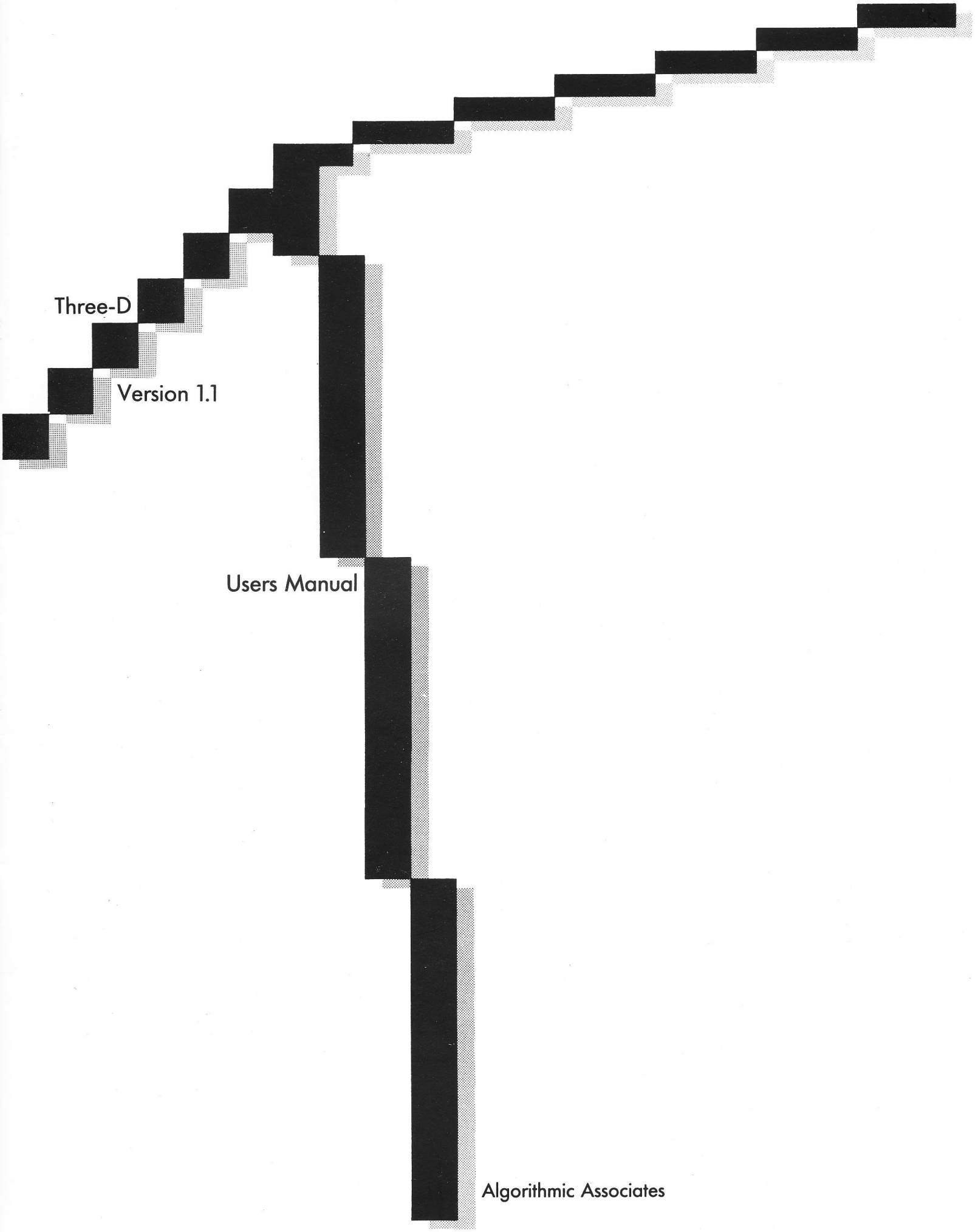

Three-D
Version 1.1
Users Manual
Algorithmic Associates

3D USERS MANUAL

Version 1.1

A three-dimensional graphic animation package
for the TRS-80*

Algorithmic Associates extends thanks to Kari Denzau, Carnegie-Mellon University, for the cover design.

3D was designed and implemented by Tom Konchan, Harvard University. The hidden line removal algorithm was designed with assistance from Aviel Klausner, Harvard University.

Algorithmic Associates
PO Box 244
Bedford, MA 01730

---

*TRS-80 is a trademark of the Tandy Corporation.

Developed and distributed by:

Algorithmic Associates
PO Box 244
Bedford, MA 01730

DISCLAIMER

ALGORITHMIC ASSOCIATES makes no representations or warranties with respect to the contents of this Users Manual or with respect to the software package described herein, and specifically disclaims any implied warranties with respect to its quality, performance, merchantability, or fitness for any particular use.

TABLE OF CONTENTS

1. INTRODUCTION

3D gives your TRS-80 the power of three-dimensional animation. 3D extends BASIC (or Assembly Language) with high-level primitives that are powerful and easy to use. These primitives enable you to quickly create programs for user-interactive three-dimensional animation applications. 3D has several features:

Graphics

3D enables you to define three-dimensional and two-dimensional objects. These objects can be scaled, rotated, displaced, and projected onto the CRT. Any number of objects may be displayed simultaneously. Objects are displayed as aesthetically as possible using the highest degree of resolution available on the TRS-80.

Animation

3D provides for smooth movement of objects. The average animation speed is from 5 to 20 frames per second depending upon the complexity of the objects and the program.

Easy to Use

3D is easy to understand because it supports consistent, intuitive representations of objects and their movement. Definitions of the high-level primitives in 3D are logical and completely documented. Examples of how to use 3D are given in simple, but extensive, demonstration programs. Programs using 3D are short and are developed quickly.

Power

3D performs fast, three-dimensional animation in a manner that utilizes the resources of the TRS-80 productively.

User Interactive

3D supports interactive applications programs such as those that allow end-users to define and manipulate objects during program execution.

Efficient

3D is efficient in three aspects: (i) programming efficient - it helps you to develop programs rapidly, (ii) time efficient - 3D is written in Assembly Language; it executes quickly, and supports non-flickering animation of multiple,

complex objects, and (iii) space efficient - 3D primitives, including space for definitions of objects, occupy about 5,000 bytes of memory, and calling programs tend to be short due to 3D's high-level primitives.

Basic Extension

All the features of BASIC are still available. 3D gives you all the power of three-dimensional animation without having to learn a new programming language. 3D can, also be used from Assembly Language for those applications requiring the utmost in space and time efficiency.

3D is mastered quickly by those familiar with BASIC. First-time users should read Section 4, "Loading 3D From Tape" and should run the four demonstration programs shown there. The remaining sections, except Section 6, "Using 3D from Assembly Language," should be read before writing your first program (or before altering the demos -- a quick way of becoming acclimated to 3D).

2. 3D CONCEPT

3D relies on eight high-level primitives to accomplish graphic animation. These primitives are: INITIALIZE, DEFPOINT, DEFSURFACE, DEFLINE, CLEAR, PLACE, DISPLAY, and DRAW. These primitives are defined in Section 3, but an intuitive explanation of their use is given here.

Each primitive requires several input parameters. The parameters are BASIC variables that can be assigned values in the normal ways supported by BASIC. For example, the DEFPOINT primitive has three parameters, X1%, Y1%, and Z1%, corresponding to the X-coordinate, Y-coordinate, and Z-coordinate of the point being defined. Here X1%, Y1%, and Z1% are BASIC variables of type integer. Values are assigned to these variables prior to calling DEFPOINT. The result of the call is a new point added to the object being defined.

The 3D concept is easily described using an example. Consider a BASIC program that defines and manipulates a cube. The program begins by calling the INITIALIZE primitive. This primitive initalizes 3D and builds a symbol table so that 3D can find the BASIC variables used as parameters. Next, the program defines the cube using the three definition primitives: DEFPOINT, DEFSURFACE, and DEFLINE. The definition of the cube is performed by: (i) calling DEFPOINT 8 times, each time defining a different corner point on the cube, then (ii) calling DEFSURFACE 6 times, each time defining a different surface, finally, (iii) calling DEFLINE 12 times to define the lines that connect the corners of the cube.

Now the cube can be manipulated. First, the position, rotation, and scale parameters are set. Then the CLEAR primitive is used to clear the virtual screen -- an area in memory where the image is created. Next, the PLACE primitive is used to place the cube on the virtual screen with the orientation and at the position specified by the parameters. Finally, the DISPLAY primitive is used to copy the contents of the virtual screen to the CRT where it can be seen. Although the use of a virtual screen adds an extra step to the program, its use guarantees flicker-free animation.

At this point, the program is displaying one frame of the animation. To move the cube, the position and orientation parameters for the object are

changed and then control must loop back to the step in which CLEAR was called. Put simply, the 3D concept is as follows:

STEP	ACTION
1	INITIALIZE 3D
2	DEFine one or more objects
3	Set the position and orientation parameters for the object
4	CLEAR the virtual screen
5	PLACE one or more objects on the virtual screen
6	DISPLAY the virtual screen
7	Change the position and orientation parameters
8	GOTO Step 4

Because the parameters are BASIC variables, they can be altered in any of the ways supported by BASIC. For example, the BASIC statements INPUT and INKEY may be used in the program to allow the end-user of the program to interact with the animation in real-time. Several demonstrations included in this manual illustrate this point.

The BASIC statements PRINT and SET can be used to augment the animation with text and additional detail. The BASIC USR function is used to make the calls to the 3D primitives.

So far we have discussed the use of 3D from BASIC. But 3D can also be used from Assembly Language. In this case, the same primitives in the same order are used. The only difference is the interfacing technique. In this case, the parameters are contained in single-byte, labeled locations and are directly accessible. Calls to 3D are via the Assembly Language CALL instruction.

There is one primitive that hasn't been discussed -- DRAW. This primitive accepts 2 points as input and draws a line connecting them on the virtual screen. This primitive is useful for drawing backgrounds or otherwise augmenting an animation scene.

3. 3D PRIMITIVES

The 8 primitives that make up 3D are described in detail in this section. The organization of this section is to facilitate its usage as a reference.

Figure 3.1 shows the parameters used by the primitives. When using the parameters from BASIC, they are integer variables. When using them from Assembly Language, they are single-byte, labeled locations.

Figure 3.1 3D Parameters

Name	Description	Units	Legal Range
HRZ%	Horizontal displacement of object	Pixels	0 to 255
VRT%	Vertical displacement of object	Pixels	0 to 255
TH%	Theta - rotation around X-axis	Degrees x 5	-128 to 127
PH%	Phi - rotation around Y-axis	Degrees x 5	-128 to 127
PS%	Psi - rotation around Z-axis	Degrees x 5	-128 to 127
SC%	Scale of object	None	0 to 127
OBJ%	Object number	None	1 to 10
X1%	X-coordinate	Pixels	-128 to 255
Y1%	Y-coordinate	Pixels	-128 to 255
X2%	X-coordinate	Pixels	0 to 255
Y2%	Y-coordinate	Pixels	0 to 255
Z1%	Z-coordinate	Pixels	-128 to 127
P1%	Point number	None	1 to 85
P2%	Point number	None	1 to 85
P3%	Point number	None	1 to 85
S1%	Surface number	None	1 to 64
S2%	Surface number	None	1 to 64

Error checking is performed by the primitives, but is not described in this section. For a description of error checking refer to Section 7.

Before presenting the 3D primitives, it is appropriate to define what is meant by an object. An object is a group of defined points, surfaces, and lines which can be manipulated by 3D. Each object must have at least one point, one surface, and one line. For example, the simplest object is a single point with a line having that point as both its end-points, and having a surface defined by that point. This will be more clear as the primitives become understood.

INITIALIZE

Input:	None
Function:	To prepare for use of 3D primitives
Notes:	Must be called only once prior to use of any other 3D primitive
Limitations:	None
Call From BASIC:	
	All versions: GOSUB 65000
Call From Assembly Language:	
	16K Version: Call 7F5FH
	32K Version: Call BF5FH
	48K Version: Call FF5FH

INITIALIZE is the only primitive written in BASIC. It is written in BASIC because a symbol table must be defined using VARPTR to tell 3D the locations of the parameters (i.e., the BASIC variables).

When calling 3D from Assembly Language, a different version of initialization is required. Both versions of INITIALIZE are described here.

The BASIC version of INITIALIZE (see Figure 3.2) is a subroutine that is called as early in the program as possible. This assures the parameters are placed at the beginning of the BASIC variable stack, resulting in efficient use of the parameters. INITIALIZE must be called once prior to usage of any other 3D primitive. It should not be called thereafter.

When calling 3D from Assembly Language, the Assembly Language version of INITIALIZE is used (see Figure 3.3). Two actions occur during initialization in this case: (i) labels are equated to certain required locations in 3D, and (ii) a call to the Assembly Language initialization routine is made.

Figure 3.2 BASIC INITIALIZE Primitive

(16K Version*)

```
65000 'INITIALIZATION
                This subroutine is called once prior to
                using9 the 3D Primitives.  For effeciency,
                it is called as early as Possible.

65002 HRZ%=0:VRT%=0:TH%=0:PH%=0:PS%=0:SC%=0:OBJ%=0:X1%=0:Y1%=0
65004 X2%=0:Y2%=0:Z1%=0:P1%=0:P2%=0:P3%=0:S1%=0:S2%=0:PTR%=-2
65006 A%=VARPTR(HRZ%):GOSUB65044
65008 A%=VARPTR(VRT%):GOSUB65044
65010 A%=VARPTR(TH%) :GOSUB65044
65012 A%=VARPTR(PH%) :GOSUB65044
65014 A%=VARPTR(PS%) :GOSUB65044
65016 A%=VARPTR(SC%) :GOSUB65044
65018 A%=VARPTR(OBJ%):GOSUB65044
65020 A%=VARPTR(X1%) :GOSUB65044
65022 A%=VARPTR(Y1%) :GOSUB65044
65024 A%=VARPTR(X2%) :GOSUB65044
65026 A%=VARPTR(Y2%) :GOSUB65044
65028 A%=VARPTR(Z1%) :GOSUB65044
65030 A%=VARPTR(P1%) :GOSUB65044
65032 A%=VARPTR(P2%) :GOSUB65044
65034 A%=VARPTR(P3%) :GOSUB65044
65036 A%=VARPTR(S1%) :GOSUB65044
65038 A%=VARPTR(S2%) :GOSUB65044
65040 FORI%=0TO62:POKE28706+I%,0:NEXTI%
65042 I%=0:A%=0:A1%=0:A2%=0:PTR%=0:RETURN
65044 A1%=A%/256:A2%=A%-A1%*256:PTR%=PTR%+2
65046 POKE28672+PTR%,A2%:POKE28673+PTR%,A1%:RETURN
```

*POKE addresses in lines 65040 and 65046 are different for the 32K and 48K versions as follows:

16K	28706	28672	28673
32K	-20446	-20480	-20479
48K	- 4062	- 4096	- 4095

Figure 3.3 Assembly Language INITIALIZE Primitive

(16K Version*)

```
00001 ;INITIALIZE
00002
00003 ;EQUate the locations of the 3D Primitives
00004
00005 INIT    EQU     7F5FH   ;Initialize Primitive
00006 DEFPNT  EQU     7061H   ;Define Point Primitive
00007 DEFSUR  EQU     70C1H   ;Define surface Primitive
00008 DEFLIN  EQU     716CH   ;Define line Primitive
00009 CLEAR   EQU     7B08H   ;Clear Primitive
00010 PLACE   EQU     77CAH   ;Place Primitive
00011 DISPLA  EQU     7B16H   ;DisPlay Primitive
00012 DRAW    EQU     7B22H   ;Draw Primitive
00013
00014 ;EQUate the locations of the 3D Parameters
00015
00016 HRZ     EQU     7F4EH   ;Horizontal
00017 VRT     EQU     7F4FH   ;Vertical
00018 TH      EQU     7F50H   ;Theta
00019 PH      EQU     7F51H   ;Phi
00020 PS      EQU     7F52H   ;Psi
00021 SC      EQU     7F53H   ;Scale
00022 OBJ     EQU     7F54H   ;Object
00023 X1      EQU     7F55H   ;X-coordinate
00024 Y1      EQU     7F56H   ;Y-coordinate
00025 X2      EQU     7F57H   ;X-coordinate
00026 Y2      EQU     7F58H   ;Y-coordinate
00027 Z1      EQU     7F59H   ;Z-coordinate
00028 P1      EQU     7F5AH   ;Point
00029 P2      EQU     7F5BH   ;Point
00030 P3      EQU     7F5CH   ;Point
00031 S1      EQU     7F5DH   ;Surface
00032 S2      EQU     7F5EH   ;Surface
00033
00034 ENTRY   CALL    INIT    ;Call the INIT Primitive
00035
```

*Add 4000H to all EQU addresses for the 32K version. Add 8000H to all EQU addresses for the 48K version.

DEFPOINT

Input:	OBJ%, X1%, Y1%, Z1%
Function:	Add the point (X1%, Y1%, Z1%) to the definition of object OBJ%.
Notes:	All points for an object must be defined by consecutive calls on DEFPOINT. That is, DEFPOINT should not be called to define a point on object 1, then a point on object 2, then back to define another point on object 1. However, calls to other 3D primitives may occur between the consecutive calls on DEFPOINT.
Limitations:	The total number of points for all objects combined must be 85 or less.
Call From BASIC:	
	16K Version: POKE 16526,97: POKE 16527,112: ER%=USR(0)
	32K Version: POKE 16526,97: POKE 16527,176: ER%=USR(0)
	48K Version: POKE 16526,97: POKE 16527,240: ER%=USR(0)
Call From Assembly Language:	
	16K Version: CALL 7061H
	32K Version: CALL B061H
	48K Version: CALL F061H

DEFPOINT adds the point (X1%, Y1%, Z1%) in three-dimensional Cartesian space (see Appendix B) to the definition of object OBJ%. As the point is defined, it is implicitly numbered. Numbering begins with 1 for each object. For example, 8 consecutive calls on DEFPOINT for the same object results in the definition of 8 points numbered 1 through 8. If DEFPOINT is then called 4 more times for a different object, then that object will have 4 defined points numbered 1 to 4. The implicit number, or point number as it is called, is used to reference the point later when using DEFSURFACE and DEFLINE.

Because pixels on the TRS-80 are twice as tall as they are wide, a square (say 5 pixels x 5 pixels) would appear as a rectangle that is twice as high as it is wide. To further complicate the problem, rotations of the object will nullify any attempt to correct this problem. To see this, consider defining

a rectangle that is twice as wide as it is high. When placed on the screen, it would appear square -- just what we want. But, if the object is rotated 90^{o} it becomes a rectangle, only this time it is 4 times as high as it is wide! To solve this problem, 3D has a built-in adjustment that is applied after an object is rotated. With this built-in adjustment, an object defined as square will appear square on the CRT, no matter what the rotation. But, if its dimensions were measured by counting pixels, the X-dimension would be twice the Y-dimension.

DEFSURFACE

Input:	OBJ%, P1%, P2%, P3%
Function:	Add the surface specified by P1%, P2%, and P3% to the definition of object OBJ%.
Notes:	All surfaces for an object must be defined by consecutive calls on DEFSURFACE. Points P1%, P2%, and P3% must be the numbers of points defined for object OBJ%. For hidden line removal to function properly, P1%, P2%, and P3% must be oriented clockwise with respect to one another as viewed from outside the object looking in at the surface.
Limitations:	The total number of surfaces for all objects combined must be 64 or less.
Call From BASIC:	
16K Version:	POKE 16526,193: POKE 16527,112: ER%=USR(0)
32K Version:	POKE 16526,193: POKE 16527,176: ER%=USR(0)
48K Version:	POKE 16526,193: POKE 16527,240: ER%=USR(0)
Call From Assembly Language:	
16K Version:	CALL 70C1H
32K Version:	CALL B0C1H
48K Version:	CALL F0C1H

DEFSURFACE adds the planar surface defined by the three-point numbers, P1%, P2%, and P3% to the definition of object OBJ%. As the surface is defined,

it is implicitly numbered, with the numbering beginning at 1. The implicit number, or surface number as it is called, is used to reference the surface later when using DEFLINE.

A planar surface is sufficiently defined by specifying any 3 points that lie on it. Here we conveniently choose previously defined corner points as P1%, P2%, and P3%. For a simple object such as a single point, we simply use that point for each of P1%, P2%, and P3%.

To ensure that hidden line removal functions properly, points P1%, P2%, and P3% must be oriented clockwise with respect to one another as viewed from outside the object. For a surface that has many points defined on it, choose 3 points that do not form a straight line. Additional information regarding hidden surface removal is contained in Section 8.

DEFLINE

Input:	OBJ%, P1%, P2%, S1%, S2%
Function:	Add the line segment specified by endpoints P1% and P2% to the definition of object OBJ%. Surfaces S1% and S2% are the two bordering surfaces and are required for proper hidden line removal.
Notes:	All lines for an object must be defined by consecutive calls on DEFLINE. P1% and P2% must be point numbers of points defined for this object. S1% and S2% must be surface numbers of surfaces defined for this object.
Limitations:	The total number of lines (for all objects combined) must be 64 or less.
Call From BASIC:	
	16K Version: POKE 16526,108: POKE 16527,113: ER%=USR(0)
	32K Version: POKE 16526,108: POKE 16527,177: ER%=USR(0)
	48K Version: POKE 16526,108: POKE 16527,241: ER%=USR(0)
Call From Assembly Language:	
	16K Version: CALL 716CH
	32K Version: CALL B16CH
	48K Version: CALL F16CH

DEFLINE requires surfaces S1% and S2% for hidden line removal processing. Although they are normally the surfaces on either side of the line, the user has the freedom to select any surfaces defined for the object. Thus, special effects are possible. The definition of a hidden line (and, hence, one that isn't displayed on the CRT), is one that has both associated surfaces hidden from view. Note that S1% and S2% need not be unique.

CLEAR

Input:	None
Function:	Clear and initialize the virtual screen.
Notes:	This primitive should be called once prior to the first call to PLACE. Thereafter, this primitive should be called anytime the virtual screen requires clearing.
Limitations:	This primitive only clears the virtual screen. To clear the CRT, first call CLEAR, then call DISPLAY.
Call From BASIC:	
	16K Version: POKE 16526,8: POKE 16527,123: ER%=USR(0)
	32K Version: POKE 16526,8: POKE 16527,187: ER%=USR(0)
	48K Version: POKE 16526,8: POKE 16527,251: ER%=USR(0)
Call From Assembly Language:	
	16K Version: CALL 7B08H
	32K Version: CALL BB08H
	48K Version: CALL FB08H

PLACE

Input:	OBJ%, HRZ%, VRT%, TH%, PH%, PS%, SC%
Function:	To place object OBJ% on the virtual screen with X-displacement HRZ%, Y-displacement VRT%, rotation around X-axis TH%, rotation around Y-axis PH%, rotation around Z-axis PS%, and scale SC%. Those portions of the object that lie outside the virtual screen are not drawn.
Notes:	None
Limitations:	Objects being placed may be of arbitrary shape and size. However, the hidden line removal feature handles only non-overlapping, convex solids (see Section 8).
Call From BASIC:	
	16K Version: POKE 16526,202: POKE 16527,119: ER%=USR(0)
	32K Version: POKE 16526,202: POKE 16527,183: ER%=USR(0)
	48K Version: POKE 16526,202: POKE 16527,247: ER%=USR(0)
Call From Assembly Language:	
	16K Version: CALL 77CAH
	32K Version: CALL B7CAH
	48K Version: CALL F7CAH

The steps performed by PLACE are listed in order below. All rotations are positive by the left-hand rule around their associated axes. The left-hand rule is: With the thumb of the left hand aligned with the axis, the curling of the fingers point in the direction of positive rotation around the axis.

1. Rotate object by TH% x 5 degrees around X-axis.
2. Rotate object by PH% x 5 degrees around Y-axis.
3. Rotate object by PS% x 5 degrees around Z-axis.
4. Scale the object by multiplying each coordinate of each point by SC%.
5. Adjust the object for asymetric pixel size by doubling the X-coordinate of each point.
6. Displace the object's defined origin by HRZ% in the horizontal and VRT% in the vertical directions.
7. Orthogonally (parallel) project onto the virtual screen, clipping as necessary.

The coordinate system used in 3D is shown in Appendix B. Note that the positive Z-axis points out of the screen. Also, note that the virtual screen and the CRT are identical in dimension and orientation.

Proper hidden line removal is dependent upon the characteristics of, and the interaction between objects. Section 8 describes the hidden line capabilities of 3D and explains how to turn this feature on or off (default is on).

Objects are drawn using the highest resolution possible on the TRS-80. The algorithm used by 3D to draw the lines in objects yields the most aesthetic lines possible.

DISPLAY

Input:	None
Function:	Display the contents of the virtual screen by copying it to the CRT (VIDEO RAM).
Notes:	This primitive ensures flicker-free animation by performing the copy quickly.
Limitations:	None
Call From BASIC:	
	16K Version: POKE 16526,22: POKE 16527,123: ER%=USR(0)
	32K Version: POKE 16526,22: POKE 16527,187: ER%=USR(0)
	48K Version: POKE 16526,22: POKE 16527,251: ER%=USR(0)
Call From Assembly Language:	
	16K Version: CALL 7B16H
	32K Version: CALL BB16H
	48K Version: CALL FB16H

After DISPLAY has been used to display the virtual screen, the image on the CRT may be enhanced by using BASIC PRINT statements or by poking into VIDEO RAM.

DRAW

Input:	X1%, Y1%, X2%, Y2%
Function:	Draw, on the virtual screen, a line segment between the screen points (X1%, Y1%) and (X2%, Y2%).
Notes:	The points need not be in view as clipping is properly handled.
Limitations:	Lines that are drawn are never subject to hidden line removal.
Call From BASIC:	
	16K Version: POKE 16526,34: POKE 16527,123: ER%=USR(0)
	32K Version: POKE 16526,34: POKE 16527,187: ER%=USR(0)
	48K Version: POKE 16526,34: POKE 16527,251: ER%=USR(0)
Call From Assembly Language:	
	16K Version: CALL 7B22H
	32K Version: CALL BB22H
	48K Version: CALL FB22H

4. LOADING 3D FROM TAPE

There are three versions of 3D, one for each of the TRS-80 Level II/ Model III memory sizes: 16K, 32K, and 48K. The version of 3D, which you have, is specified on your cassette. The only difference between the versions is where 3D resides in memory. In each version, 3D is placed as high in memory as possible, leaving maximum space for BASIC programs. For this reason, the 16K version, for example, will also run on a TRS-80 with 32K of memory. But, in this case, 16K of memory would not be accessible to BASIC.

When 3D is loaded you must restrict BASIC from overwriting it. Respond to the "Memory Size?" question at start-up as shown in Figure 4.1.

Figure 4.1 Responses To "Memory Size?"

Your Version of 3D	Response
16K	27647
32K	44031
48K	60415

3D and several demonstration programs are stored on a single 500 baud cassette. The contents of the cassette are described in Figure 4.2

Figure 4.2 3D Cassette Tape Contents

File	Location	Name	Type	Description
1	5	THREED	SYSTEM	3D Package
2	35	"1"	BASIC	Cube Demo
3	60	"2"	BASIC	Icosahedron Demo
4	85	"3"	BASIC	House Demo
5	110	ADEMO	SYSTEM	Assembly Demo
6	120	SDEMO	Assembly	ADEMO Source Code
7	150	THREED	SYSTEM	3D Backup Copy

File 1 is 3D - a package of Assembly Language routines. It is loaded from SYSTEM mode prior to loading a demo or application program. After loading 3D, either: (i) load an Assembly Language program (such as file 5) from SYSTEM mode, or (ii) hit (BREAK) and then CLOAD a BASIC program (such as file 2, 3, or 4). An example of how to load and run the first BASIC demo follows:

0. Ensure that the computer is set to 500 baud, that the cassette is rewound, and that your recorder is set to play.
1. From BASIC type:

   SYSTEM (ENTER)

   The computer responds:

   *?
2. Type:

   THREED (ENTER)

   The computer loads 3D and responds:

   *?
3. Type:

   (BREAK)
4. The computer responds: (Model III only)

   Cass?

   Type:

   L
5. The computer responds:

   Memory Size?

   For 16K Version, type:

   27647 (ENTER)

   For 32K Version, type:

   44031 (ENTER)

   For 48K Version, type:

   60415 (ENTER)

   The computer responds:

   >
6. Type:

   CLOAD "1" (ENTER)

   The computer loads the demo and responds:

   >

7. Type:

RUN (ENTER)

The program responds:

Object 1 or 2?

8. Type:

1 (ENTER)

For more information on loading from cassette, consult the reference manuals for the computer and recorder.

File 5 is an Assembly Language demo. To run it, first load 3D from SYSTEM mode, then load file 5 from system mode. Immediately after file 5 is loaded, type:

/ (ENTER)

to begin running the demo. File 6 is the Assembly Language source code for file 5. It can be loaded and edited using the Radio Shack* Editor/Assembler (Cat. No. 26-2011).

Files 3 and 4 are additional BASIC demos that can be CLOADed and RUN in a way similar to that of file 1. Instructions for interacting with these demos is found by LISTing the first lines of each program.

---

*Radio Shack is a registered trademark of the Tandy Corporation.

5. USING 3D FROM BASIC

Using 3D from BASIC is best explained by walking through an example program. The example we use is the first demonstration program on your cassette (see Figure 5.1 for a listing of this program). This example demo shows no error checking. As this is a debugged program, it had the error-checking statements removed for efficiency. See Section 7 for an explanation of how to use error checking.

Program line 20 calls the INITIALIZE primitive. The BASIC CLEAR instruction isn't used in this program; if it was used, it must come before the call to INITIALIZE.

Program line 40 sets up a loop for defining 2 objects -- a cube and a corner-cube (a cube with one corner flattened). The first time through the loop object 1 is defined. The second time through, object 2 is defined.

Program lines 50-220 define the points, surfaces, and lines for each object. The data for the objects is read from the DATA statements in lines 1000-1690. To help visualize the definition technique, refer to Figure 5.2 while examining program lines 50-220 line-by-line. Object 2 is defined such that its origin is in the center of the corner-cube. This is very convenient as rotation of this object will not cause it also to be displaced (rotations are with respect to the object's origin).

Program line 280 sets 3D parameters in preparation for entering the animation loop. HRZ%=128 and VRT%=128 causes the object to be displayed in the middle of the CRT. Program lines 310-330 are the heart of the animation loop. They call the CLEAR, PLACE, and DISPLAY primitives. Program line 335 prints a message on the CRT after the current frame of the animation has been displayed. Program lines 340-390 alter the 3D parameters to give the object the appearance of moving in space. Tumbling of the cube is done by increasing TH%, PH%, and PS% by 2 each time through the animation loop. When TH% reaches 72 it is reset to 0 as 72 x 5 = 360 degrees (one complete revolution). The values for TH%, PH%, and PS% must be in the range -128 to 127. If either of them are outside this range, then only its low-order 8 bits are used. These are interpreted as a two's-complement number in the range -128 to 127.

Figure 5.1 Demo 1 Program Listing

(16K Version)

```
10 'BOUNCING CUBE DEMONSTRATION
        1. Load 3D using SYSTEM mode.
        2. CLOAD this demonstration program.
        3. RUN this program.

20 GOSUB 65000 'INITIALIZE

30 'Define objects
40 FOR OBJ% = 1 TO 2

50 'Define Points
60 READ PTS%
70 FOR I% = 1 TO PTS%
80      READ X1%, Y1%, Z1%
90      POKE 16526,97: POKE 16527,112: ER%=USR(0) 'DEFPOINT
100 NEXT I%

110 'Define surfaces
120 READ SUR%
130 FOR I% = 1 TO SUR%
140     READ P1%, P2%, P3%
150     POKE 16526,193: POKE 16527,112: ER%=USR(0) 'DEFSURFACE
160 NEXT I%

170 'Define lines
180 READ LIN%
190 FOR I% = 1 TO LIN%
200     READ P1%, P2%, S1%, S2%
210     POKE 16526,108: POKE 16527,113: ER%=USR(0) 'DEFLINE
220 NEXT I%

230 NEXT OBJ%

240 'User interface
250 INPUT "Object 1 or 2"; OBJ%
260 IF (OBJ%<>1)AND(OBJ%<>2) THEN PRINT "Bad input, try again. "
        ;: GOTO 250

270 'Set Parameters and variables
280 SC%=10: HRZ%=128: VRT%=128: XX%=4: YY%=3: SS%=1

290 'Animation loop
300 FOR I% = 1 TO 10000
310     POKE 16526,8:   POKE 16527,123: ER%=USR(0) 'CLEAR
320     POKE 16526,202: POKE 16527,119: ER%=USR(0) 'PLACE
330     POKE 16526,22:  POKE 16527,123: ER%=USR(0) 'DISPLAY
335     PRINT@472,"3D Animation";
340     IF (SC%<6)OR(SC%>24) THEN SS%= -SS%
350     IF (HRZ%<85)OR(HRZ%>171) THEN XX%= -XX%
360     IF (VRT%<113)OR(VRT%>143) THEN YY%= -YY%
370     IF TH%=72 THEN TH%=0
380     TH%=TH%+2: PH%=TH%: PS%=TH%
390     SC%=SC%+SS%: HRZ%=HRZ%+XX%: VRT%=VRT%+YY%
400 NEXT I%
410 END
```

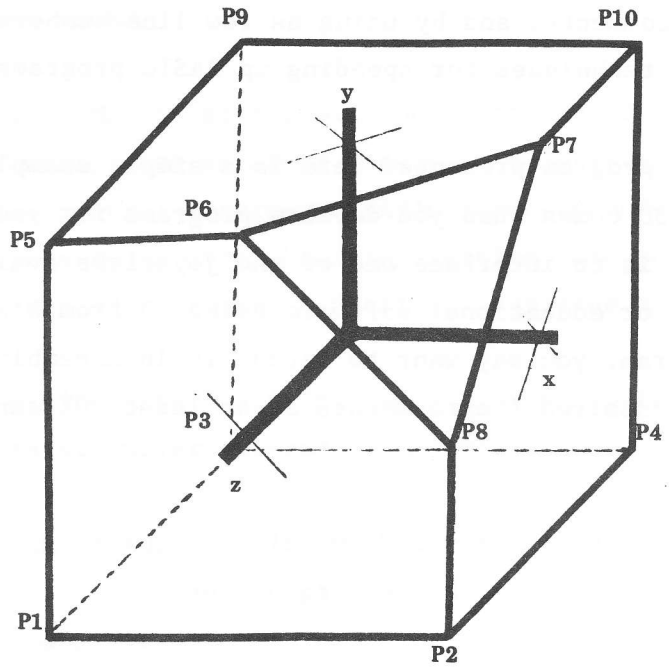
Figure 5.1 Demo 1 Program Listing (16K Version) - Continued

```
1000 '8 points for object 1
1010 DATA 8
1020 DATA -1,-1,1
1030 DATA 1,-1,1
1040 DATA -1,-1,-1
1050 DATA 1,-1,-1
1060 DATA -1,1,1
1070 DATA 1,1,1
1080 DATA -1,1,-1
1090 DATA 1,1,-1
1100 '6 surfaces for object 1
1110 DATA 6
1120 DATA 1,5,6
1130 DATA 7,5,1
1140 DATA 8,7,3
1150 DATA 4,2,6
1160 DATA 6,5,7
1170 DATA 1,2,4
1180 '12 lines for object 1
1190 DATA 12
1200 DATA 1,2,1,6
1210 DATA 1,3,2,6
1220 DATA 2,4,4,6
1230 DATA 3,4,3,6
1240 DATA 1,5,2,1
1250 DATA 2,6,1,4
1260 DATA 4,8,4,3
1270 DATA 3,7,3,2
1280 DATA 5,7,5,2
1290 DATA 5,6,5,1
1300 DATA 6,8,5,4
1310 DATA 7,8,5,3
```

```
1320 '10 Points for object 2
1330 DATA 10
1340 DATA -1,-1,1
1350 DATA 1,-1,1
1360 DATA -1,-1,-1
1370 DATA 1,-1,-1
1380 DATA -1,1,1
1390 DATA 0,1,1
1400 DATA 1,1,0
1410 DATA 1,0,1
1420 DATA -1,1,-1
1430 DATA 1,1,-1
1440 '7 surfaces for object 2
1450 DATA 7
1460 DATA 1,5,6
1470 DATA 5,1,3
1480 DATA 10,9,3
1490 DATA 2,8,10
1500 DATA 5,9,10
1510 DATA 1,2,4
1520 DATA 6,7,8
1530 '15 lines for object 2
1540 DATA 15
1550 DATA 1,2,1,6
1560 DATA 1,3,2,6
1570 DATA 2,4,4,6
1580 DATA 3,4,3,6
1590 DATA 1,5,2,1
1600 DATA 2,8,1,4
1610 DATA 4,10,4,3
1620 DATA 3,9,3,2
1630 DATA 5,9,5,2
1640 DATA 5,6,5,1
1650 DATA 7,10,5,4
1660 DATA 9,10,5,3
1670 DATA 6,7,5,7
1680 DATA 6,8,1,7
1690 DATA 7,8,4,7
```

```
65000 'INITIALIZATION
            This subroutine is called once prior to
            using the 3D Primitives.  For effeciency,
            it is called as early as possible.
  .
  .
  .
65046
```

Figure 5.2 Definition of Demo Object 2



P1 = (-1,-1,1)	P 6 = (0,1,1)
P2 = (1,-1,1)	P 7 = (1,1,0)
P3 = (-1,-1,-1)	P 8 = (1,0,1)
P4 = (1,-1,-1)	P 9 = (-1,1,-1)
P5 = (-1,1,1)	P10 = (1,1,-1)

In this demo, displacement of the object is controlled to keep the object in view. In general, however, displacement values are in the range 0 to 255 for HRZ% and VRT%, where the position (HRZ% = 64, VRT% = 104) is the lower left corner of the screen (see Appendix B). If HRZ% or VRT% is outside this range, then only its low-order 8 bits are used. They are treated as an 8-bit unsigned integer.

In this demo, the cube always moves at the same rate. An easy way to speed it up is to use larger increments in the rotation and displacement parameters.

Varying speeds are possible by varying the parameter increment sizes. To slow down the cube, delay loops can be used.

After a program has been developed, its efficiency is improved by removing extraneous spaces and comments, and by using as few line numbers as possible; that is, all the usual techniques for speeding up BASIC programs are utilized.

The demonstration program presented here is a simple example. The real fun and usefulness of 3D comes when you develop programs for your own applications. One suggestion is to interface one of the joysticks available for the TRS-80 and write games or educational software using 3D from BASIC. Once you have debugged the program, you may want to recode it in Assembly Language if the utmost in speed is desired (improvements of at least 70% can be expected).

6. USING 3D FROM ASSEMBLY LANGUAGE

Use of 3D from Assembly Language is explained by walking through the ADEMO program. ADEMO is on your cassette and is listed in Figure 6.1. We assume the reader is familiar with the Radio Shack Editor/Assembler (Cat. No. 26-2011) that was used to create ADEMO.

When running ADEMO, no errors will occur in the 3D primitives because it is debugged. In general, when an error occurs in a 3D primitive called from Assembly Language, control returns to BASIC start-up mode. This can be changed so that control goes to an error handling routine that you write or to the entry point of a debugger program. Section 7 explains how to make this change.

Each 3D primitive alters all Z80 registers, including the primed ones. But, 3D parameters are never altered by the primitives.

A walk-through of ADEMO begins with program lines 12-39 that EQUate locations of the 3D primitives and 3D parameters to labels. Program line 41 calls the INITIALIZE primitive.

Program lines 43-95 define a two-dimensional square. The technique used is simple but not flexible. Other techniques are possible, including ones that request the input of points, lines, and surfaces from a cassette or user.

Program lines 97-108 set the 3D parameters in preparation for the animation. Program lines 110-135 are the animation loop. Only Z-axis rotation is used to make the square spin as it is enlarged. When SC=48, it is reset to 0. When PS=72, it is reset to 0.

Figure 6.1 Assembly Language Demo Program Listing
(16K Version)

```
00001 ;ASSEMBLY LANGUAGE DEMONSTRATION
00002 ;         1. Load the 3D Primitives using SYSTEM mode.
00003 ;         2. Load this Program using SYSTEM mode.
00004 ;         3. Type a slash (/) then hit ENTER to run.
00005
00006           ORG     6A00H
00007
00008 ;INITIALIZE
00009
00010 ;EQUate the locations of the 3D Primitives
00011
00012 INIT      EQU     7F5FH    ;Initialize Primitive
00013 DEFPNT    EQU     7061H    ;Define Point Primitive
00014 DEFSUR    EQU     70C1H    ;Define surface Primitive
00015 DEFLIN    EQU     716CH    ;Define line Primitive
00016 CLEAR     EQU     7B08H    ;Clear Primitive
00017 PLACE     EQU     77CAH    ;Place Primitive
00018 DISPLA    EQU     7B16H    ;DisPlay Primitive
00019 DRAW      EQU     7B22H    ;Draw Primitive
00020
00021 ;EQUate the locations of the 3D Parameters
00022
00023 HRZ       EQU     7F4EH    ;Horizontal
00024 VRT       EQU     7F4FH    ;Vertical
00025 TH        EQU     7F50H    ;Theta
00026 PH        EQU     7F51H    ;Phi
00027 PS        EQU     7F52H    ;Psi
00028 SC        EQU     7F53H    ;Scale
00029 OBJ       EQU     7F54H    ;Object
00030 X1        EQU     7F55H    ;X-coordiate
00031 Y1        EQU     7F56H    ;Y-coordinate
00032 X2        EQU     7F57H    ;X-coordinate
00033 Y2        EQU     7F58H    ;Y-coordinate
00034 Z1        EQU     7F59H    ;Z-coordinate
00035 P1        EQU     7F5AH    ;Point
00036 P2        EQU     7F5BH    ;Point
00037 P3        EQU     7F5CH    ;Point
00038 S1        EQU     7F5DH    ;Surface
00039 S2        EQU     7F5EH    ;Surface
00040
00041 ENTRY     CALL    INIT     ;Call the INIT Primitive
00042
```

Figure 6.1 Assembly Language Demo Program Listing (16K Version) - Continued

```
00043 ;Define object 1  (a square frame with 1 surface)
00044          LD       A,1
00045          LD       (OBJ),A
00046
00047 ;Define Points for object 1
00048          LD       A,-1      ;Define Point 1
00049          LD       (X1),A
00050          LD       (Y1),A
00051          LD       A,0
00052          LD       (Z1),A
00053          CALL     DEFPNT    ;(X1=-1, Y1=-1, Z1=0)
00054
00055          LD       A,1       ;Define Point 2
00056          LD       (X1),A
00057          CALL     DEFPNT    ;(X1=1, Y1=-1, Z1=0)
00058
00059          LD       A,1       ;Define Point 3
00060          LD       (Y1),A
00061          CALL     DEFPNT    ;(X1=1, Y1=1, Z1=0)
00062
00063          LD       A,-1      ;Define Point 4
00064          LD       (X1),A
00065          CALL     DEFPNT    ;(X1=-1, Y1=1, Z1=0)
00066
00067 ;Define surface for object 1
00068          LD       A,3       ;Define surface 1
00069          LD       (P1),A
00070          LD       A,2
00071          LD       (P2),A
00072          LD       A,1
00073          LD       (P3),A
00074          CALL     DEFSUR    ;(P1=3, P2=2, P3=1)
00075
00076 ;Define lines for object 1
00077          LD       A,2       ;Define line from 1 to 2
00078          LD       (P1),A
00079          LD       A,1
00080          LD       (P2),A
00081          LD       (S1),A
00082          LD       (S2),A
00083          CALL     DEFLIN    ;(P1=2, P2=1, S1=1, S2=1)
00084
00085          LD       A,3       ;Define line from Point 2 to 3
00086          LD       (P2),A
00087          CALL     DEFLIN    ;(P1=2, P2=3, S1=1, S2=1)
00088
00089          LD       A,4       ;Define line from Point 3 to 4
00090          LD       (P1),A
00091          CALL     DEFLIN    ;(P1=4, P2=3, S1=1, S2=1)
00092
00093          LD       A,1       ;Define line from Point 4 to 1
00094          LD       (P2),A
00095          CALL     DEFLIN    ;(P1=4, P2=1, S1=1, S2=1)
00096
```

Figure 6.1 Assembly Language Demo Program Listing (16K Version) - Continued

```
00097 ;Initialize Parameters
00098          LD       A,128     ;HRZ=128, VRT=128
00099          LD       (HRZ),A
00100          LD       (VRT),A
00101
00102          LD       A,0       ;TH=0, PH=0, PS=0
00103          LD       (TH),A
00104          LD       (PH),A
00105          LD       (PS),A
00106
00107          LD       A,1       ;SC=1
00108          LD       (SC),A
00109
00110 ;ANIMATION LOOP (spin and enlarge frame)
00111
00112 LOOP     CALL     CLEAR     ;Clear virtual screen
00113          CALL     PLACE     ;Place object 1
00114          CALL     DISPLA    ;Display on CRT
00115
00116          LD       HL,PS     ;PS=PS+2
00117          INC      (HL)
00118          INC      (HL)
00119
00120          LD       A,(HL)    ;If PS>70 then PS=0
00121          CP       71
00122          JP       M,CONT1
00123          LD       (HL),0
00124
00125 CONT1    LD       HL,SC     ;SC=SC+1
00126          INC      (HL)
00127
00128          LD       A,(HL)    ;If SC>47 then SC=0
00129          CP       48
00130          JP       M,CONT2
00131          LD       (HL),0
00132
00133 CONT2    JP       LOOP      ;Inifinite loop
00134
00135          END      ENTRY
```

7. ERROR HANDLING

When certain 3D primitives are called with improper parameter values, error handling takes place. The nature of this error handling depends on whether the primitive was called from BASIC or Assembly Language, and whether the programmer had chosen to act on the error by including the appropriate program statements.

Error handling is only performed by DEFPOINT, DEFSURFACE, DEFLINE, and PLACE. The other primitives do not need error handling as they either accept no inputs or accept all values of inputs for their parameters.

Errors During Calls From BASIC

In this discussion, it is assumed that the POKE addresses/values in the calling program are correct, as this type of error is not detected by 3D.

When an error occurs, control immediately returns to BASIC, and an error code is assigned to the variable in the active USR statement. The specific actions performed by each of the primitives follow.

When DEFPOINT is called, it first checks whether OBJ% is between 1 and 10 inclusive. If this isn't true, it returns to BASIC with an error code of 1. Otherwise, it checks that less than 85 points have already been defined. If this isn't true, then it returns to BASIC with an error code of 2. Figure 7.1 shows an example of how to utilize the error handling feature of DEFPOINT.

Figure 7.1 BASIC Error Handling For DEFPOINT
(16K Version)

```
POKE 16526,97: POKE 16527,112: ER%=USR(0) 'DEFPOINT
IF ER%=1 THEN PRINT"ILLEGAL OBJ": STOP
IF ER%=2 THEN PRINT"ILLEGAL PNT": STOP
```

When DEFSURFACE is called, it first checks whether OBJ% is between 1 and 10 inclusive. If this isn't true, then it returns to BASIC with an error code of 1. Otherwise, it checks whether less than 64 surfaces have been defined. If not, then it returns to BASIC with an error code of 2. Finally, a check is made to ensure that points P1%, P2%, and P3% have been defined for the current object. If not, control returns with an error code of 2.

Figure 7.2 shows an example of how to utilize the error handling feature of DEFSURFACE.

Figure 7.2 BASIC Error Handling for DEFSURFACE
(16K Version)

```
POKE 16526,193: POKE 16527,112: ER%=USR(0) 'DEFSURFACE
IF ER%=1 THEN PRINT"ILLEGAL OBJ": STOP
IF ER%=2 THEN PRINT"ILLEGAL PNT/SUR": STOP
```

When DEFLINE is called, it first checks whether OBJ% is between 1 and 10 inclusive. If not, an error code of 1 is returned. Otherwise, a check is made to ensure that less than 64 lines are defined. If not, an error code of 2 is returned. Finally, a check is made to ensure that P1%, P2%, S1%, and S2% are defined for the current object. If not, an error code of 2 is returned. Figure 7.3 shows an example of how to utilize error handling with DEFLINE.

Figure 7.3 BASIC Error Handling for DEFLINE
(16K Version)

```
POKE 16526,188: POKE 16527,113: ER%=USR(0)  'DEFLINE
IF ER%=1 THEN PRINT"ILLEGAL OBJ": STOP
IF ER%=2 THEN PRINT"ILLEGAL PNT/SUR/LIN": STOP
```

When PLACE is called, it first checks whether OBJ% is between 1 and 10 inclusive. If not, then an error code of 1 is returned. Next, a check is made as to whether the object has at least 1 point, 1 line, and 1 surface. If not, an error code of 2 is returned. Figure 7.4 shows an example of how to utilize error handling with PLACE.

Figure 7.4 BASIC Error Handling for PLACE
(16K Version)

```
POKE 16526,202: POKE 16527,119: ER%=USR(0) 'PLACE
IF ER%=1 THEN PRINT"ILLEGAL OBJ": STOP
IF ER%=2 THEN PRINT"INCOMPLETE OBJ": STOP
```

Errors During Calls From Assembly Language

3D recognizes the same errors when called from Assembly Language as when called from BASIC. When using Assembly Language, however, the errors are necessarily handled differently. When an error is discovered, the HL register is loaded with the error code and control returns to the location pointed to by the error vector. The error vector (see Figure 7.5) contains the address of an error routine that is branched to when an error occurs. The default value is 0000H, the entry to BASIC. To change the error vector to, say, an error routine you wrote, load the error routine address into the error vector after calling the INITIALIZE primitive.

Figure 7.5 Error Vector

Vector Location			Contents
16K Version	32K Version	48K Version	———————
716AH	B16AH	F16AH	LSB of error routine
716BH	B16BH	F16BH	MSB of error routine

8. HIDDEN LINE CONCEPT

3D utilizes a novel hidden line removal algorithm. The algorithm is fast and requires only that surfaces and lines for objects be specified correctly.

When defining a surface for an object, you should choose previously defined points P1%, P2%, and P3% that are oriented clockwise with respect to one another as viewed from outside the object. When defining a line for an object, you should choose the previously defined surfaces S1% and S2% that adjoin the line. If the line lies on a previously defined surface, such as a window that lies on the side of a house, then make S1% and S2% that one surface.

The hidden line algorithm is used by PLACE to determine which lines in an object are hidden from view and should not be drawn. To determine which lines are not to be drawn, the algorithm examines the two surfaces adjoining each line. If both of these surfaces are hidden from the viewer, then the line is not drawn.

The algorithm works for any number of non-overlapping convex (having a boundary that bulges outward) polyhedra (solids bounded by polygons).

The hidden line removal feature is normally on, but may be turned off and on at any time by altering a control bit in 3D (see Figure 8.1). To turn the feature off, you must set the control bit to 0 prior to calling PLACE. When PLACE is called, the object will be drawn with all "hidden lines" included. To turn the feature back on, you must set the control bit to 1.

Figure 8.1 Hidden Line Control Bit

Version	Location	To Turn Off From BASIC
16K	7741H	POKE 30529,0
32K	B741H	POKE -18623,0
48K	F741H	POKE - 2239,0

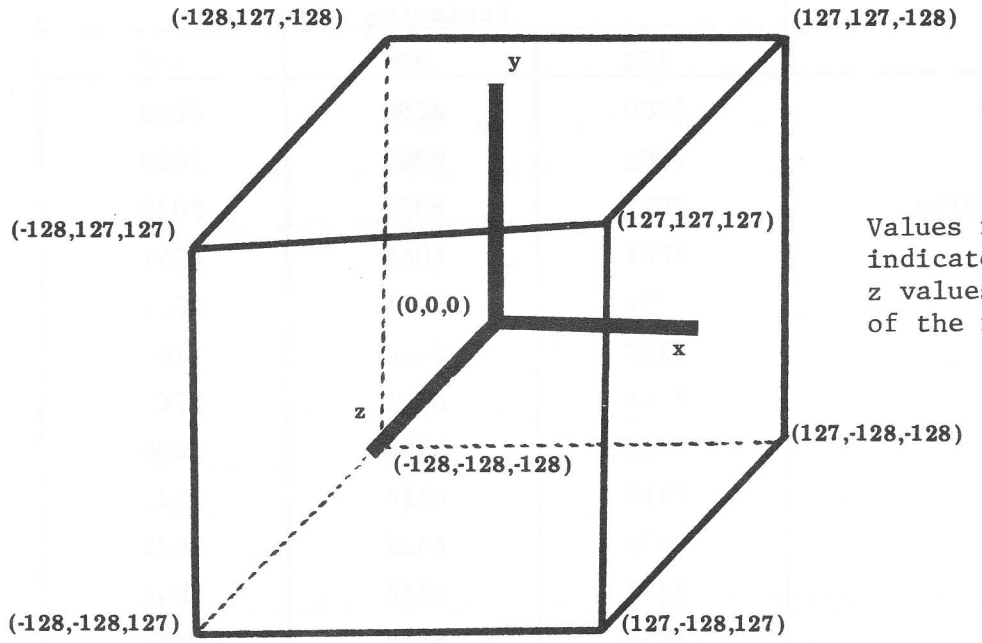
Appendix A

3D Memory Map

Contents	Beginning Addresses		
	16K	32K	48K
Beginning of 3D	6C00	AC00	EC00
Symbol Table	7000	B000	F000
Initialization Area	7022	B022	F022
DEFPOINT	7061	B061	F061
DEFSURFACE	70C1	B0C1	F0C1
DEFLINE	716C	B16C	F16C
PLACE	77CA	B7CA	F7CA
CLEAR	7B08	BB08	FB08
DISPLAY	7B16	BB16	FB16
DRAW	7B22	BB22	FB22
Virtual Screen	7B4E	BB4E	FB4E
INITIALIZE	7F5F	BF5F	FF5F
End of 3D	7FFF	BFFF	FFFF

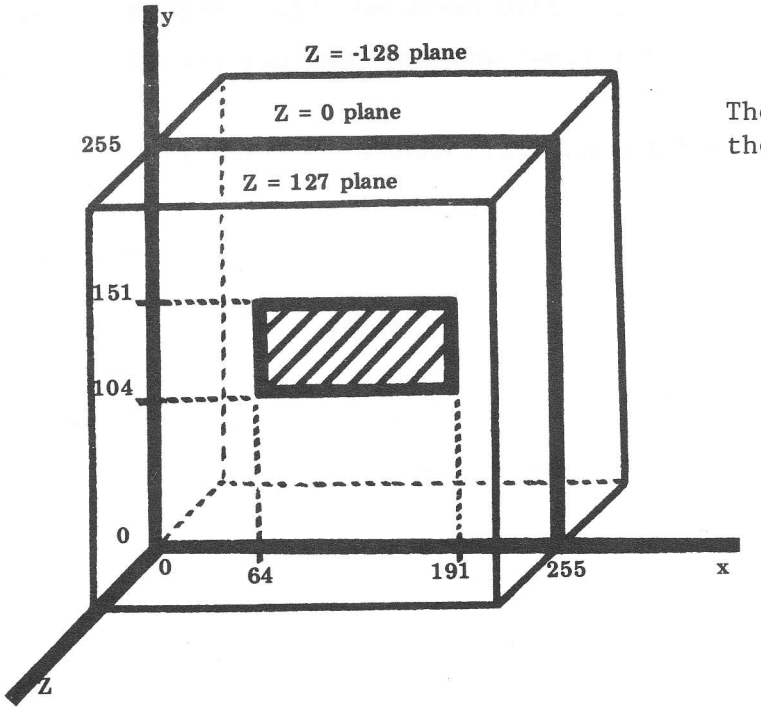
Appendix B

3D Object Definition Space



Values in parentheses indicate the x, y, and z values, respectively, of the indicated point.

3D Screen Space



The hashed area denotes the visible region.

Dec. 19, 1983

Roxton Baker
Box 8272
APO San Fran.
96555

Algorithmic Assoc.
Box 244
Bedford, Mass. 01730

Dear Sirs,

This is an update to my recent letter requesting another tape copy of the Three-D package I bought from you. I have after some time been able to recover (I think) the 32K version of the object code and Basic demo programs. In running them I have found what may be a serious flaw in the Three-D program.

It appears from the HOUSE demo that while both the Y and Z axes remain properly fixed in space, the X axis does not. It remains fixed instead to the object as originally defined. Thus as you rotate the house around the Y axis (see attached printouts), the X axis also rotates around the Y axis - eventually becoming the same as the Z axis (when the Y rotation is 90 degrees). In other words, no matter what Y or Z rotations have been entered, the X axis will ALWAYS be a line drawn through the two ends of the house!

This is confirmed by the CUBE demo, if you manually set and change PH% and TH%. So it can't be a problem in the object definition data.

Another effect I noticed is that when the rotation values TH%, PH%, etc. reach 127 or 128, or some multiple, the picture will suddenly rotate an additional 180 degrees. Apparently this is the nature of things, and not really a problem - but you might mention it in the program remarks.

Finally, the problem with running the 32K programs under Disk Basic can be solved by adding the lines shown to the initialization section, to handle negative numbers returned from the VARPTR function.

Again, it may be that I simply have a bad copy of the 32K code, and there is not really a bug in the X axis realization. If you do find a genuine error I'm sure you will fix it, in which case I would rather that you wait and send the tape with corrected code, as opposed to sending it with new copies of the V1.1. In either event, please do let me hear from you on this subject. I have a good deal of money tied up in the program, and would like to be able to use it.

Thank you,

Roxton Baker

Algorithmic Associates
PO Box 244
Bedford, Massachusetts 01730

12/29/83

Dear Mr. Baker,

Please find enclosed replaced tapes for the faulty one we previously shipped. Our apologies are extended for the inconvenience caused.

Thanks for the information on how to make 3D compatible with Disk Basic. At some point in the future we may support 3D on disk, however, presently we support only a cassette version for Level II/III Basic. We're happy its working for you!

There is no bug with X-axis rotation. 3D uses the standard terminology and definition of 3-space rotation, as such, x-axis is always performed first, followed by y-axis rotation, and finally z-axis rotation. This is done for each frame of animation. Page 13 of users manual clarifies this in the text describing PLACE.

The effects you noticed with TH%, PI%, and PS% when they reach -128 or 127 is explained in the last paragraph of page 19 - remember that rotations are in units of 5°.

Sincerely Yours,

Tom