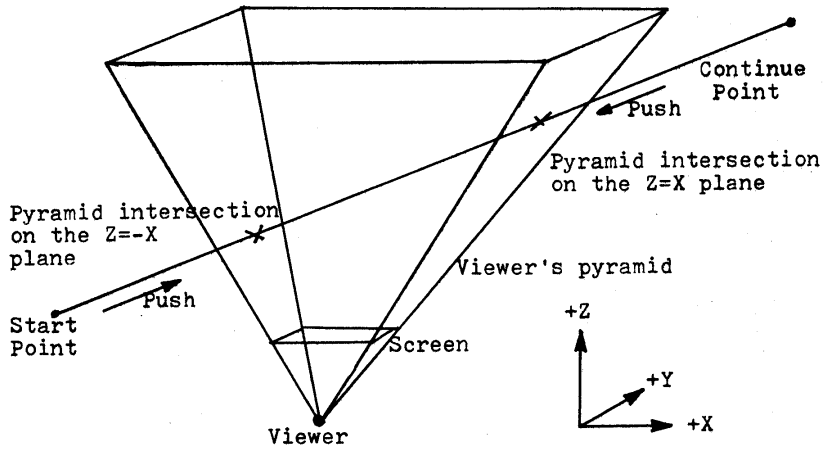


Figure A2-3. A line in need of clipping

Every line's start and end points are assigned code values which indicate which side of the viewing pyramid the points fall. The viewing pyramid consists of four intersecting planes whose apex is at the viewer's eye. A pyramid cross section represents the screen onto which objects are projected. The equations of the four planes are: X=Z, -X=Z, Y=Z and -Y=Z.

After translation and rotation, a 4 bit code is set up for each point in space. The four bits indicate:

C0= 1 = point to left of -X=Z plane
C1= 1 = point to right of X=Z plane
C2= 1 = point is above the Y=Z plane
C3= 1 = point is below the -Y=Z plane

If a point's code is all zeros, the point is within the viewing pyramid. If it has some ones in it, it is off the screen but may represent a line which intersects the screen. The line's start and end point's codes are compared to check if the line is off the screen. One sure off-screen test is to see if the start and end points are off the screen in the same direction (both to the right of the screen for example). By simply "anding" the two codes, any common off-sides condition can be found.

It is not always this easy however. Suppose the start

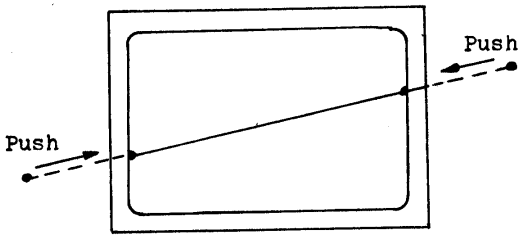
point is to the left of the screen and the end point is to the right. In this case, the codes are 1000 and 0100. The "and" of the codes is 0000 which means the line might be on the screen partially. The 1000 code indicates that the start point is to the left of the screen and must be pushed right while the 0100 code means the end point is too far to the right and must be pushed left. The push mathematics are performed for the right push:

```
k=(z(a)+x(a))/(x(a)-x(b)-z(b)+z(a))
x(a)=k*(z(a)-z(b))-z(a)
y(a)=k*(y(b)-y(a))+y(a)
z(a)=-x(a)
```

and for the left push:

```
k=(z(a)-x(a))/(x(b)-x(a)-z(b)+z(a))
x(a)=k*(z(b)-z(a))+z(a)
y(a)=k*(y(b)-y(a))+y(a)
z(a)=x(a)
```

and the line is ready to be projected onto the screen. Essentially the following has been done:



Sometimes after one push, it becomes apparent that the line will not intersect the viewing pyramid after all and the line must be eliminated.

Projection

After the line has been clipped, the 3D to 2D perspective projection must be performed. By plotting space coordinates X/Z and Y/Z for every point within the viewing pyramid, a true perspective image can be generated on the display device. Division by the point's depth (Z) causes objects in the the distance to appear smaller. Care must be taken to avoid projecting points lying at the base of the viewing pyramid (X=Y=Z=0) as division by zero will result. A point at the base of the wiewing pramid is not definable because it implies a view of an infinitesimally small point from a distance of zero (at the viewer's eye).

Integer Graphics

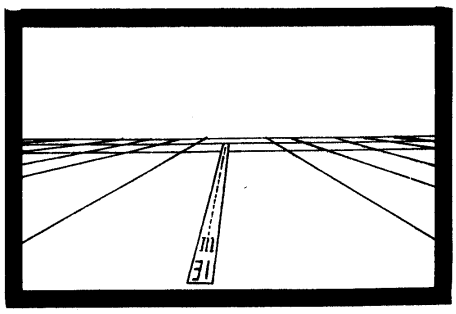
Integer arithmetic is, speedwise, far superior to floating point and was thus chosen for the assembly language 3D graphics package. Double precision 8 bit words are used for all space coordinates providing a range of 32767 units in each direction. The boundaries of the 3D scene, however, should be less since the viewer's translational offsets will be added to each point. In order to increase processing speed, no overflow checking is performed in additions and multiplications and points which overflow will end up on the wrong side of the scene resulting in display distortion.

APPENDIX 3

APPLICATION NOTES - - - FLIGHT SIMULATION

Much of the existing information and many of the accomplishments in the 3D graphics field are a direct result of flight simulation research. Flight simulation is an area where 3D graphics has many advantages over the real thing. Flight training costs are lowered, ther is no interference from bad weather,and there is no risk of crashing.

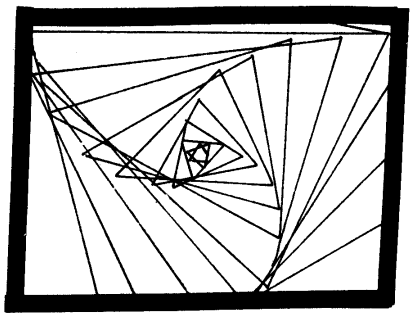
Pilots using a 3D graphics equipped simulator can learn more about the flight characteristics of an aircraft. Spins, steep dives, near crashes and generally pushing the plane to and beyond its limits are all safe maneuvers and a pilot can learn what to expect in any of these situations.



Hidden line elimination adds very little to flight simulation once a viewer is more than a few feet above the ground. This makes very realistic, fast simulations possible at a low cost.

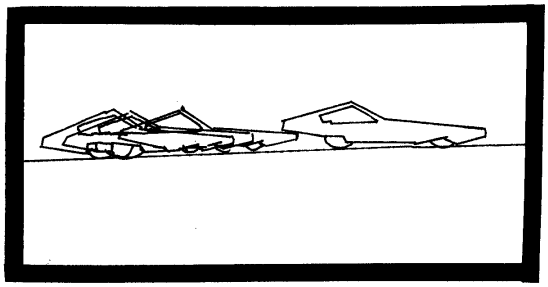
APPLICATION NOTE - - - COMPUTER ART

The SubLogic 3D Microcomputer Graphics package makes a very good computer art tool. It can do things which would be nearly impossible using standard 2D techniques. An example of this is the spiraling triangle. By placing a triangle at a great distance and slowly approaching it while increasing the bank angle, a very interesting picture results. Frame erasing between frames has been turned off as figure a illustrates.



By rotating, superimposing and moving in space, dramatic curved and radially spiraling figures are generated. Very complex figures as well as simple triangles can be used.

Motion effects are also possible by superimposing 3 or 4 frames with different reference frames.



Film makers can also utilize 3D graphics as an animation aid. Additional realism is possible since accurate perspectives are always generated.

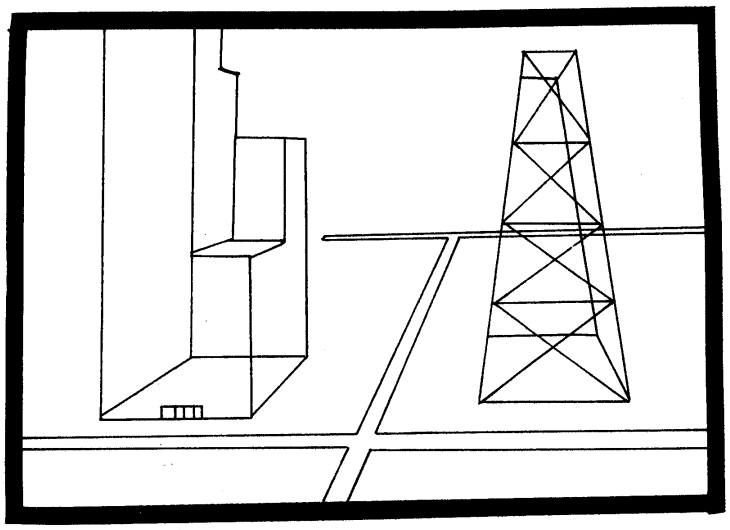
APPLICATION NOTE - - - ARCHITECTURAL DESIGN

Architectural models have been used in the development of buildings and other structures for centuries. Computer generation of views of buildings, however, is a relatively recent innovation. Computer generated projections offer a few important advantages over more conventional models:

1. They are less expensive when implemented with a microcomputer based system
2. They are easily constructed and modified
3. The user has the ability to observe the scene from between and inside the buildings.

Hidden line elimination is very beneficial to architectural projections.

The BASIC version of the microcomputer graphics package is well suited to architectural design graphics. Taking a few minutes to generate a complex scene is acceptable and high precision is a must.

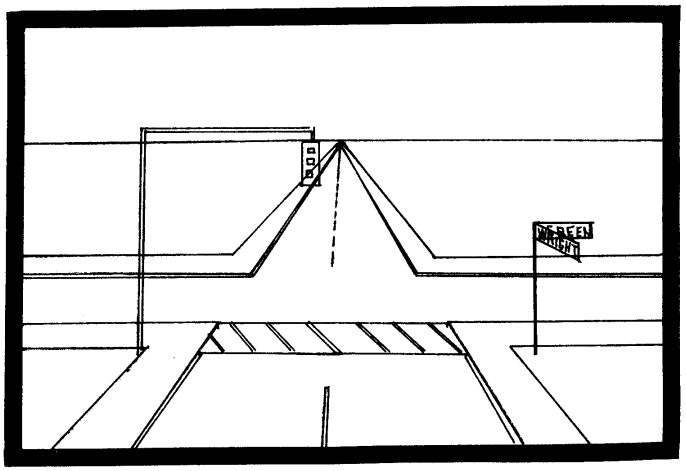


APPLICATION NOTE - - - DRIVING SIMULATION

Driver training is one area where very few computer simulations are currently used. The main reason is that it is not cost effective. A whole fleet of driver education cars can be bought for the price of a single dedicated 3D graphics generator. The microcomputer, in conjuction with the SubLogic 3D Microcomputer Graphics Package can change this.

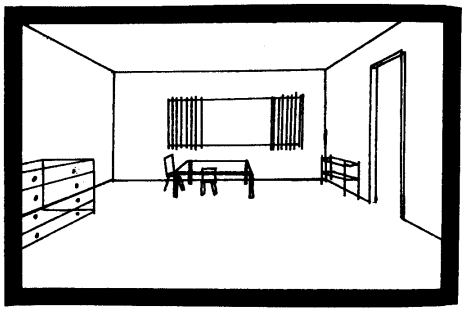
By projecting a training course on an inexpensive projection television in front of the driver, he can practice driving all day without an instructor. With todays rising gas and auto prices and dropping computer costs, this sort of simulation gets even more attractive.

As with flight simulation, you can do things with a driving simulator which you would never do on a road. A student's emergency procedures can be tested by having another car pull out in front of him unexpectedly. Driver control at high speeds can also be tested.



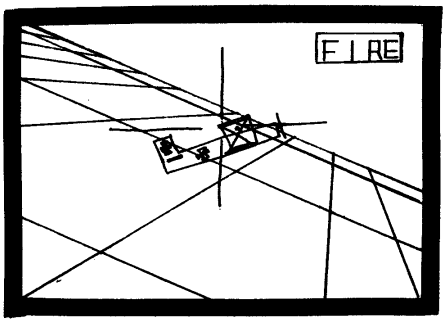
APPLICATION NOTE - - - ROOM LAYOUT VISUALIZATION

Using sketches and cardboard cut-outs to visualize a room layout before moving or buying furniture is helpful,but the final result never quite looks like what you expected it to. The 3D microcomputer graphics package can project views of rooms with true perspective. Once the data base is in the computer you will be able to look at a room from any angle and location. Walls and ceilings can also be included in the simulation



APPLICATION NOTE - - - 3D GAMES

Computer games have always been popular but is seems that half the microcomputer applications now-a-days involve a game of some sort. Most of the games involve 2D displays. Three dimensional graphics can add a whole new dimension to these games but imagine games like"3D tank" or 3D dog-fight. Two WW 1 aces can be flying in each others data bases. An actual two playeraerial battle is possible.



APPLICATION NOTE - - - ENGINEERING DRAWING

Traditional engineering drawings help an engineer design and get a good idea of what his finished product will look like. Typically, three oblique views are projected (a top, front and side view) and a perspective 3D view is often included. The engineer or draftsman must do all the calculations to determine what the views will be. This amounts to four drawings and a lot of work. Three dimensional microcomputer graphics can be a great benefit in the construction of these drawings.

Every engineering graphics student learns about the two ways to calculate cross sectional views of objects. There are the standard graphics methods which amount to drawing lines from the original to a projection line and back to another view, and there is the much more accurate but very difficult analytic method which uses equations to project lines. The 3D graphics package uses the analytic method resulting in more accurate as well as faster drawings.

An engineering drawing can be set up using the 3D Microcomputer Graphics Package as follows:

1. The object (machine, architectural structure, road, bridge, etc.) should be put into a 3D data base form and loaded into computer memory.

2. A telephoto view of the object from a great distance should be projected. An oblique view will result.

3. A top, side and front view, as well as any desired cross-sectional views should be projected.

4. Finally, a close-up view with a wide angle field of view can be projected resulting in a dramatic perspective view.

With a little work and imagination even more impressive things can be done. An interface program can be set up to take 4 passes through the data base before an object is projected. Four views at once can then be put on the screen as figure a shows. The computer can do in seconds what would have taken a draftsman hours. Needless to say, a high resolution graphics device is very desirable in this application.

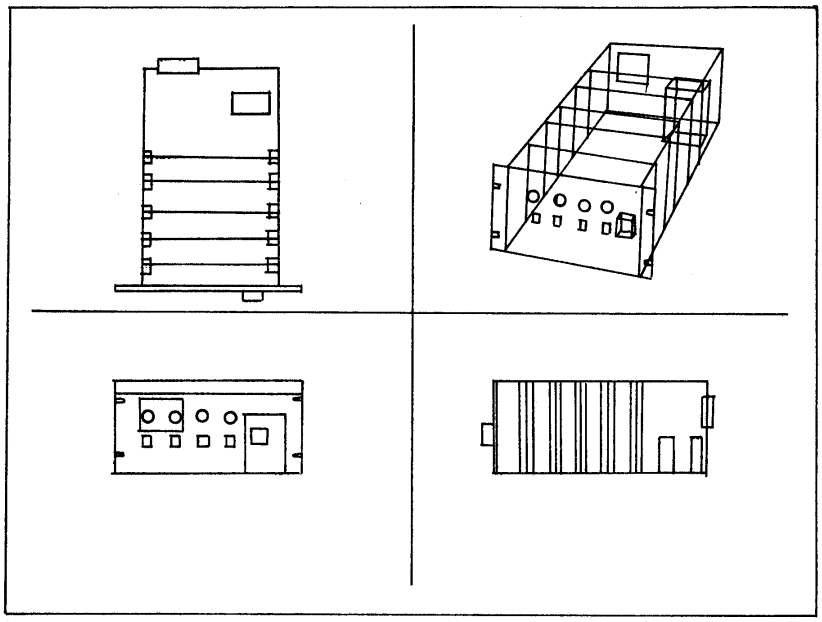


Figure a. An engineering drawing generated using 3D microcomputer graphics

APPENDIX 4

MISCELLANEOUS TOPIC - - - ADVANCED GRAPHICS CONCEPTS

Three dimensional wire-frame projections are a very simple form of computer graphics. More realistic projections can be generated using more advanced and much more difficult projection algorithms.

Hidden line elimination is a very desirable feature. Lines which are blocked by other surfaces in space are clipped against them or eliminated. The problem with hidden line elimination is computation time. A number of hidden line elimination algorithms exist. These algorithms either compare every line against every surface or use nondeterministic methods to look for conflicts and try to resolve them. Both methods are very time consuming.

A simple line projection program can not easily be converted into a hidden line algorithm program. In hidden line elimination algorithms, surfaces and planes are dealt with. A whole different method of representing objects is the result. Figure a illustrates hidden line elimination.

1) a wire frame object 2) hidden lines removed

Figure a. Hidden line elimination

A number of interesting problems can exist when working with hidden line and hidden surface situations. Figure b, for example, shows a condition where two surfaces block one-another.

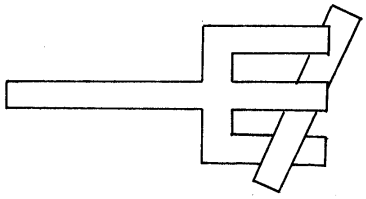


Figure b. Hidden surface conflict

Problems like these can be handled by breaking surfaces into smaller surfaces, but this takes even more computation time. A very good article concerning ten hidden line and surface algorithms can be found in Computing Surveys magazine, March 1974.

Another advanced graphics technique is shading. Surfaces at different angles have different color shades when projected due to light angle and viewing angle. Shading adds a very realistic effect to 3D pictures when used with hidden surface elimination.

Shadowing is a difficult task and adds little to the realism of a picture other than the feeling that light is striking objects from a certain direction. It is very interesting to experiment with, however. Figure c shows shadowing.

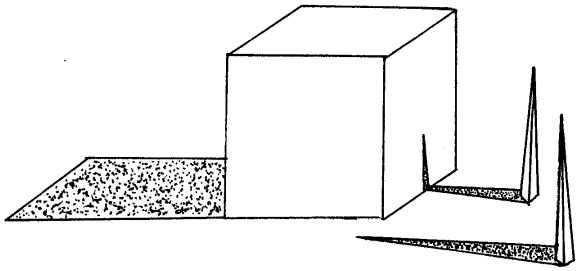


Figure c. Shadowing

Atmospheric degradation makes objects fade away as they get farther away. The effect can be used to simulate fog or haze.

Specialized hardware which produces 3D graphics with shading, hidden surface elimination, and atmospheric degradation at the rate of 30 frames per second currently exists but is very expensive. Instead of having a subroutine perform a function, this equipment has a logic card perform the function.

MISCELLANEOUS TOPIC - - - GROUND TEXTURE

Dynamic 3D graphics is very useful in flight and driving simulations. The technique of ground texture generation can be used to dramatically increase the realism and enhance the user's ability to tell where and how he is moving in space. By laying out a grid on the ground in the 3D scene, an illusion of a solid ground surface is created (see figure a).

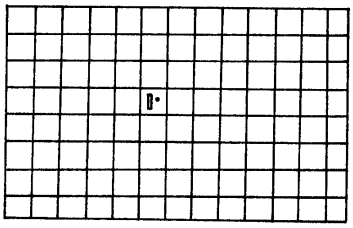
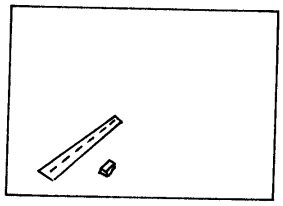
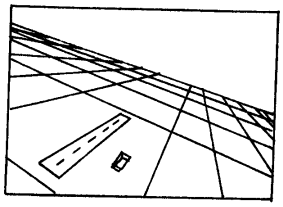


Figure a. A runway and a ground grid

Notice the difference in the final result on the screens of figure b. The orientation of the runway is much clearer when the ground grid is present, not to mention the more dramatic look. Perspective is also more obvious.



1) the runway



2) the runway with ground texture

Figure b. The difference ground texture makes

A ground grid provides a good vertical and horizontal movement queue also. At large distances from the airport, aircraft movement can be sensed as you fly over the grid lines. Additional realism is created by the creation of a "horizon" at the end of the grid. This is something everyone is used to seeing and can judge bank by.

MISCELLANEOUS TOPIC - - - THE EXACT FIELD OF VIEW

Throughout the 3D Graphics Package the concept of field of view was described from a wide angle/ narrow angle telephoto point of view. There is, however, a geometrically correct viewing angle for any given situation. This angle is determined by the screen's physical size and the distance the viewer is from it. Figure a illustrates this concept.

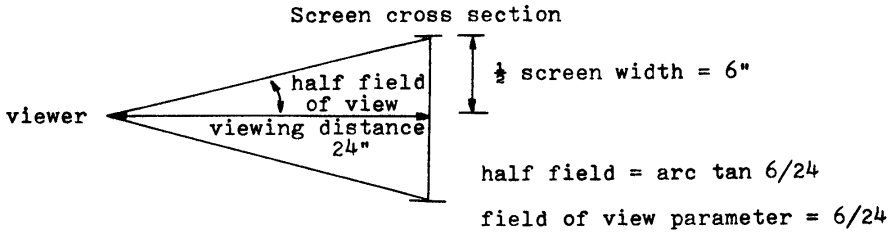


Figure a. The correct field of view

A simple way to calculate the precise viewing parameter is by dividing half the screen width by the distance the viewer is from the screen.

Using any other viewing angle is geometrically incorrect but the views are still very acceptable. This principle also applies to television and photographs which look acceptable from many different viewing distances.

MISCELLANEOUS TOPIC - - - PARALLEL PROCESSING AND GRAPHICS

General purpose computer tasks are usually quite serial in nature making parallel processing of data undesirable. Three dimensional graphics, however, is one of those rare cases where there are almost unlimited parallel processing prospects. Offset addition, matrix multiplication, clipping, projecting and vector drawing can all be handled independently. Since a large array of lines are usually transformed, one processor can operate on one line while another works on the next line.

SubLogic has used parallel processing to a small extent. The Graphics One terminal has a built-in microcomputer which was programmed to draw white vectors, erase the screen, and erase individual lines. Line's start and end points were sent to the G1 terminal where vectors were computed hile the next 3D to 2D conversion was being performed by the SWTP 6800 CPU. Very little time was saved however since 3D to 2D conversion took many times as long as vector generation.

The subroutine which takes the longest to perform on a microcomputer without hardware multiply capabilities is the matrix multiply section. Clipping time runs a close second. If a hardware multiply is available, however, the matrix multipication time is reduced to only about 10% of the calculation time with clipping jumping to nearly 75% of the processing time. Percent figures like these are very machine dependent.

Before trying parallel processing, it is wise to time the program components to see where the time is going. An even task distribution among processors is the mark of a good parallel processing system. If program timings are not performed, you are likely to have the same problem general purpose program parallel processing systems have: idle processor elements. Figure a shows a simple parallel processing system.

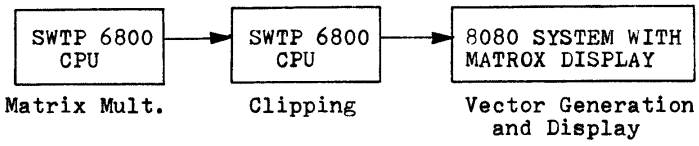


Figure a. A parallel processing system

MISCELLANEOUS TOPIC - - - DYNAMIC DATA BASES

There is no reason why the data base or scene which will be projected must remain static. In many applications a moving object in a scene is desirable.

A driving simulation is a good example of the use of a dynamic data base. Other cars on the road should be able to move, dart out in front of you and cut you off just like they do in real life.

There are two ways to create a dynamic data base. One is by actually manipulating the data base values with a user written subroutine. Values can be added and subtracted from every coordinate point. Using different reference frames is another way to make objects move. Actually, both methods are the same. Using another reference frame lets the 3D to 2D converter add and subtract the offsets for you.

MISCELLANEOUS TOPIC - - - GENERATING DATA BASES

Laying out a 3D scene on a large sheet of graph paper is a hard way to generate a 3D data base. If many data bases are going to be used, it may be worth while to have your computer help you with the task of generating them. There are a few methods ranging from very expensive to no cost at all which can be used.

A data tablet can be used to specify lines in 3D space. Just drawing the lines on the tablet will automatically calculate 3D space coordinates and enter them in the data base. Data tablets are very expensive.

A joystick arrangement is just as versatile as the data tablet. By directing a scene-drawing cursor with the joystick a 3D data base can be entered.

The least expensive method,which anyone can use, consists of a keyboard controlled relative movement program. Instead of entering every point in the 3D data base, the user specifies where in space the next point should be, relative to the last. Commands such as +30X would generate array entries corresponding to the absolute location. To initially start the cursor or to start a new start point, an absolute command such as A 25,1050,35 could be used to specify the X,Y and Z values.

MISCELLANEOUS TOPIC - - - INDEPENDENT REFERENCE FRAMES AND THE HORIZON LINE

The 3D to 2D converter subroutine can be used to handle many different input arrays in a single program. A ground grid array can be transformed followed by an airport array and so on. This brings up the possibility of not only transforming arrays seperately, but differently as well. By changing viewer's information and creating new transformation arrays between 3D to 2D conversions, objects in different reference frames can be generated. Take a driving simulation as an example. You may wish to project the view out the windshield. The view of the world will depend on X, Y and Z viewer location but the hood of the car will always be right out in front of the viewer. By setting up two data bases, one for the car and one for the world, and using two reference frames, two arrays can be transformed into the desired image.

By having a seperate reference frame with a 180 degree heading, and a very small screen width parameter, and with a little biasing, a rear view mirror could even be set up!

A very good use for the variable reference frame is in horizon line generation. The edge of a ground grid can be used as a horizon but it is not an accurate horizon. First of all, it does not represent the true location at infinity where the horizon should be. The closer you get to the horizon (ground grid edge) the worse the the distortion becomes (see figure a)

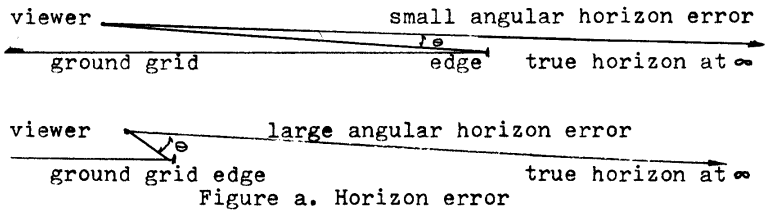


Figure a. Horizon error

A better horizon can be generated by putting a square boundary around the edge of the horizon data base and transforming it seperately. When transforming, however, the viewer's location should be set to 0,0,0 and only rotation (P,B and H) should be performed. The result will be a true horizon which you can never fly up to or over.

On displays having color capabilities it is desirable to project multi-color images. Since the 3D graphics program is array oriented, this task can be performed by simply setting aside an individual array of 3D lines for each seperate color. Each array can be processed and sent to the display device with the proper color code. Figures a and b illustrate the color arrays and the projection order.

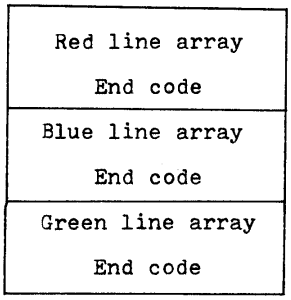


Figure a. Three seperate color arrays

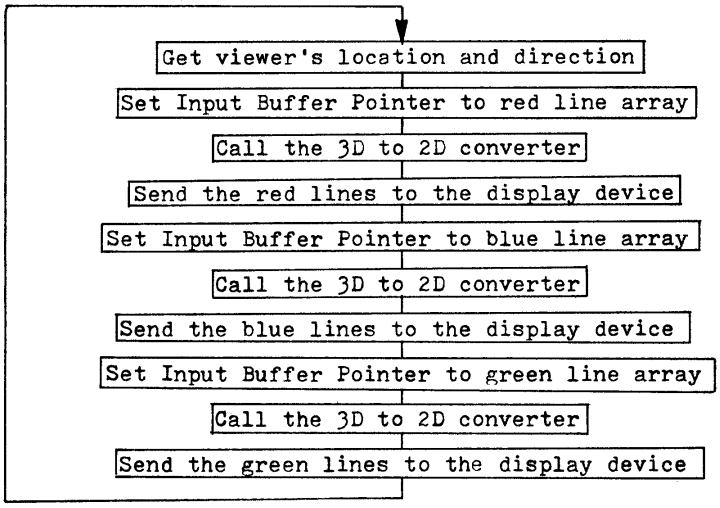


Figure b. The projection of 3 seperate arrays

MISCELLANEOUS TOPIC - - - TRANSFORMATION MATRIX DERIVATION

The 3D graphics program "rotates the world" by multiplying each point in the data base by a transformation matrix, as described in the graphics principles section. This transformation matrix is actually a concatenation of three matricies. These matricies would rotate the world about the X, Y, and Z axes if applied seperately. The concatenated matrix performs all three rotations simultaneously. The order of matrix concatenation is very important. In the 3D graphics package the heading matrix, pitch matrix, and finally the bank matrix are applied.

The matrix concatenation mathematics will now be shown.

The following symbols will be used:

SP= Sine (Pitch) CP= Cosine (Pitch)
SB= Sine (Bank) CB= Cosine (Bank)
SH= Sine (Heading) CH=Cosine (Heading)

The pitch matrix "P" is:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & CP & -SP \\ 0 & SP & CP \end{bmatrix}$$

The bank matrix "B" is:

$$B = \begin{bmatrix} CB & -SB & 0 \\ SB & CB & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The heading matrix "H" is:

$$H = \begin{bmatrix} CH & 0 & SH \\ 0 & 1 & 0 \\ -SH & 0 & CH \end{bmatrix}$$

Concatenating the P and B matricies:

$$PB = \begin{bmatrix} CB & -SB & 0 \\ SBCP & CBCP & -SP \\ SPSB & SPCB & CP \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & CP & -SP \\ 0 & SP & CP \end{bmatrix} X \begin{bmatrix} CB & -SB & 0 \\ SB & CB & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Concatenating the H and PB matricies results in the final transformation matrix "T":

$$T = HPB = \begin{bmatrix} CHCB + SHSPSB & -CHSB + SHSPCB & SHCP \\ SBCP & CBCP & -SP \\ -SHCB + CHSPSB & SBSH + CHSPCB & CHCP \end{bmatrix} = \begin{bmatrix} CH & 0 & SH \\ 0 & 1 & 0 \\ -SH & 0 & CH \end{bmatrix} X \begin{bmatrix} CB & -SB & 0 \\ SBCP & CBCP & -SP \\ SPSB & SPCB & CP \end{bmatrix}$$

This is the same transformation shown in the graphics principles section.

MISCELLANEOUS TOPIC - - - ACCURATE TRIG

The BASIC Microcomputer Graphics Package does not require any BASIC trigonometric functions because it has an internal sine/cosine generator subroutine. If a BASIC interpreter with trig functions is used, the transformation matrix can be generated with much higher precision. The slight bit of accuracy gained won't make very much difference in the final projection, but it will simplify the program and help satisfy perfectionists who want everything to be as precise as possible.

The image projected using the more accurate transformation matrix will be identical, geometrically, to the old image. The improvement gained will be in the direction of view precision. For example, instead of looking at an object with a viewing direction of 29.995 degrees pitch, .003 degrees bank and 45.008 degrees heading, you will now be able to see the world from the desired 30 degrees pitch, 0 degrees bank and 45 degrees heading. Figure a illustrates the new transformation matrix generator. The Sine/Cosine generator subroutine in the BASIC 8300-8380 block can be eliminated if this matrix generator is used.

```
8200  R1=SIN (P*1.7453292E-2)
8206  R2=COS (P*1.7453292E-2)
8215  R3=SIN (B*1.7453292E-2)
8222  R4=COS (B*1.7453292E-2)
8230  R5=SIN (H*1.7453292E-2)
8237  R6=COS (H*1.7453292E-2)
8245  T1=R4*R6+R5*R1*R3
8248  T2=-R6*R3+R5*R1*R4
8251  T3=R5*R2*V
8253  T4=R2*R3
8256  T5=R2*R4
8259  T6=-R1*V
8262  T7=-R5*R4+R6*R1*R3
8265  T8=R5*R3+R6*R1*R4
8269  T9=R2*R6*V
8272 RETURN
```

Sin/Cos Calculation

Transformation Matrix Generation

Return

Figure a. High accuracy matrix generator

MISCELLANEOUS TOPIC - - - BASIC SPEED-UPS

Speeding up a BASIC graphics program to perform like an assembly language version is not possible. Assembly language has a 100 to one speed advantage over the BASIC package, but BASIC has one very desirable feature; it is extremely accurate. It is therefore well suited to large, complex, static displays used in architectural design, art, engineering, etc. It may take minutes to generate one frame so even a 50% speed improvement will save the user a minute or two for each display frame. A few speedup methods which can be employed in many BASICs will now be presented. Make sure to check your particular BASIC manual before trying them. They may not help or even work with your particular BASIC.

1. Remove the REM statements or make sure program execution jumps around them. You don't want BASIC wasting time deciding not to execute a comment.

2. Don't use LET if your BASIC accepts statements without it. It just takes up memory and wastes interpretation time.

3. Type all statements without spaces. Some interpreters waste time skipping over spaces. For example, type:

```
IFA=BTHENGOTO80
```

instead of:

```
IF A=B THEN GOTO 80
```

The listing may look strange but as long as you have a nicely formatted listing to back it up, that's alright.

MISCELLANEOUS TOPIC - - - ADVANCED ARRAY STORAGE

The BASIC Microcomputer Graphic Package is very precise and is well suited to intricate scenes with many edges. Storing large data bases in BASIC arrays however is very memory-wasteful. BASIC usually assigns large BCD values for each array entry. A method that may be used to cut array storage size by up to 90% will now be described.

If your BASIC interpreter has PEEK and POKE instructions, you can store a data base in memory in a double precision integer format. Only two 8 bit bytes must be used for values with a range of ± 32767 units. A 3D line's start and end points can therefore be stored in 12 bytes. In other words, 341 lines can be stored in every free 4K block of memory.

A floating point value can be put into 2 8 bit bytes in the following way:

```
M=X/256
M=INT (M)
L=X-(256*M)
```

where

X= floating point value to be split (where X is positive)
M= most significant byte
L= least significant byte

These values can now be POKEd into memory. Converting data back to floating point is even easier. The following sequence will calculate floating point X.

```
X= (PEEK (A)*256) + (PEEK (A+1))
```

where

A= address of first byte stored

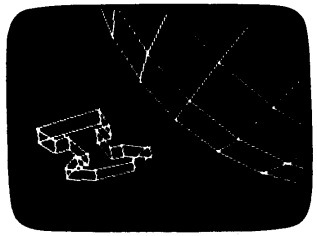
Two processor-system dependent things must be worked out before this scheme can successfully be used. First, an address handling program must be written. You must know where in memory to place the array. Overwriting data into program area or the BASIC interpreter can be disastrous. Sequencing through memory must also be controlled.

The other thing which must be worked out is a way to express negative values. Probably the best way to handle this is to store all values as positive integers and add offset values when the value is back in floating point.

Box V, Savoy, IL 61874

1979 Catalog
Computer Graphics from subLOGIC

The engineering & graphics software people



of start and enpoints of lines to be displayed on the screen. The user's software takes it from there and sends the output array data to the screen. Display hardware and software should be capable of drawing lines, erasing the display screen, and sending data to the display device. To assist the user in interfacing, the graphics package contains information on line drawing methods and presents interface examples for common devices.

Program Media

The BASIC 3D package contains a BASIC listing of the program only. It has been optimized for size. The 6800 assembly language comes with a Mikbug/ Kansas Cassette. The 8080/Z80 versions come with paper tape or TDL relocatable object code on a Tarbell cassette. A loader for the TDL format is also provided. Both assembly language versions come with hex listings for those users not having the supplied media. No source listings are provided with the assembly language packages.

Display Devices

The only display requirement for the 3D packages is that you must have a device which your computer can somehow draw lines on. It can be a terminal, plotter, graphics display, or anything else. A powerful high-resolution device is convenient, but a device as simple as a 64x64 dot matrix will work also.

A Note to TRS-80, PET, and Apple II Owners:

The output of programs is x,y coordinates of line endpoints.

TRS-80: Your 128x48 display allows pleasant, simple street, architecture, and space scenes. You must create a subroutine to draw lines between endpoints. An algorithm is given in the manuals for this task. If you are uneasy about Z80 programming, then we recommend the BASIC version (Level II required).

PET: Most PET owners use the BASIC version to obtain line endpoints for manual graphing. A variable screen width feature allows you to scale the output according to the units desired.

APPLE II: You must be able to use Floating Point BASIC and the high-resolution graphics routine simultaneously. Our marketing department uses an Apple II with Applesoft ROM card for BASIC program demonstration. The high resolution of the Apple permits a beautiful presentation of complex scenes.

Load and Go Packages

The Dazzler and Matrox ALT-256**2 programs are written for instant use. Transformed scenes are displayed on the Dazzler or Matrox display with no user programming necessary. Our engineering department calls them "bullet proof."

It will be easy for you to use the programs with the step-by-step manual. A sample data base and test program are provided for quick familiarization.

The programs are not ROMable or relocatable, nor can they be used with other display devices, but they include many valuable features:

- The program may be controlled from assembly language or BASIC.
- On the Matrox, adaptive screen erase speeds up display and minimizes flicker.
- The Dazzler has double buffering to minimize flicker.

Relocatable loader and object code for Tarbell cassette or keyboard loading are available as options.

New Load and Go Graphics for Apple II with Applesoft ROM

Includes BASIC manual plus programs with interface and sample data base on Applesoft II cassette. Load and Go program and documentation by Jim Harter. Introductory price (until March 1, 1979) $26. ($28 thereafter.)

2D Drivers

SubLOGIC has received many requests for Interpreters (interface programs, drivers) to aid 3D experimenters' packages. Our engineering department has now gone one step beyond: we have interface programs for the Dazzler, Matrox ALT-256, and Vector Graphics high resolution displays that provide many convenient features for all 2D display users:

- Screen erase
- Draw-a-line
- Plot-a-point
- Continue-a-line
- Ray (draw a ray from a point, then continue)
- Relative line capability (relocate whole series of lines, i.e., relocate or move a whole object)
- Chain lines
- Skip-and-jump lines
- Shaded polygons
- Circles and shaded circles

Perhaps the most powerful aspect of the 2D programs is that they allow you to build universal data bases (your Dazzler images can be projected on any display device or plotter you eventually upgrade to).

The interpreter includes a printed relocatable object code and either TDL format Tarbell tape or paper tape (a relocatable loader is included for TDL format tape).

Whether you use 3D or 2D graphics, you'll find that the 2D interpreters are invaluable for your applications.

- 66 page manual
- Specify Matrox, Dazzler, or Vector Graphic High Resolution Board

NO SOURCE $20 U.S. and Canada, $23 Foreign

Hardware

We have been getting many requests for information about S100 display devices. With the introduction of our 8080/Z80 package we thought it important, therefore, to evaluate current display devices to see which is best for 3D graphics applications. These characteristics are of prime importance: screen access time, erase speed, interface requirements, microprocessor bus overhead, and cost. The microprocessor bus overhead requirement eliminated most of the DMA devices from consideration, at least in dynamic applications. The board which seemed to meet all of our requirements was the Matrox ALT-256**2, by Matrox Electronic Systems, Montreal, Canada. This is a 256 x 256 bit map with its own internal 65K bit memory. It is S100 compatible and is preassembled, tested, and configured. We were also impressed by the board's clean design (no variable resistors or other adjustment devices) and its quality construction. A screen erase command feature allows quick screen erasing, something that takes a long time under software control.

We are now offering the Matrox ALT-256**2 card to our customers. Interfacing this card to the 8080/Z80 package is trivial since the interface example in the manual is for this card. The price of this card is higher than for DMA display cards because of the onboard 65K memory and the preassembly, but the increase in performance is well worth it. If you would like a copy of the 42-page manual for the Matrox board, send $3.00; the price can be deducted from your order for the board.

Specifications

On the following four pages are the specifications for all of our 3D software packages as well as for the Matrox ALT-256**2 display board we offer.

Please note that while we assume that most of our customers will be using microcomputer hardware, any computer with BASIC can accept the universal BASIC 3D program.

BASIC LANGUAGE

Program Number	3DGBU.VO3
Program Language	Minimal Set BASIC (let, for-next, =, +, −, /, *, if-then, 1 dimensional arrays, goto, gosub, return). No trig functions are needed.
Projection Method	3D to 2D wire frame perspective with 3D clipping. No hidden line elimination capabilities.
Viewing Range	X,Y,Z range of BASIC (floating point range) 3 axis freedom: 0 to 359.999 degrees.
Special Features	Variable viewing window (telephoto/wide angle) Universality due to BASIC.
Memory Requirements	Depends on BASIC efficiency. A 12K system should be adequate in most cases.
Package Contents	3D to 2D converter programs in BASIC, Usage information, Application notes, Interfacing information, Test procedures, Algorithm description, 60 page manual.
Price	$22.00 U.S. and Canada $25.00 Foreign (to cover shipping)

M6800 ASSEMBLY LANGUAGE

Program Number	3DG68.V3.1
Program Language	Optimized 6800 assembly language
Projection Method	3D to 2D wire frame perspective with 3D clipping. No hidden line elimination capabilities.
Viewing Range	X,Y,Z range: ±32767 units. 3 axis freedom: 0 to 359 degrees in 256 even steps
World Size	1912 cubic miles using one foot/unit resolution
Special Features	Variable viewing window Optimized clipping High rate dynamic capabilities
Memory Requirements	4K plus data base
Projection Rate	100-300 lines per second (1 MHz 6800)
Package Contents	3D to 2D converter program, Object listing, Mikbug/Kansas cassette, Usage information, Application notes, Interfacing information, Test procedures, Algorithm procedures, Familiarization section, Sample data base, 74 page manual. NO SOURCE.
Price	$28.00 U.S. and Canada $31.00 Foreign (to cover shipping)

8080-Z80 ASSEMBLY LANGUAGE

Program Number	3DG80.V03 (8080)
Program Language	Optimized 8080 and Z80 assembly language
Projection Method	3D to 2D wire frame perspective with 3D clipping. No hidden line elimination.
Viewing Range	X,Y,Z range: ±32767 units. 3 axis freedom: 0-359 degrees in 256 even steps
World Size	1912 cubic miles using one foot/unit resolution
Special Features	Variable aspect ratio, Variable screen bit ratio, Optimized clipping, 10 data base entry modes, Relocatable object mode, ROM-ability, High rate dynamic capabilities.
Memory Requirements	5K plus data base
Projection Rate	200-500 lines per second (4MHz 8080)
Package Contents	3D to 2D converter program, Object listing, TDL relocatable format Tarbell cassette, Usage information, Relocatable loader, Relocatable keyboard loading method, Application notes, Test procedures, Algorithm description, Familiarization section. 85 page manual. NO SOURCE.
Price	$30.00 U.S. and Canada $33.00 Foreign (to cover shipping)

MATROX AND DAZZLER 3D LOAD AND GO

	MATROX	DAZZLER
Program Number	3DG MTX LG	3DG DAZ LG
Program Language	8080 or Z80	8080 or Z80
Projection Method	3D to 2D wire frame perspective with 3D clipping. No hidden line elimination.	3D to 2D wire frame perspective with 3D clipping. No hidden line elimination.
Viewing Range	X,Y,Z range: ± 32767 units. 3 axis freedom:0-359 degrees in 256 even steps.	X,Y,Z range: ± 32767 units. 3 axis freedom: 0-359 degrees in 256 even steps.
World Size	1912 cubic miles using one foot/ unit resolution.	1912 cubic miles using one foot/ unit resolution.
Special Features	Variable aspect ratio, Optimized clipping, 4 projection modes, High rate dynamic capabilities, Adaptive erase.	Variable aspect ratio, Optimized clipping, 4 projection modes, High rate dynamic capabilities, High speed erase.
Memory Requirements	5K plus data base	5K plus data base
Location in RAM	1000-2200 hex	1000-3200 hex
Projection Rate	200-500 lines per second	150-400 lines per second
Package Contents	3D to 2D converter program, Tarbell format, Tarbell cassette or paper tape (specify medium), Test procedures, Familiarization section 26 page manual NO SOURCE	3D to 2D converter program, Tarbell format, Tarbell cassette or paper tape (specify medium), Test procedures, familiarization section 26 page manual NO SOURCE
Price	*$15.00 U.S. and Canada **$18.00 Foreign (Airmail)	*$15.00 U.S. and Canada **$18.00 Foreign (Airmail)
Options	Relocatable Loader and Object Relocatable Tarbell cassette $10.00 Keyboard entry $10.00 Non Relocatable program on North Star Disk $10.00	

* Introductory until January 31, 1979
Then $25.00.

** Introductory until January 31, 1979
Then $28.00.

MATROX ALT-256**2 DISPLAY (Hardware)

Display Type	256 x 256 raster scan bit map with onboard refresh memory
Dot Write Time	3.4 usec max, faster in common row or col. modes.
Erase Time	33 milliseconds using fast erase command.
Interface	S100 Altair/Imsai compatible.
Dimensions	9"x 5". Slightly taller than most S100 cards.
Power	8v, 600mA; -18v, 10mA
Outputs	Composite video; 75 Ohm, x tal controlled; TTL video, horizontal and vertical syncs and blank outputs
Synchronization	Internal or external
TV Standard	American standard (262 vertical lines, 60 Hz; 240 vertical video lines) 4:3 aspect ratio; American Non-standard (280 lines, 60Hz, 256 video lines), horizontal freq=16.8 KHz (1:1 aspect ratio); European (312, 50Hz, 1:1 aspect ratio). Non-interlaced picture. Standard selectable on the board.
Monitor	Any standard TV monitor or modified TV set.
Remote Display	75 Ohm, up to 2500 ft. Multiple monitors (max=25).
Addressing	Four output ports, one input port. Ports selectable.
Documentation	42 page manual, complete description, schematics, test program, and application notes.
Warranty	90 days parts and labor
Price	$395.00 postage paid U.S. and Canada $410.00 Foreign
More Information	For more information, send $3.00 for a copy of the manual, application notes, and schematics (42 pp.)
Note:	We have successfully used the ALT 256**2 with a 4MHz Z80 system. It worked well with a Jade (Ithica Audio) Z80 card and TDL 2MHz Z80 card also.

Ordering Information

Orders in the U.S. and Canada

Special Fourth Class postage is paid by SubLOGIC for orders in the U.S. and Canada. Delivery time averages 10 days.

For First Class mailing, add $1.50 per graphics package. Delivery time averages 3 days.

For UPS shipment, add $.75 per graphics package. up to a maximum of $4.00 per order. Delivery time averages 4 days.

For COD orders via UPS, add $.75 per graphics package, plus $2.00 for handling, up to a maximum of $6.00 per order.

When ordering a Matrox board, send a money order, cashier's check, or any other form of prepaid check. Personal checks may delay your order for up to two weeks.

Orders paid by charge card or COD may be placed by phone. Call (217) 367-0299.

To wire payment, send to the Champaign, Illinois, Western Union Office and include our telephone number.

Foreign Orders

Add $3.00 extra per graphics package for airmail shipment.

Our cable address is "SUBLOGIC."

We are pleased to accept orders in both English and French.

Who We Are

The use of computer graphics in science, engineering, art, and many other fields has been increasing over the past 15 years. Display devices continue to drop in price as many manufacturers enter the market with new hardware. The advent of the microcomputer and MOS-LSI memories has brought graphics systems down to a very affordable level, and a system costing one or two thousand dollars can have very advanced graphics capabilities.

SubLOGIC was therefore formed in 1977 to develop software that would allow experimenters, engineers, architects, designers, pilots, and anyone else interested in three-dimensional wireframe drawings or spatial viewing to generate what they wanted on a TV screen or paper plotter. The company is headed by Bruce Artwick and Stuart Moment and supported by a team of marketing and production people who specialize in reliable service.

What We Sell

SubLOGIC's first efforts resulted in a number of 3D microcomputer graphics packages which can be used with the new inexpensive hardware systems. We have now branched out into simpler-to-handle Load and Go packages, 2D graphics software, and display boards as well.

Reliability

SubLOGIC software is combinatorially segment-tested to insure maximum durability. SubLOGIC strives for the highest standards of quality, and an ongoing quality control program helps meet the goal.