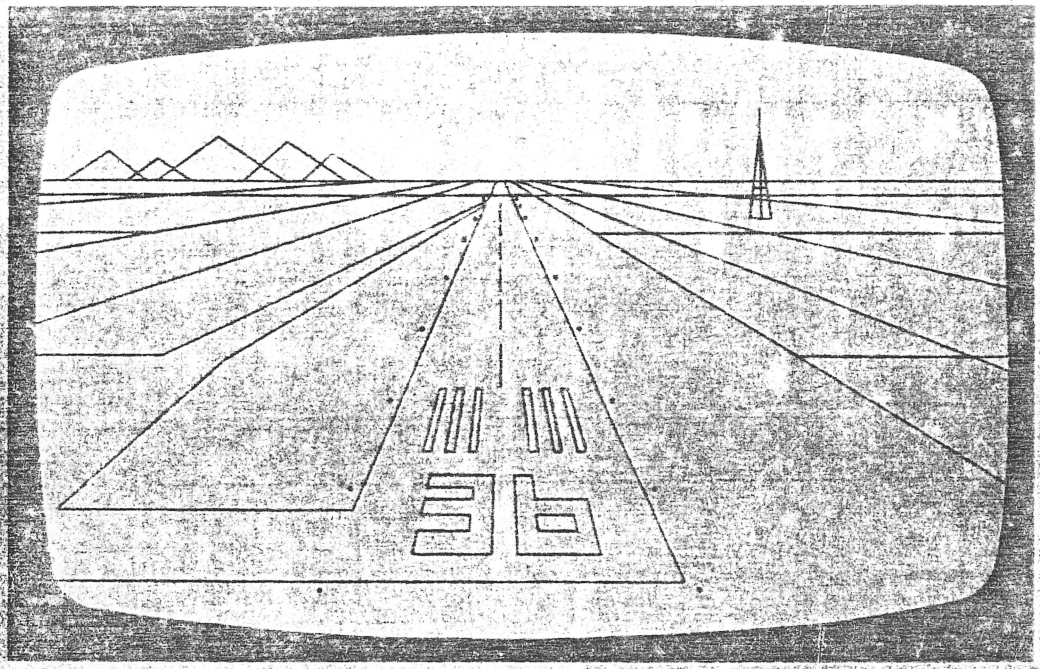
Animation Package

3D Microcomputer Graphics



Apple II Assembly Language

subLOGIC
Box V, Savoy, IL 61874

THREE - DIMENSIONAL MICROCOMPUTER GRAPHICS

6502/APPLE II ASSEMBLY LANGUAGE

by

Bruce A. Artwick

August 1979

Sublogic Company
201 W. Springfield Avenue
Champaign, Illinois 61820

First Edition
Second Printing

Printed in U.S.A.

A2-3DI 3D ANIMATION PROGRAM REFERENCE CARD

COMMAND NAME	COMMAND (decimal)	COMMAND (hex)	ARGUMENTS	FUNCTION
PNT	00	00	X lsb, X msb, Y lsb, Y msb, Z lsb, Z msb	Define 3D Point
SPNT	01	01	X lsb, X msb, Y lsb, Y msb, Z lsb, Z msb	Define 3D Start Point
CPNT	02	02	X lsb, X msb, Y lsb, Y msb, Z lsb, Z msb	Define 3D Continue Point
RAY	03	03	X lsb, X msb, Y lsb, Y msb, Z lsb, Z msb	Define 3D Ray
CLPSW	04	04	n where n = 0 clipper on, n = 1 clipper off	Clipper Control Switch
EYE	05	05	X lsb, X msb, Y lsb, Y msb, Z lsb, Z msb, P, B, H	Viewer's X, Y, Z, P, B, H
LIN2D	06	06	X1, Y1, X2, Y2	Draw 2D Line from Point 1 to 2
DISP	07	07	n where n = 50 set graphics n = 51 set text n = 52 clear mixed n = 53 set mixed n = 54 page 1 set n = 55 page 2 set n = 56 clear HI-RES n = 57 set HI-RES	Display Screen Select
ERAS	08	08	n where n = 00 erase page 1 n = 01 erase page 1 n = 02 fill page 1 n = 03 fill page 2	Erase Screen
DRAW	09	09	n where n = 00 draw page 1 n = 01 draw page 2	Write Screen Select
PNT2D	10	0A	X, Y	Plot 2D Point
JMP	11	0B	A lsb, A msb where A is the jump address	Interpretive Jump
LMODE	12	0C	n where n = 00 normal line n = 01 exclusive or line	Set Line Drawing Mode
ARRAY	13	0D	A lsb, A msb where A is output array start address	Turn On Output Array Generation
SCRSZ	14	0E	Screen width, Screen height, X center, Ycenter	Screen Size Selection
FIELD	15	0F	axr lsb, axr msb, ayr lsb, ayr msb, azr lsb, azr msb	Field of View Selection
INIT	16	10	none	Easy Initialize
NOP	17	11	none	No Operation

subLOGIC
Box V, Savoy, IL 61874

Program reference card.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

Computer graphics is one of the most interesting areas of computer science. Interest in this field has been increasing over the last few years due to the wide variety of reasonably-priced graphics hardware that new integrated circuit technology makes possible. The 6502-based personal computer is one product that is leading the way in this new "micro graphics" field. When loaded with the proper software, a 6502-based system can yield some very impressive displays.

The Sublogic A2-3D1 three-dimensional graphics package is designed to add to the graphics capabilities of the new 6502-based personal computers. The package allows users to create scenes in three dimensional space, and view them from any direction or location in space. This capability opens whole new fields of 3D animation, architectural drawing, modeling and simulation, and enhances special effects in game programs. This program also brings the sought-after pan , zoom, rotation, and scaling functions to 2D displays.

The A2-3D1 program is designed to run on 6502-based microcomputers in general, and on the APPLE II computer in particular. The program internally consists of two general parts: a 6502 3D-to-2D converter and a high performance 2D high-resolution driver for the APPLE II . The 6502 3D to 2D converter is useful to anyone owning a 6502-based microcomputer. This part of the program is device-independent and produces a list of line start and end points in memory. This list of lines can be used to generate 2D projections of 3D scenes on plotters, bit maps, terminals, or anything else that can plot points or draw lines.

The A2-3D1's high performance 2D driver allows Apple II users to project these lines on an Apple II high-resolution screen. The 2D driver is, in itself, a very powerful graphics tool. It can be used seperately to increase the erase, line drawing, and point plotting rates of programs currently using APPLESOFT's high-resolution routines by 600, 500, and 1000 percent respectively. Table 1 presents the important characteristics of the A2-3D1 package.

TABLE 1. Sublogic 6502/APPLE II assembly language 3D graphics package characteristics.

CHARACTERISTIC	DATA
Program Number	A2-3D1
Program Language	6502 Optimized Assembly Language
Projection Method	3D to 2D wire frame perspective transformation with 3D clipping.
Viewing Capabilities	X,Y,Z range: + or - 32767 units 3 axis freedom: 0 to 359 degrees in 256 even steps
World Size	1912 cubic miles using one foot units
Special Features	Variable viewing window (telephoto to wide angle) Variable screen aspect and bit size ratios Clipped or nonclipped projection control Start, continue, ray and pure point projection High performance APPLE II 2D driver Screen display or output array selection
Execution Rate	3D line drawing (including APPLE II screen draw) =150/sec APPLE II 2D line draw (full screen width) =333 lines/sec APPLE II point plotting = 10,000 points/sec APPLE II screen erase= 22 erases/second
User Aids	Seperate technical and load-and-go manuals. BASIC and assembly language interface programs. Zero-page and register restore upon subroutine return.

The A2-3D1 program is a derivation of earlier Sublogic 3D drivers (Z-80 and 6800) and also contains many of the advanced array handling features of our Universal Graphics Interpreters. Every attempt was made to make the program as reliable as possible. One of the biggest problems in developing any graphics software, however, is 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 - 32767 units in the X, Y, and Z directions has 2.6×10^{14} (64000 cubed) possible coordinate points. There are therefore 6.9×10^{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, combined with user feedback, promises to increase program performance and reliability.

This manual describes the implementation and use of the A2-3D1 package. Explicit instructions on how to get the program running on the APPLE II as well as algorithms, programs, and hints to help non-APPLE II 6502 users are given. We hope you find the performance and capabilities of this package to your liking and welcome any comments concerning it.

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 3D-to-2D converter subroutine (the heart of the A2-3D1 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 and screen control commands in a preset format) in processor memory using BASIC POKEs or an assembly language monitor. The user then CALLs the subroutine and the 3D image appear on the APPLE II high resolution screen. Non-APPLE II owners will instead find all the lines to be plotted in an easy-to-use output array in memory, ready for plotting (see the Using the A2-3D1 with Non-APPLE II Systems section). Figure 1 interrelates the input array, 3D-to-2D converter, and displayed image. Notice that non-APPLE II owners use the 3D-to-2D output directly.

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 A2-3D1 program.

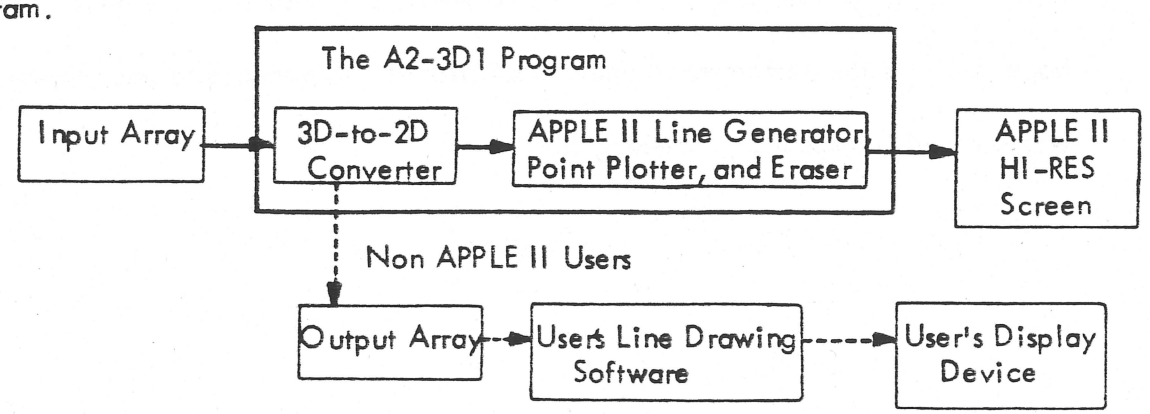


Figure 1. The A2-3D1 program and its arrays.

SPACE AND SCREEN COORDINATES

The first concept that must be well understood is that of space and screen coordinates. It should be noted that the term screen used throughout this discussion implies plotters and other display devices as well as the APPLE II screen.

3D SPACE COORDINATES. Every point in 3-dimensional space has an X, Y, Z space coordinate associated with it as figure 2a shows. A straight line is represented by its start and end points as shown in figure 2b.

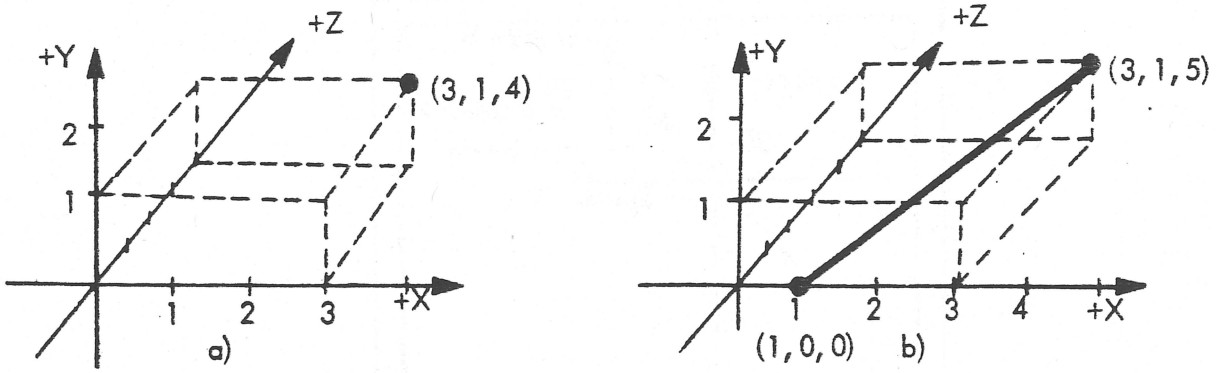
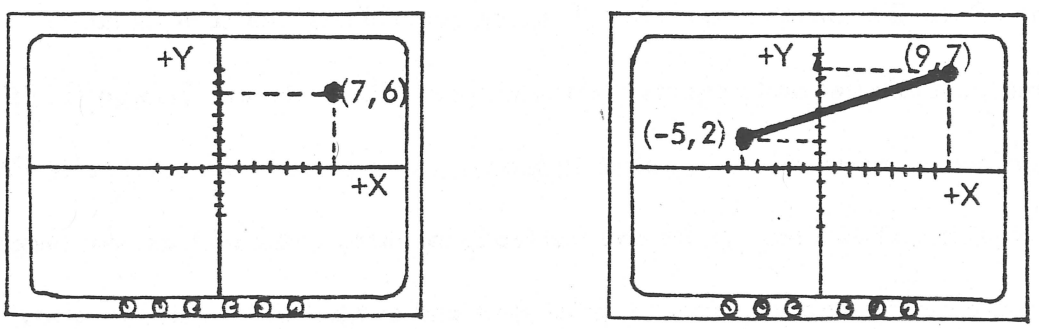


Figure 2. Three dimensional space coordinates.

2D SCREEN COORDINATES. Every point appearing on the screen has a 2D screen coordinate associated with it. Figure 3a shows a point on a screen, and a screen line is represented by a screen start and end point as shown in figure 3b.



a) A screen point

b) A screen line

Figure 3. Two dimensional screen coordinates.

3D AND 2D 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 4 illustrates, the X and Y space coordinate axes viewed through a screen match the X and Y screen axes. The Z axis represents 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.

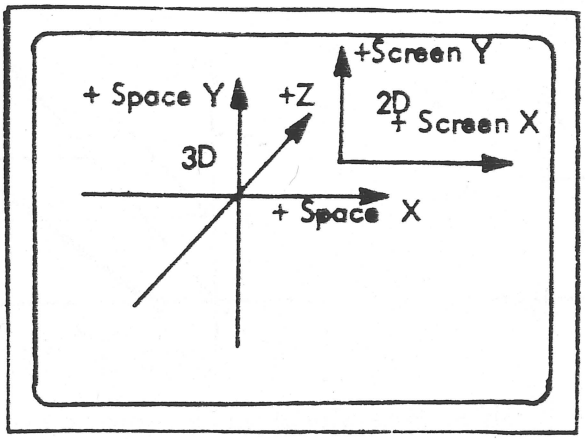
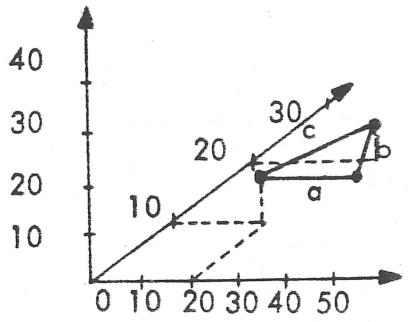


Figure 4. 3D and 2D coordinate alignment.

OBJECT CONSTRUCTION IN SPACE COORDINATES

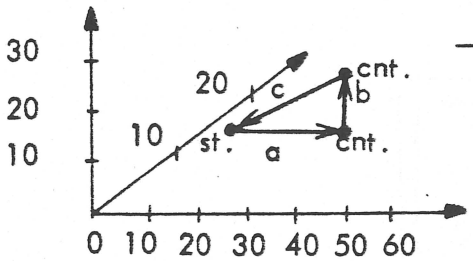
The 3D-to-2D converter converts 3D space coordinates into corresponding 2D screen coordinates and projects them onto the APPLE II screen. Straight lines in space are represented by two points in space: a start point and an end point. Wire-frame objects and outlines can be constructed using many straight lines. An image might therefore be described as a series of start and end points in a list (or array) in memory as figure 5 illustrates.



	Start pt. X, Y, Z	End pt. X, Y, Z
Line a	20, 10, 10	40, 10, 10
Line b	40, 10, 10	30, 10, 20
Line c	30, 10, 20	20, 10, 10

Figure 5. The conventional way of representing lines.

Notice that line a's end point is the same as b's start point, and b's end point is the same as c's start point. When creating outlines of objects, this end-to-end form of line construction is so common that the "continue point" is used instead of the "end point" in the A2-3D1 program. The triangular image is therefore represented as shown in figure 6.



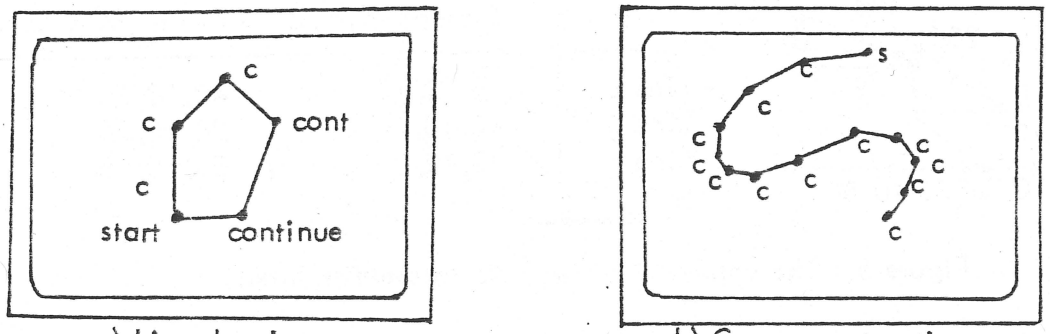
	Line Specifier (one or two points)	
Line a	20, 10, 10 start	40, 10, 10 continue
Line b	30, 10, 20 continue	
Line c	20, 10, 10 continue	

Figure 6. The A2-3D1's way of representing lines.

Line projection time and the amount of memory needed to represent a scene are greatly reduced using start and continue rather than start and end points.

Although no official curves are allowed in the 3D scene, they can be drawn with surprising realism as a long string of short, straight lines. Although very realistic curves can be generated using large numbers of line segments, projection speed suffers as each line must be projected seperately. Figure 7 shows a polygon and a curve drawing.

A group of many lines with a common vertex can efficiently be represented using "ray points". A ray point is similar to a continue point but the continuation of the



a) Line drawing b) Curve construction

Figure 7. Object construction in 3D space

line doesn't advance with each new ray. Figure 8 illustrates an object constructed using start, continue, and ray points.

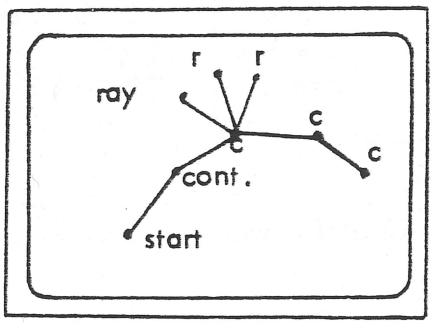
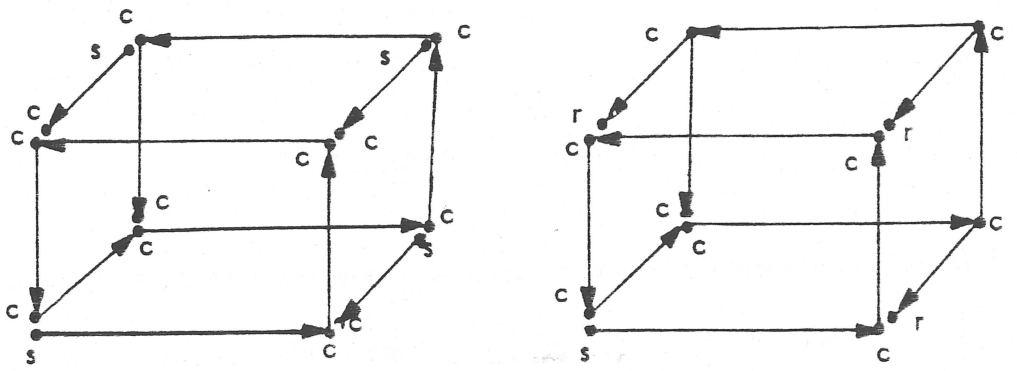


Figure 8. Object construction with ray points.

Rays are useful when drawing 3D polyhedra where one tends to "get stuck" using only start and continue points. Cube drawing is a good example of this situation. Up to nine edges (lines) of a twelve-edged cube may be drawn using a single start and nine continue points, and three seperate start-continue pairs must be used to pickup the remaining three edges as shown in figure 9a. With ray points these edges can be generated without having to use seperate start-continue point pairs as figure 9b illustrates.



a) using start and continue points b) using start, continue, and ray points

Figure 9. Cube construction in 3D space.

Finally, if only points need to be projected, such as in an astronomy, molecular model, or space flight application, pure point representation can be used. Figure 10 shows a scene with a few pure points.

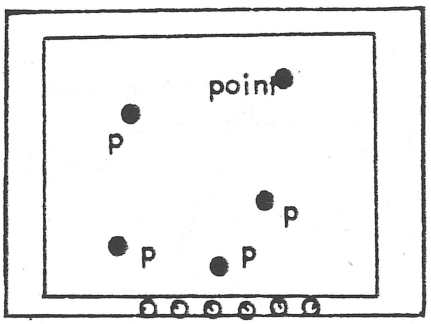


Figure 10. A scene with a few pure points.

Once a user has decided what he wants in his data base, the scene must be coded-up and put into the proper input array format in memory. The INPUT ARRAY DETAILS section of this manual will discuss the exact input array formats.

VIEWING CONTROL

A few 3D to 2D converter control parameters are needed before a 3D-to-2D conversion can be performed. The location, direction from which one wants to view the scene and the field of view are required.

VIEWER LOCATION AND VIEWING DIRECTION. First the viewer's location in space must be specified. Figure 11 illustrates what is meant by viewer's location. An X, Y, Z viewer location must be submitted in the input array. Movement of the viewer in the X, Y, Z directions is called translation.

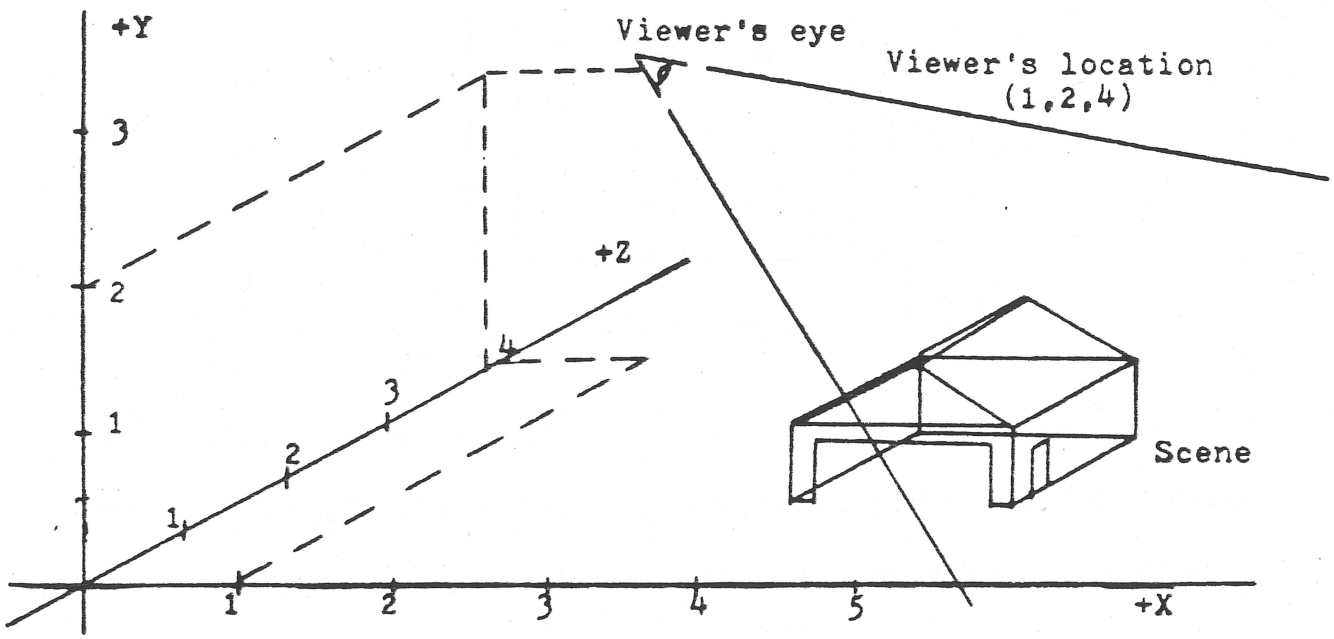


Figure 11. Viewer's location in space

Viewer's direction must also be specified as figure 12 illustrates. A pitch, bank and heading are submitted. Change in the viewer's direction is called rotation.

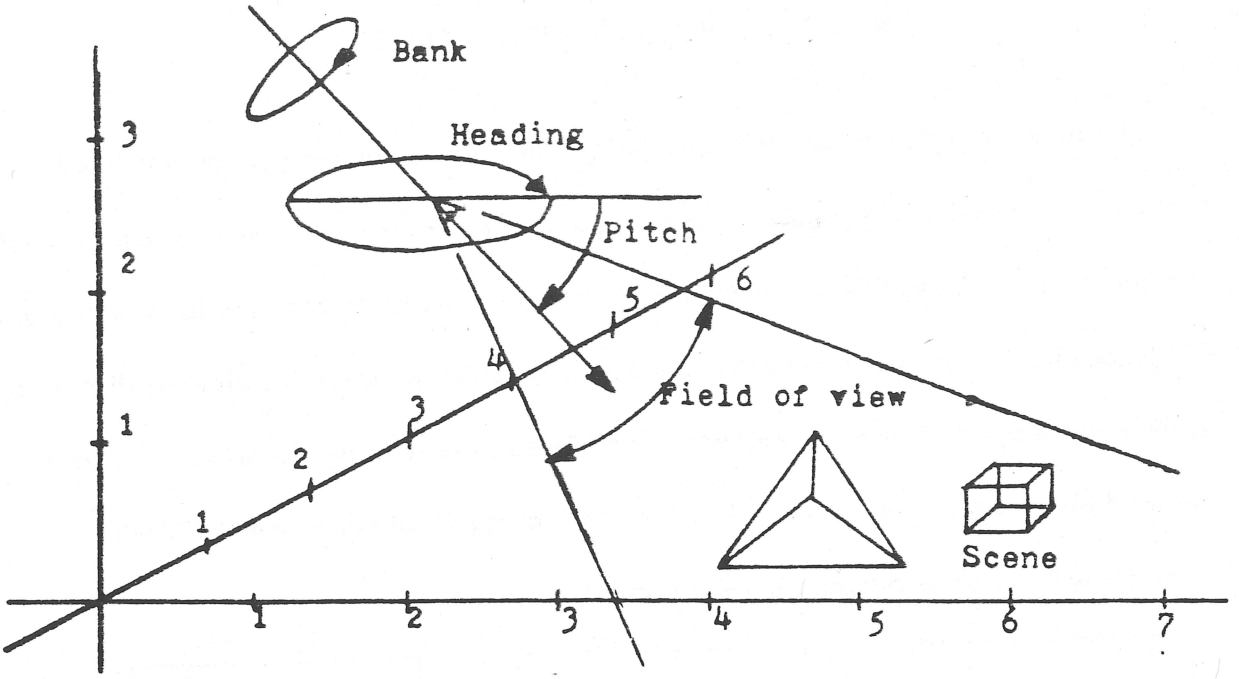


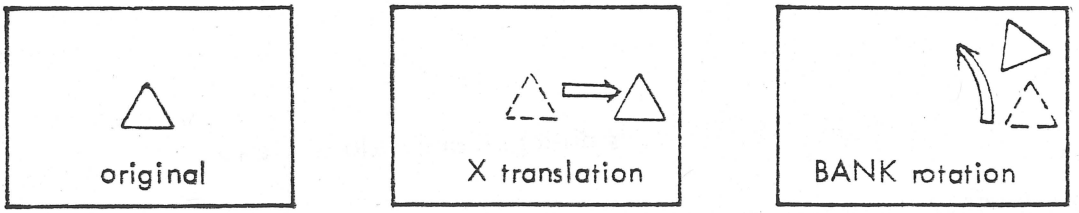
Figure 12. Viewer's direction and field of view.

FIELD OF VIEW. Another parameter not related to viewer's movement and viewing direction may also be specified in the input array. This parameter controls the field of view (similar to a comera's telephoto and wide angle effect) and lets the user select wide angle, narrow angle, and medium angle views. Since the field of view is continuously variable, zoom effects are possible. Only a limited field of view can fit into a viewing screen and the viewer must decide whether a wide angle or telephoto view is desired. Figure 12 shows the field of view.

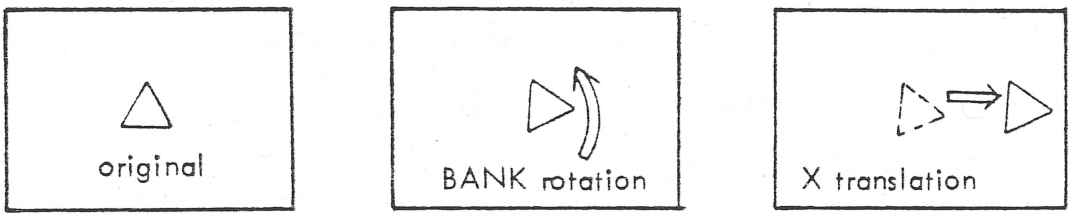
The 05 and 0F command sheets cover viewer's location, direction and field of view in greater detail.

ORDER OF TRANSFORMATION

The order in which transformations (X, Y, Z movement and pitch, bank and heading) are performed by the 3D-to-2D converter 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 a viewer's location in space is considered before his viewing direction, a different projection than if they were considered in reverse would result. Figure 13 shows these two orders of projection. Notice that the final results are different, even though the same operations were applied.



a) X translation before BANK rotation.



b) X translation after BANK rotation.

Figure 13. The difference transformation order makes.

The A2-3D1 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 (roll about the lateral axis)

Figure 14 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 causes an object to move to the left if the viewer is at a 0 degree bank angle. If the viewer is in a 90 degree bank, however, the cube appears to move down instead.

Sometimes senses of rotation actually seem to operate in the wrong manner. Changes in heading will produce exactly the same results as changes in bank when looking straight down for instance. A little thought, however, convinces one that this is indeed the way things should work.

It should be noted that positive Y represents the viewer being at a positive altitude.

PROJECTION MODES AND CLIPPING

With full user freedom in scene design and viewing direction selection, scenes invariably end up being fully or partially off the screen. If the object is behind the viewer, it is not visible. The process of eliminating these off-screen lines and cutting partially on-screen lines down to size is called clipping.

Elimination of off-screen lines is very simple computationally and presents no problems. Clipping partially on-screen lines, on the other hand, is the hardest and most time-consuming portion of the 3D projection process. The A2-3D1 program has full 3D clipping capabilities and even offers a selection of two different clipping modes.

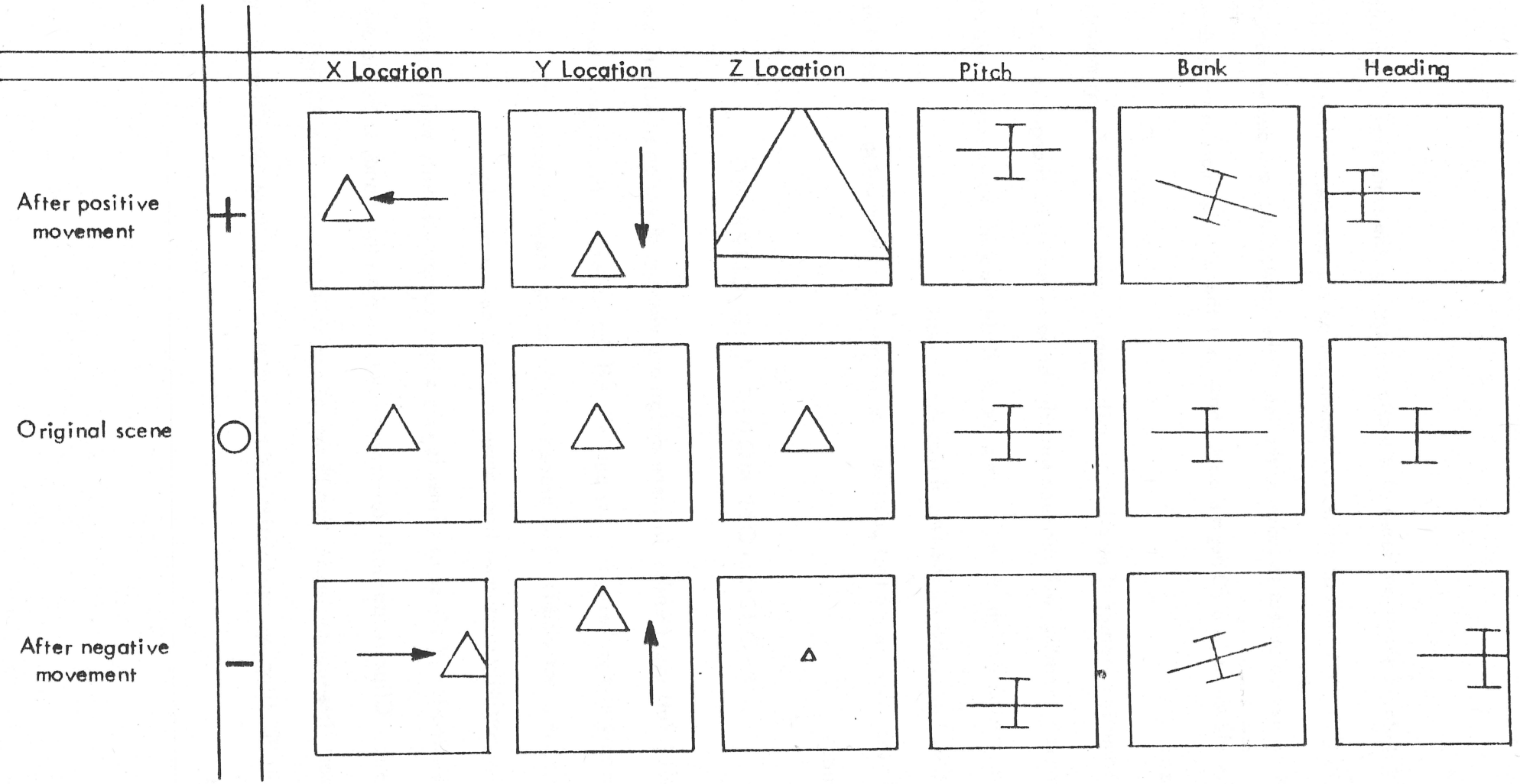


Figure 14. Direction of movement conventions.

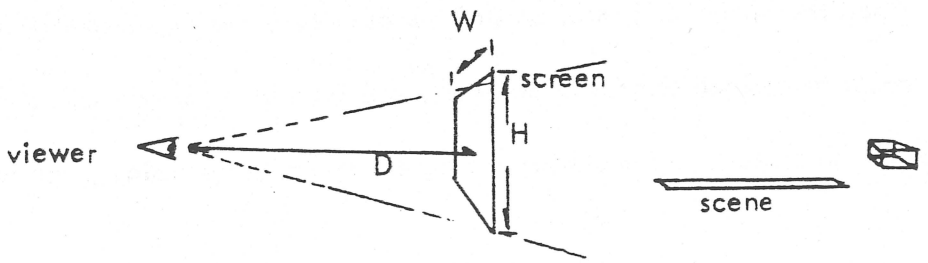
The normal clipped projection mode performs the traditional clipping functions of off-screen line elimination and partially on-screen line clipping. The non-clipped projection mode totally eliminates partially on-screen lines as well as off-screen lines. This mode is preferable for small objects that tend to leave the screen very quickly such as runway markers, trees at a distance, small windows on buildings, etc. Using this mode on small objects keeps the 3D-to-2D converter from wasting valuable time clipping these trivial objects thereby increasing the overall projection rate.

A special command may be submitted in the input array to switch between the two clipping modes. The CLPSW command sheet (04) gives details of this command.

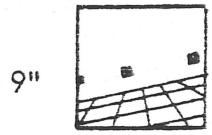
ASPECT RATIO AND SCREEN DIMENSION CONTROL

The A2-3D1 program has total field of view, screen aspect ratio, screen bit ratio, and screen centering control. In order to properly control these functions it is first necessary to understand their operations. Since the four areas of control work together,they will be described together.

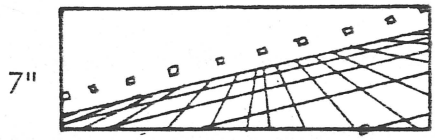
Field of view is similar to the wide angle and telephoto characteristics of a camera. A wide viewing angle will fit more of a scene into a projection. When viewing an image on a screen, the geometricallly correct field of view is determined by the screen width and viewer's distance from the screen.



Aspect ratio refers to the shape of a rectangular viewing window. A different view is seen from a long, narrow viewing window than from a square one. The 3D to 2D converter must know the aspect ratio (the width to height ratio) of the screen to know where to clip the lines.



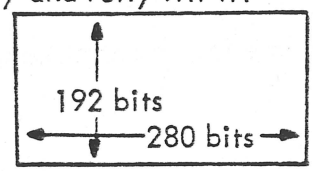
a) a 1:1 aspect ratio



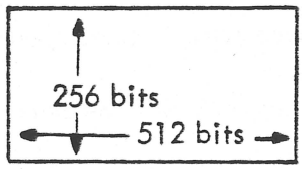
b) a 3:1 aspect ratio

It should be noted that most standard television sets have an aspect ratio of approximately 4:3.

Bit ratio is strictly a display generator characteristic. Bit map displays come in many sizes (64 x 96, 256 x 256, 270 x 192, etc) and few of them have 1:1 bit ratios. In the final projection, the 3D to 2D converter has to know how many bits wide and high the display device is in order to make sure the output values address the screen properly and fully fill it.



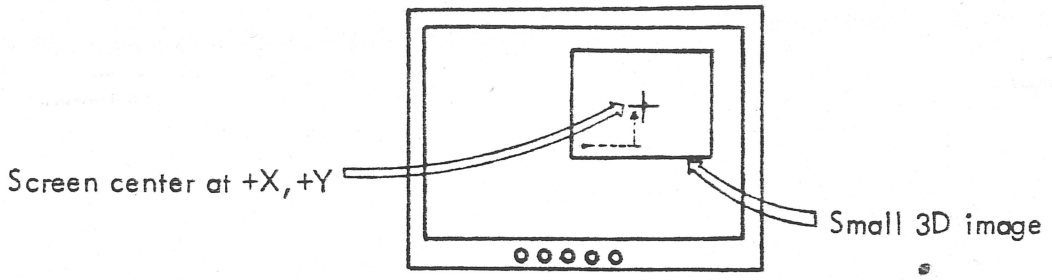
a) 280:192 bit ratio
(APPLE II)



b) 512:256 bit ratio
(MATROX ALT-512)

Screen centering refers to the placement of the rectangular projected frame on the screen. When the bit ratio is set to fill the entire screen, the screen center should be 0, 0. When the bit ratio is reduced (for the purpose of putting a smaller image on the screen in order to include some text perhaps), a 0, 0 centered image will shrink toward the center of the screen. By changing the centering or "screen bias", the small image

may be moved over to one side or corner of the screen. This makes screen splitting effects possible



The 0E and 0F commands may be put into an input array to adjust the field of view, aspect ratio, bit ratio and centering. The screen aspect ratio in the A2-3D1 program is setup to be correct for the APPLE II screen using an average size (17" • 4:3 aspect ratio) television set, but non-APPLE II users who use different display devices, users desiring special effects, and perfectionists who demand 100% aspect ratio and field of view correctness must use the 0F and 0E commands. See the 0F and 0E command sheets for details and formats of these functions.

SPECIAL APPLE II FUNCTIONS

The A2-3D1 high performance 2D driver section offers the user many features above and beyond standard 3D line drawing. The following capabilities are provided:

1. 2D line drawing
2. page 1 or page 2 line draw selection
3. page 1 or page 2 black erasing
4. page 1 or page 2 white filling
5. page 1 or page 2 display select
6. 2D point plot
7. "exclusive or" line drawing for selective erase
8. screen initialization (versatile)
9. quick and easy screen initialization

The command sheets further describe these functions. Proper application of these functions allow you to smooth-out flicker in animation, mix 3D and 2D imagery, set mixed or full graphics, and much more. Sections of this manual will expand on these functions.

GETTING AN IMAGE ON THE SCREEN

The basics of object construction in 3D space and display control have already been covered. Briefly, objects consisting of points and lines must be cunstructed with pure points, start points, continue points, and ray points. The viewer may also control his X,Y,Z location and pitch, bank and heading. In order to get images on the screen one must also know how to put all of the commands in an input array and how to call the A2-3D1 subroutine. An understanding of using the program with other software such as BASIC is also desirable. These items will now be covered.

INPUT ARRAY FORMAT. The input array consists of a series of pure, start, continue, and ray points and control commands stored sequentially in computer memory starting at location 1B00 hex (6912 decimal). Each of the points consists of 7 bytes:

1. An 8-bit code indicating a pure, start, continue, or ray point.
2. A 2-byte 16-bit 2's complement, byte swapped, space coordinate X
3. A 2-byte 16-bit 2's complement, byte swapped, space coordinate Y
4. A 2-byte 16-bit 2's complement, byte swapped, space coordinate Z

The term byte swapped implies that the most significant half of the 16-bit word appears in the lower addressed byte.

Each display control command consists of from one to ten bytes (depending on the control command). Like the point formats, the control commands always use the first byte as an 8-bit code indicating the command. Command function control bytes follow the command code byte. Points and control commands may appear in any order in memory as long as they sequentially follow one another. The control commands given in an input array apply to all points and lines following that control command.

Finally, an end of file code must end the input array. Any non-command code will do.

The general format is thus:

Address	Data	Function
1B00	10	"initialize the screen" code
1B01, 1B02	08, 00	erase the screen
1B03	05	set X, Y, Z, pitch, bank, heading to:
1B04, 1B05	01, 00	X value (byte swapped 0100 hex)
1B06, 1B07	34, 12	Y value (byte swapped 1234 hex)
1B08, 1B09	79, 19	Z value (byte swapped 1979 hex)
1B0A	00	0 degrees pitch
1B0B	25	25 degrees bank
1B0C	01	01 degree heading
1B0D	01	start point code
1B0E	00, 01	X (100 hex)
1B10	00, 02	Y (200 hex)
1B12	21, 43	Z (4321 hex)
1B14	02	continue point code
1B15-1B1B	x, y, z	X, Y and Z coordinate of continue pt. (same as st. pt.)
1B1C	03	ray point code
1B1D-1B22	x, y, z	X, Y and Z coordinate of ray point (same as st. pt. format
1B23	08, 02	fill the screen with all white
1B25	79	end of file

This input array would initialize the screen, erase it, set the viewers location and viewing direction to the values shown, draw 2 lines, and finally make the screen go completely white. This would take a total of approximately 200 milliseconds. Why anyone would want to project this particular image is questionable since the lines would appear as a flash before the screen turned completely white, but the sequential interpretation of the input array would project precisely what was just described. Two important points to note about this example are the start of the array address (1B00) and the end of file code on the end (a 79 was used in this case).

The familiarization section of this manual takes a step-by-step approach to familiarization of the input array concept and helps describe it. The output array section presents

a large input and output array along with a good figure that is also helpful in visualizing the input array concept. Input arrays are really quite easy to work with as they are similar to BASIC or assembly language programs: they execute command (instruction) after command in a sequential order. The input array's "instructions", however, are graphics commands.

COMMAND SHEETS. The following command sheets describe, in detail, the commands available to the A2-3D1 user. It is very important to understand the input array concept before working with these commands. Before building large data bases it is wise to get to know all commands and the RULES regarding their use.

Also note that not all commands must be used in an input array. All screen control commands are, in fact, optional. The A2-3D1 program will assume a 0, 0, 0 X, Y, Z viewer location, a wide angle view, and an APPLE II screen unless told otherwise.

Addresses used in the examples in the command sheets have no real significance. They are there only to show how a particular command fits into a given area of memory. X, Y, and Z coordinates of points are also chosen to be random, representative values that have no special meaning.

The letters "lsb" and "msb" in the command sheets stand for least significant byte and most significant byte respectively.

PURE POINT — COMMAND = 00 hex PNT

OPERATION: This command specifies a point in space using X, Y, and Z coordinates. The 3D-to-2D converter converts this into a 2D point to be projected on a screen. The point is not displayed if it falls off the screen.

BYTES: 7 bytes.

FORMAT: 00, X lsb, Xmsb, Y lsb, Y msb, Z lsb, Z msb in sequential memory locations where x, y, and z are double precision, two's complement, byte-swapped values.

EXAMPLE:

Input:

Address	Data	Meaning
1B39	00	PNT code
1B3A, 3B	34, 12	X coordinate (1234 in this case)
1B3C, 3D	79, 19	Y coordinate (1979 in this case)
1B3E, 3F	00, 01	Z coordinate (0100 in this case)

Result: The 3D point 1234, 1979, 0100 would be projected onto the 2D screen.

USES: Points are useful where single small dots are required (stars for example). They are good for games, astronomy, space flight, molecular modeling, and object detailing.

RULES: Any number of points may be used in any location in the input array.

START POINT

COMMAND = 01 hex SPNT
= 01 dec

OPERATION: SPNT specifies the beginning of a line in space using X, Y, Z coordinates. The 3D-to-2D converter takes the start point and the following continue point and projects a 2D line on the screen. The line will be eliminated if off the screen and will be clipped if it is partially on the screen. If it is partially on the screen and the clipper has specifically been turned off the line will be eliminated.

BYTES: 7 bytes.

FORMAT: 01, x lsb, x msb, y lsb, y msb, z lsb, z msb in sequential memory locations where x, y, and z are double precision, two's complement, byte-swapped values.

EXAMPLE:

Input:

Address	Data	Meaning
1C08	01	SPNT code
1C09, 0A	00, 01	X coordinate (0100 hex)
1C0B, 0C	77, 00	Y coordinate (0077 hex)
1C0D, 0E	21, 43	Z coordinate (4321 hex)
1C0F	xx, xx	Continue point code and coordinates.

Result: The line defined in 3D space by the start point and the following continue point is projected as a 2D line on the screen.

USES: The line is the most-used element in 3D projections, and every line starts with a start point (directly or indirectly). Wherever lines are needed, the start point will be useful.

RULES: A continue point must follow a start point. No other command or type of point will do. If a continue point doesn't follow a start point, the following command, whatever it may be, will be assumed to be a continue point causing a bad line at best and an out-of-command synchronization problem and subsequent crash at worst.

CONTINUE POINT

COMMAND =02 hex CPNT
=02 dec

OPERATION: CPNT specifies the continuation of a line in space using X, Y, Z coordinates. The line can be a continuation from a start point or from another continue point or ray. The line will be projected by the 3D-to-2D converter onto a 2D screen. The line will be eliminated if off the screen and will be clipped if it is partially on the screen and the clipping mode is in effect. If it is partially on the screen and the clipper has specifically been turned off, the line will be eliminated. Future continue and ray points pick-up where this continue point leaves-off.

BYTES: 7 bytes.

FORMAT: 02, x lsb, x msb, y lsb, y msb, z lsb, z msb in sequential memory locations where x, y, and z are double precision, two's complement, byte-swapped values.

EXAMPLE:

Input:

Address	Data	Meaning
1f06	xxxx	previous start, continue, or ray point
1f0d	02	CPNT code
1f0e, 0f	00, 34	X coordinate (3400)
1f10, 11	01, 00	Y coordinate (0001)
1f12, 13	53, 39	Z coordinate (3953)
1f14	xxxx	Continue point, ray or other command

USES: As with the start point, the continue point will be useful wherever lines are needed.

RULES: The CPNT has no specific rules as it will always continue a line from where the last one was drawn. It is wise to check that a previous line was indeed drawn in the input array, otherwise the continue point will assume a random location for the start of the line it generates.

RAY POINT

COMMAND =03 hex RAY
=03 dec

OPERATION: The RAY specifies the continuation of a string of lines in space using X, Y, Z coordinates, but unlike the continue point that advances the line drawing "cursor" for the following lines, the RAY leaves it at its original location. The 3D-to-2D converter takes the start point (original position of the line drawing cursor) and the ray point and draws a 2D line on the screen, leaving the cursor not updated. The line will be eliminated if off the screen and will be clipped if partially on the screen. If it is partially on the screen and the clipper has specifically been turned off, the line will be eliminated.

BYTES: 7 bytes.

FORMAT: 03, x lsb, x msb, y lsb, y msb, z lsb, z msb in sequential memory locations where x, y, and z are double precision, two's complement, byte-swapped values.

EXAMPLE:

Input:

Address	Data	Meaning
1b06	xxxx	previous continue or ray point
1b0d	03	RAY code
1b0e, 0f	00, F4	X coordinate (F400)
1b10, 11	12, 34	Y coordinate (3412)
1b12, 13	00, 0A	Z coordinate (0A00)
1b14	xxxx	Continue point, ray or other command

Result: The ray defined in 3D space by the RAY point and the previous cursor position is projected as a 2D line on the screen.

USES: Rays are useful where lines with a common vertex are required. Starburst type patterns, polyhedra, and tick marks on a chain of continue lines are just a few examples.

RULES: A ray point must never follow a start point. If it does, it will be treated as a continue point. If you need a RAY after a start point, simply turn the line around and us a CONTINUE point. RAYs may follow continue points and other RAYs with no problems.

CLIPPER CONTROL

COMMAND =04 hex CLPSW
=04 dec

OPERATION: CLPSW enables the user to turn the 3D-to-2D converter's clipping function off or on. In clipped mode, lines that fall totally off the screen are eliminated and lines that are partially on the screen are cut down to size. In non clipped mode, any line that falls partially or totally off the screen is eliminated.

BYTES: 2 bytes

DEFAULT: Upon 3D-to-2D converter entry the clipper is turned on. Every time the 3D-to-2D converter is called, the clipper is turned back on before display file interpretation.

FORMAT: 04,00 in sequential memory locations turn the clipper on while 04, 01 in sequential memory locations turns the clipper off. 04, n where n is anything but 00 or 01 aslo turns the clipper off.

EXAMPLE:

Input:

Address	Data	Meaning
1B00	xxxx	Lines and other commands in clipped mode
1B72,73	04,01	turn clipper off
1b74	xxxx	Lines and other commands in nonclipped mode
1C08,09	04,00	turn clipper on
1C0A	xxxx	Lines and other commands in clipped mode

Results: The lines between 1b74 and 1C07 will be projected in non-clipped mode.

USES: Nonclipped projection is primarily meant to be a time saving feature. Small objects like small windows, runway markers, objects at a distance, etc. don't cause any loss of realism if they drop off the screen all at once, so it wastes time clipping them. Nonclipped projection should be used in animation where projection rate is of prime importance.

RULES: The clipper can be turned on or off as desired. Although values other than 01 will turn the clipper off, its a good idea to stick to 01 as a value since future versions of this program may make use of some of these other variables.

VIEWERS POSITION

COMMAND = 05 hex EYE
= 05 dec

OPERATION: EYE specifies the viewer's X, Y, Z location in space and the viewer's Pitch, Bank, and Heading. The 3D-to-2D converter uses this information to project the proper view of the scene.

BYTES: 10 (decimal) bytes.

DEFAULT: Upon 3D-to-2D converter entry the viewer's location and direction of view is set to x=0, y = 0, z=0, pitch = 0, bank = 0 and heading = 0. The viewers location will remain at this location until an 05 command is encountered.

FORMAT: 05, x lsb, x msb, y lsb, y msb, z lsb, z msb, pitch, bank, heading in sequential memory locations provides the 3D-to-2D converter with viewer's location and direction of view. The X, Y, and Z values are 2's complement byte-swapped double precision with a +and - 32767 range. The pitch, bank and heading are "pseudo degrees" (256 divisions to a circle).

EXAMPLE:

Input:

Address	Data	Meaning
1b03	05	EYE code
1b04, 05	00, 06	X viewer position (0600)
1b06, 07	C4, 13	Y viewer position (13C4)
1b08, 09	25, E4	Z viewer position (E425)
1b0a	00	Pitch (00 pseudodegrees)
1b0b	20	Bank (20 pseudodegrees)
1b0c	Ae	Heading (Ae pseudodegrees)

Result: All lines after 1b0c are projected with the X, Y, Z and pitch, bank and heading given in the EYE command.

USES: The EYE command is used whenever one wants to change his location in space or direction of view.

RULES: The EYE command can be used wherever desired in an input array. There are no restrictions as to how many times it may be used.

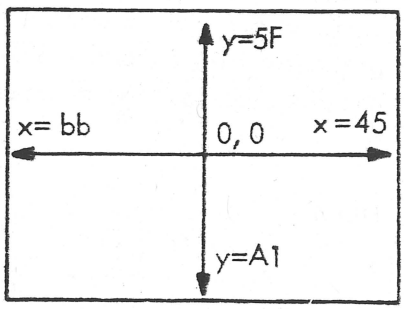
DRAW A 2D LINE

COMMAND = 06 hex LIN2D
= 06 dec

OPERATION: LIN2D is a command that feeds line drawing information directly into the APPLE II 2D driver. It takes a start and end point specified in the command and draws it on the APPLE II high resolution screen in white. Aspect ratio, bit ratio, screen centering and array turn-on commands have no effect on this command.

BYTES: 5 bytes.

FORMAT: 06, x, y, x', y' in sequential memory locations plots the line x,y to x',y' on the APPLE II screen. X values may be in the range of + and - 45 hex (bb to 45) and Y values may range from +5f to -5f (A1 to 5f hex). The screen is layed out with 0,0 at the center and increasing X and Y to the right and up respectively.



EXAMPLE:

Input:

Address	Data	Meaning
1b0f	06	line code
1b10, 11	02, ff	line start point (x, y)
1b12, 13	40, 32	line end point (x', y')

Results: The line from 02, ff to 40, 32 is drawn.

USES: This command is used for drawing additional 2D images on the screen, overlapping or around the borders of a 3D image.

RULES: This is strictly an APPLE II command and should not be used on any other machine. It puts no entry into the output array. NEVER USE X,Y VALUES OUTSIDE OF THE SPECIFIED RANGE. If you do, you will "overshoot" the line drawer's tables and draw outside of screen memory, thereby destroying the program or BASIC.

DISPLAY SCREEN SELECT

COMMAND = 07 hex DISP
= 07 dec

OPERATION: DISP feeds screen selection information to the APPLE II's screen control unit. This command can specify text, mixed, or full-screen graphics modes, display page 1 or 2, and high or low resolution modes.

BYTES: 2 bytes.

FORMAT: 07, n in sequential memory locations sets an APPLE II screen mode. The value n determines which mode will be set.

n	mode
50 hex	set color graphics mode
51 hex	set text mode
52 hex	clear mixed graphics mode
53 hex	set mixed graphics (4 text lines)
54 hex	set page 1 graphics
55 hex	set page 2 graphics
56 hex	clear HI-RES mode
57 hex	set HI-RES mode

EXAMPLE:

Input:

Address	Data	Meaning
1b00	07,53	set mixed graphics (4 lines text)
1b02	07,50	set color graphics mode
1b04	07,57	set high resolution mode
1b06	07,55	set page 2 graphics

Result: This sequence of commands would cause page 2, high resolution mixed graphics mode to be displayed on the screen.

USES: This command is used to select which screen will be viewed and can also control special effects such as ping-ponging between screens.

RULES: This command actually sends a 0 out to memory location C0XX where XX is the value specified by n in the command. Using other n values will cause other control events to occur. Avoid using other n values.

ERASE SCREEN

COMMAND = 08 hex ERAS
= 08 dec

OPERATION: ERAS causes the APPLE II screen to be erased or filled with black or white. Page one or page two erasures are performed depending on the user-specified value following the command.

BYTES: 2 bytes.

FORMAT: 08, n in sequential memory locations causes a screen erase or fill to occur. The value n defines the action as follows:

n	result
00	erase page 1
01	erase page 2
02	fill page 1 with all white
03	fill page 2 with all white

EXAMPLE:

Input:

Address	Data	Meaning
1b94	08, 00	erase page 1

Result: Page 1 is erased.

USES: Screen erase is used to clear the display screen before drawing a new image. Screen fill is useful for black-on-white images using the exclusive-or line drawing function.

RULES: Care must be taken in erasing the screen as it actually wipes-out 8K of memory to perform the action. The proper screen must be erased. Never erase page 2 if there is a program in the page 2 memory space. Likewise with page 1.
This is strictly an APPLE II command and shouldn't be use in non APPLE II applications (unless you need a very fast 8K memory eraser!)

WRITE SCREEN SELECT

COMMAND = 09 hex DRAW
= 09 dec

OPERATION: DRAW selects which screen (page 1 or page 2) the APPLE II line drawer will draw on.

BYTES: 2 bytes.

DEFAULT: Unless specified otherwise, page 1 is selected for line drawing. Page 1 is in effect upon program loading, but is not returned to on every call of the 3D-to-2D subroutine. If page 2 is selected, it remains selected until another write screen command is encountered.

FORMAT: 09, n in sequential memory locations selects page one or two for line drawing and point plotting as follows:

n	result
00	draw on page one
01	draw on page two
all others	draw on page two

EXAMPLE:

Input:

Address	Data	Meaning
1b79,7A	09,01	from now on, draw on page 2

USES: This command is useful in system configurations where page 2 instead of page 1 is to be used. It is also useful for special effects such as screen ping-ponging in pursuit of display smoothness.

RULES: This command must be used before all others in systems that are using page 2 only as a display page. If a line is drawn before this command, valuable program information in page 1 may be overwritten by the line. In screen ping-ponging cases, one must realize that the page selected remains in effect unless selected otherwise- - even from call to call. This is because this command used instruction modifying code to perform its task.

PLOT A 2D POINT

COMMAND = 0A hex PNT2D
= 10 dec

OPERATION: PNT2D is a command that feeds point plotting information directly into the APPLE II 2D driver. It takes the point specified in X, Y, Z coordinates in the command and plots it on the APPLE II screen. Aspect ratio, bit ratio, screen centering and array turn-on commands have no effect on this command.

BYTES: 3 bytes

FORMAT: 0A, x, y, in sequential memory locations plots the point x, y on the APPLE II screen. X values may be in the range of + and - 45 hex (bb to 45) and Y values may range from +5f to -5F (A1 to 5f). The screen is layed out with 0, 0 at the center and increasing X and Y to the right and up respectively. See the 06 command sheet for a diagram of this coordinate system.

EXAMPLE:

Input:

Address	Data	Meaning
1b42	0A	PNT2D code
1b43, 44	23, 4F	x, y of the point to be plotted (X=24, Y=4F)

Results: The point 24, 4F is plotted on the screen

USES: This command is used for drawing additional 2D images on the screen, overlapping or around the borders of a 3D image.

RULES: This is strictly an APPLE II command and should not be used on any other machine. It puts no entry into the output array. NEVER USE X, Y VALUES OUTSIDE OF THE SPECIFIED RANGE. If you do, you will "overshoot" the line drawer- point plotter's tables and plot a point outside of screen memory, possibly destroying a program or BASIC.

INTERPRETIVE JUMP

COMMAND = 0B hex JMP
= 11 dec

OPERATION: This command causes the 3D-to-2D converter to continue reading the input array starting at a new address in memory. The new address is specified in the JMP command.

BYTES: 3 bytes.

FORMAT: 0B , a lsb, a msb will cause interpretation of the input array to resume at address "a" where a is a byte-swapped value.

EXAMPLE:

Input:

Address	Data	Meaning
1b75	0B	JMP command code
1b76,77	08,20	jump address (2008 hex)
2008	xxxx	next point or command to be read

USES: The JMP command can be used to break the input array into a number of small pieces that can be placed at convenient locations in memory. The JMP command can also be used to jump to an output array that has just been created. This output array will then be read as an input array of points and lines. See the discussion on SMOOTHING in this manual for more details.

RULES: The JMP command may be used at any time. Care should be taken to make sure the address is byte-swapped, or the interpreter will resume reading in a random location. Input array segments with a JMP at the end require no end of file mark, but at the end of the final segment an end of file mark is required. Be careful not to cause an array to jump to itself. The A2-3D1 program will go into an endless display generation loop if this is done.

SET LINE DRAWING MODE

COMMAND = 0C hex LMODE
= 12 dec

OPERATION: This command selects between normal ("or") line drawing and exclusive "or" line drawing, and turns off output array generation.

BYTES: 2-bytes.

DEFAULT: Upon 3D-to-2D converter loading, the normal line drawing mode is entered. Once changed to exclusive "or" line drawing, the program remains in this mode until specified otherwise- - even from call-to-call.

FORMAT: 0C, n in sequential order in memory causes either normal or exclusive "or" line drawing modes to be entered. An n of 00 specifies normal mode, and an n of 01 specifies exclusive "or" mode.

EXAMPLE:

Input:

Address	Data	Meaning
1b80, 81	0C, 01	draw the following lines in exclusive "or" mode
1b82	xxxx	lines to be drawn (in exclusive "or" mode)
1c56 , 57	0C, 00	switch back to regular line drawing
1c58	xxxx	more lines to be drawn (in normal mode)

Result: The first set of lines are drawn as exclusive "or" lines, and the second set as normal lines.

USES: Exclusive "or"lines can be used in a number of different ways. They can be used to draw black on white, white on black or can be used to selectively erase lines. The section on selective erasing in this manual should be consulted for details.

RULES: This command can be used nearly anywhere in an input array. Remember that the program will remain in the line drawing mode it was last in until changed; even from call-to-call. This is caused by the instruction modifying nature of this function.

TURN ON OUTPUT ARRAY — COMMAND = 0D hex = 13 dec — ARRAY

OPERATION: This command causes an output array to be generated instead of letting the scene be drawn on the APPLE II screen. An arguement within the ARRAY command specifies the address where the output array will start.

BYTES: 3-bytes.

DEFAULT: Upon input array entry, and every time the 3D-2D program is called, the output array generating feature is turned off.

FORMAT: 0D, a lsb, a msb in sequential memory locations causes an output array to be generated starting at the location specified by address a in the command. The a arguement is byte-swapped. Array generation ends when an end of array command or 0C (set line drawing mode) command is encountered. The end of file is denoted by an end of array mark (a 79 hex).
The output array is identical in construction to the input array and consists of 0A (points) and 06 (2D lines) only. The output array may be interpreted as an input array by the A2-3D1 program. The 0C command turns off output array generation.

EXAMPLE:

Input:

Address	Data	Meaning
1cd0	xxxx	lines being drawn on APPLE II screen
1d78	0D,	turn on output array
1d79,7a	00, 1f	create array at 1f00 hex.
1d7b	xxxx	lines to be placed in output array

Result: The lines starting at 1d7b are put into the output array (after 3D-to-2D conversion).

USES: The output array is useful on non-APPLE II machines where seperate line drawing routines are being used. The output array can also be used to save images for later display , and to hold-off projection of 2D lines until all the 3D transformations have been done, resulting in a smoother display.

RULES: Care should be taken in setting the output array's address. It should not overlap a program or screen display area. APPLE II screen control commands should not be used while creating an output array. They create no array entries but still affect the APPLE II screen functions and memory.

SCREEN SIZE SELECT

COMMAND = 0E hex SCRSZ
= 14 dec

OPERATION: This command provides the 3D-to-2D converter with screen bit ratio and screen centering information. Arguments in the command specify screen bit height and width, and X and Y screen center locations.

BYTES: 5-bytes

DEFAULT: Screen size on program load-up is preset to correspond to the APPLE II screen in the white mode (width 140, height 192 decimal). The screen center is set to zero.

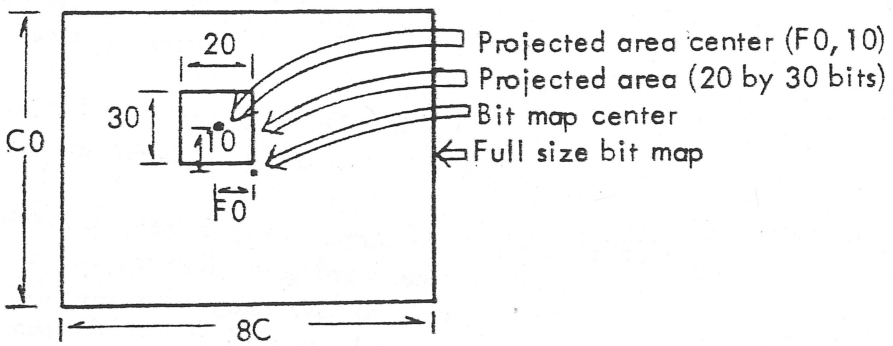
FORMAT: 0E, scr. width, scr. height, scr. X center, scr. Y center, in sequential memory bytes set the screen bit ratio and center. Screen height and width can reach a maximum of 256 x 256, and screen center can be placed anywhere within a + and - 127 range.

EXAMPLE:

Input:

Address	Data	Meaning
1F19	0E	SCRSZ code
1F1A	20,30	bit ratio is 20= width and 30 = height.
1F1C	F0,10	screen center is F0 = X and 10 = Y

Result: The 3D images are projected onto the following screen:



USES: This command is useful for setting screen size on non-APPLE II devices, and for moving the screen around on all devices.

RULES: Don't center the screen in such a way as to cause the image to fall outside of the display devices plotting boundaries. This is especially true in the APPLE II case as the line drawer will destroy memory outside of the display page if you do.

FIELD OF VIEW SELECTION

COMMAND = 0F hex = 15 dec FIELD

OPERATION: FIELD provides the 3D-to-2D converter with field of view and aspect ratio information.

BYTES: 7-bytes.

DEFAULT: Upon subroutine entry, the field of view is set to a wide angle view with a 4:3 aspect ratio, assuming an APPLE II display device.

FORMAT: 0F, axr lsb, axr msb, ayr lsb, ayr msb, azr lsb, azr msb, in sequential order in memory set the aspect ratio and field of view.
Axr is the horizontal aspect ratio control. It is a 2-byte 2's complement value that compresses the image in the X dimension thereby fitting more horizontal information into the screen. Wide screens will use low value axrs. Axr is a fractional value with 32767 representing 1.0. For a wide screen an axr value of .5 (16384) may be appropriate.
Ayr is identical to axr but works in the vertical direction. Tall screens use lower value ayrs. A very tall screen might use a .5 (16384) ayr.
Azr controls field of view. A very wide angle view is 32767. A very narrow view is 4000.
Negative field values produce mirror and backward images.

EXAMPLE:

Input:

Address	Data	Meaning
1F09	0F	Field of view code
1F0a	FF,5F	axr value = 24575 (wide screen axr)
1F0c	FF,7f	ayr value = 32767 (normal height)
1F0e	45,25	azr value = 9541 (telephoto view)

note: values after "= sign" is decimal equiv. of Data

Result: A 4:3 aspect ratio (wide screen) telephoto view will be projected.

USES: Field of view is used to create proper screen geometry for the display device being used. Field of view may also be used for zoom, mirror imaging and other special effects.

RULES: Field of view, once changed, remains the same from call-to-call unless intentionally changed again.

EASY INITIALIZE

COMMAND = 10 hex INIT
= 16 dec

OPERATION: This command puts the APPLE II screen into the high resolution, split graphics/text, page 1 viewing mode.

BYTES: 1 byte.

FORMAT: 10 in a memory location in the input array initializes the screen.

EXAMPLE:

Input:

Address	Data	Meaning
1b00	10	initialize the screen

Result: The screen goes into high resolution, split graphics/text, page 1 viewing mode.

USES: This command is a good way to start most page 1 input arrays. It puts the screen in an easy to use mode since 4 lines of text are provided on the bottom of the screen. This command avoids having to use 3 seperate 07 (display screen select) commands.

RULES: It is a good idea to use this at the beginning of input arrays. There are no location restrictions however.

NO OPERATION

COMMAND = 11 hex NOP
= 17 dec

OPERATION: This command performs no operation and is skipped-over by the 3D-to-2D converter.

BYTES: 1 byte.

FORMAT: An 11 in the input array is ignored and skipped-over.

EXAMPLE:

Input:

Address	Data	Meaning
1e05	11	NOP code
1e06	11	NOP code
1e07	xxxx	valid points and commands

Result: The two NOPs are skipped and the valid points and commands at 1e07 are processed.

USES: The NOP is good for filling space that might be used later (especially at the beginning of an input array where initializes, erases, and location information may be put). It is also good for eliminating unwanted commands without compressing the entire array to fill-up the gap left by the command's removal.

RULES: NOPs may be used anywhere. There are no restrictions.

SUBROUTINE CALLING. After the input array has been created and loaded at 1B00 hex (6912 decimal), the 3D-to-2D converter must be called using a BASIC CALL or assembly language JSR instruction.

The general purpose calling address of the A2-3D1 program is 0800 hex (2048 decimal). When called at this address, there is no need to worry about the 6502's zero page, X, Y, stack, A or P registers being modified by the 3D program. The program uses this memory and these registers, but it restores them before it returns from the subroutine. When working with large data bases, the saving and restoring of the zero page takes a very small fraction of a percent of the conversion time.

For very small data bases and the performance-minded user, a second subroutine call address is provided at 0803 hex (2051 decimal). This is the "fast calling address". Entry at this address side steps the 1.6 millisecond zero page and register save/restore sequence and gets right down to the 3D conversion. The destruction of zero page variables 60 to D0 hex and the processor registers (except for the stack pointer) is the price paid for this extra speed.

USING BASIC AND OTHER SOFTWARE. Using the A2-3D1 program with BASIC and other software is a simple proposition as long as the 0800 hex (2048 decimal) subroutine call is used. The only memory area the 3D graphics program uses under these conditions is the area between 800 and 1AFF hex. The chosen graphics display page on the APPLE II (locations 2000-3FFF or 4000-5FFF) is also modified if the A2-3D1 line drawer, point plotter or erase functions are used. No zero page variable or

register modification problems arise because all registers and the zero page are left intact.

The first thing to do when getting the A2-3D1 program up is to decide where in memory things should reside. Figure 15 shows a few possibilities, and one of these is likely to fit your system. Notice that disk systems running Disk APPLESOFT must have 48K of memory. With 10.5K of DOS, 12K APPLESOFT, 4.75K A2-3D1, and an 8K display page (35.25 K total), even a 36K system is inadequate. Tape or ROM APPLESOFT is therefore recommended for small and medium memory sized systems.

When using the program with any BASIC or DOS, two cautionary measures should be taken. First, make sure to set HIMEM and LOMEM to the proper range. If you dont, BASIC will write over A2-3D1. Also make sure not to erase or draw lines on a page you aren't using. If you do, A2-3D1 will write over BASIC or DOS. This is particularly important if you are using page 2 graphics. A write screen select command must precede your lines and points in the input array.

A BASIC USE EXAMPLE. An example of the use of the A2-3D1 program is in order at this time. Configuration B of figure 15 will be used. This configuration requires 16K of memory and Integer Basic. APPLESOFT ROM or APPLE PLUS BASIC will also work in this configuration although LOMEM and HIMEM must be written into the program using these two BASICs.

The following procedure can be used to get the program going:

1) Load the A2-3D1 tape (see the loading instructions in the appendix sections)
2) Get into Integer BASIC (ctrl B return)
3) Set LOMEM and HIMEM:

```
HIMEM: 8191     (sets HIMEM to 1FFF)
LOMEM: 7168     (sets LOMEM to 1C00)
```

1. Bits and pieces of zero page and 300-3FD available. See APPLE II reference manual.
2. Set LOMEM and HIMEM inside BASIC program.
3. Double buffered screens for smooth animation.
4. Use interpretive jump to break input array into two pieces if needed.
5. If no DOS is used, this area usable as program and input array area
6. 1B00 to 1D00 is input array, 1D00 to 1FFF is assembly language program.
7. 1B00 to LOMEM is input array, LOMEM to HIMEM (1FFF) is BASIC data.

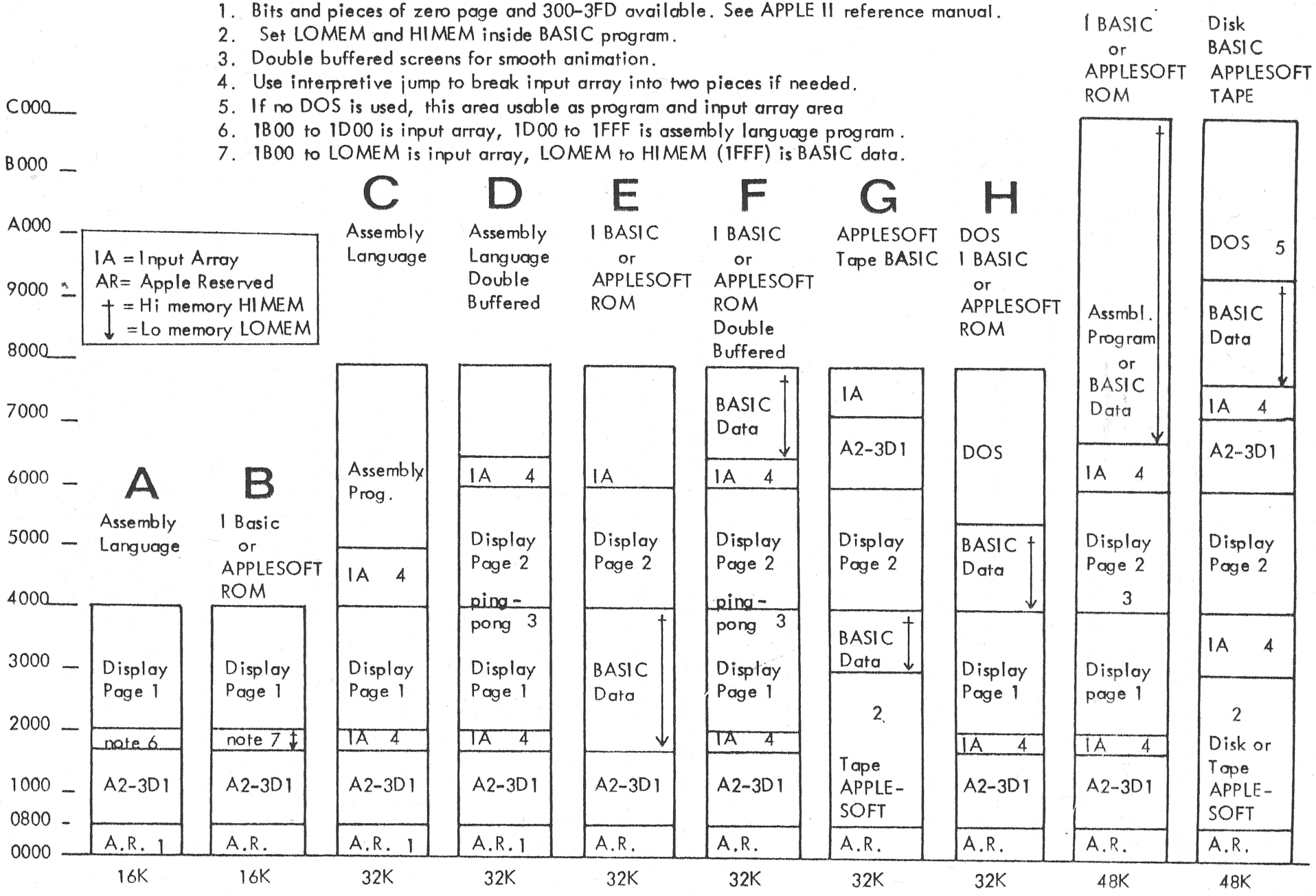
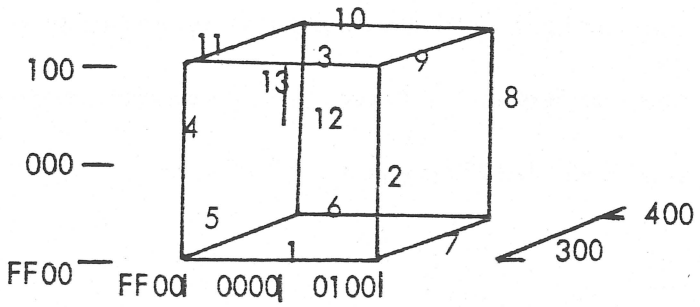


Figure 15 A2-3D1 /APPLE II system configuration guide.

Note that LOMEM and HIMEM must be written into the BASIC program in APPLESOFT. At this point, the A2-3D1 program, a small test cube and edge line input array at 1B00, and a 1K BASIC work area from 7168 to 8191 exist in memory. The input array at 1B00 is:

Address	Data	Meaning	
1B00	10	Initialize the APPLE II screen	
1B01	05, 00, 00, 00, 00, 00, 00	Location in space X, Y, Z	
1B08	00, 00, 00	Pitch, Bank, Heading	
1B0B	08, 00	Erase the screen	
1B0D	01, 00, ff, 00, ff, 00, 03	Line 1 start	cube
1B14	02, 00, 01, 00, ff, 00, 03	Line 1 continue	
1B1B	02, 00, 01, 00, 01, 00, 03	Line 2 continue	
1B22	02, 00, ff, 00, 01, 00, 03	Line 3 continue	
1B29	02, 00, ff, 00, ff, 00, 03	Line 4 continue	
1B30	02, 00, ff, 00, ff, 00, 05	Line 5 continue	
1B37	02, 00, 01, 00, ff, 00, 05	Line 6 continue	
1B3E	03, 00, 01, 00, ff, 00, 03	Line 7 ray	
1B45	02, 00, 01, 00, 01, 00, 05	Line 8 continue	
1B4C	03, 00, 01, 00, 01, 00, 03	Line 9 ray	
1B53	02, 00, ff, 00, 01, 00, 05	Line 10 continue	
1B5A	03, 00, ff, 00, 01, 00, 03	Line 11 ray	
1B61	02, 00, ff, 00, ff, 00, 05	Line 12 continue	
1B68	01, 00, 00, 00, 01, 00, 03	Line 13 start	edge line
1B6F	02, 00, 00, 80, 00, 00, 03	Line 13 continue	
1B76	79	End of array	



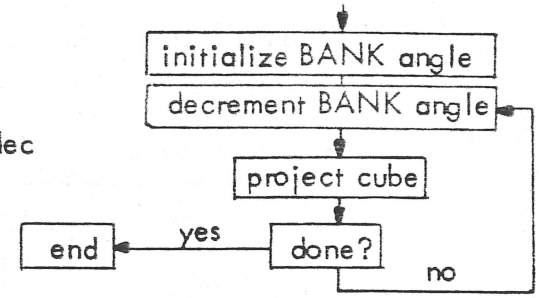
You may examine it with PEEKs to varify it if you wish. The subroutine call address is 2048 decimal. The cube may be projected now by typing:

```
> CALL 2048
```

The cube should appear on the APPLE II screen.

Notice that address 1B09 in the input array is the BANK parameter. This may be changed using POKEs. Enter the following program:

```
10 A=256
20 A=A-1
25 POKE 6921,A      note: 1B09 hex = 6921 dec
30 CALL 2048
40 IF A> 0 THEN GOTO 20
60 END
```



Running this program yields a rotating cube. The cube will rotate 256 pseudodegrees (one full rotation) and the program will end. Changing the POKE address to the PITCH parameter's address (1B08 hex, 6920 decimal) causes pitch changes, and changing it to the HEADING parameter's address causes heading changes.

Similar procedures should be used in getting any configuration up and running. The tape loading section tells how to get the 6000 origin version of the A2-3D1 program into memory if your system requires it (see figure 15, configurations G and J). Remember to use the subroutine call addresses shown in the **map** if the 6000 origin version is used.

The watchword in getting any program up and running is "avoid program-to-program overlap and interference". Never perform a screen erase over BASIC or a LOMEM-HIMEM over the A2-3D1 program.

OUTPUT ARRAY GENERATION CONCEPTS

Command 0E (turn on the output array) is described in the command sheets of this manual. This command is useful for non APPLE II owners who dont need or want the APPLE II screen projection and APPLE II owners who want smoother-running displays. It is necessary to understand exactly what this command does before it can be properly used, however.

The 6502 3D-to-2D converter takes the input array of 3D lines and transforms them, one-by-one, into start and end points of lines to be projected onto a screen. APPLE II users usually want lines to appear on the screen, and lines are automatically fed to the high speed line projector for the APPLE II -- unless the user has previously turned on the output array with the 0E command. If an output array is requested, the 3D-to-2D converter builds an output array that is very similar to the input array in memory and supresses the APPLE II screen display.

The output array consists of start and end points of 2D screen lines. Entries in the output array have the following format:

LINES

address	data	
1F94	06	line code
1F95	x	start pt. X
1F96	y	start pt. Y
1F97	x'	end pt. X
1F98	y'	end pt. Y

POINTS

address	data	
192A	0A	point code
192B	x	point X
192C	y	point Y

Notice that the coordinates have a range of + and - 127 units since they are single byte parameters.

The output array represents lines in the Sublogic Universal Graphics Interpreter format: the center of the screen is 0, 0 and X and Y increase in value moving to the right and up respectively. Figure 16 illustrates the screen coordinate system.

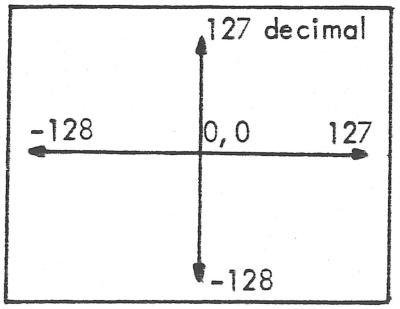


Figure 16. The screen coordinate system.

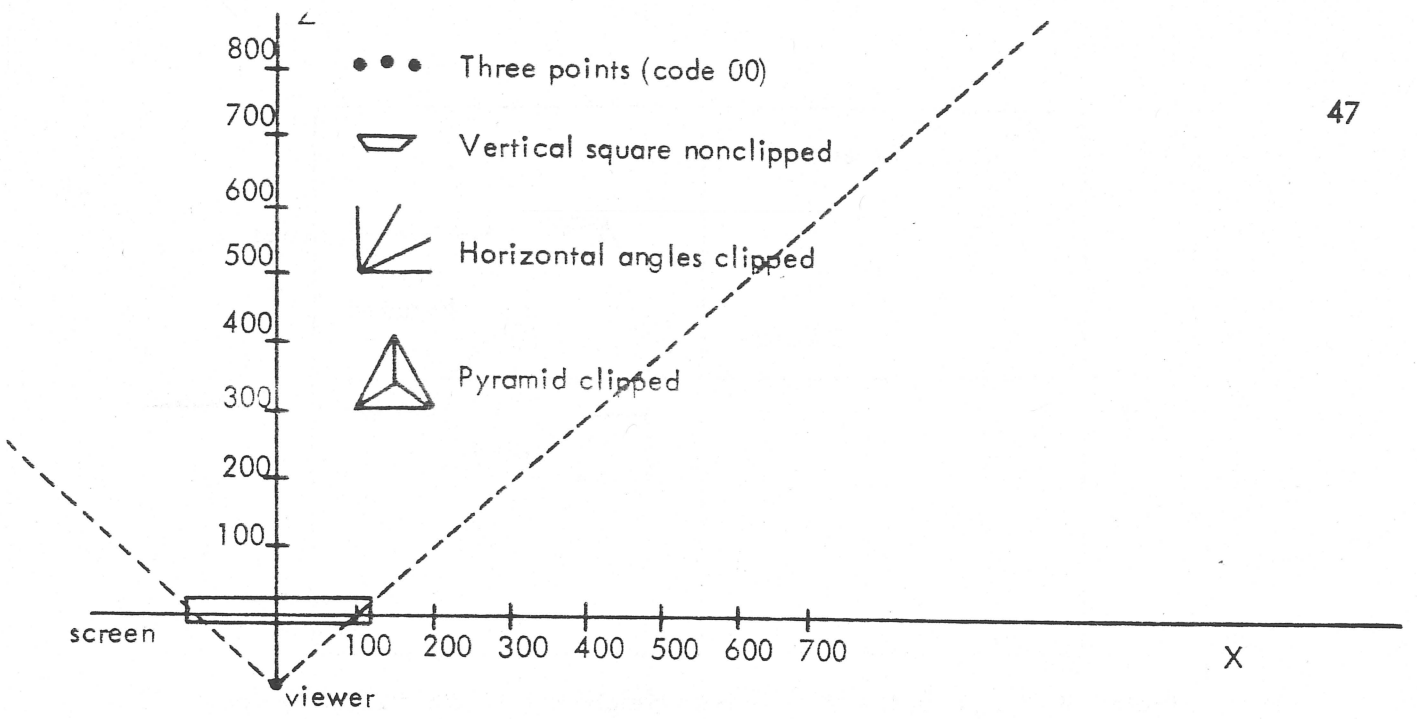
The 0E and 0F commands may be used to scale the field of view, bit width, or aspect ratio to suit the user's display device.

The 06, 07, 08, 09, 0A, 0C, and 10 commands (2D points and lines, APPLE II screen select, erase, write, exclusive or lines, and screen initialize) produce no outputs in the output array as these are strictly APPLE II 2D driver functions. These functions are not de-activated during array generation, however, so one must be careful not to erase output arrays generated in the high or low graphics page in memory (addresses 4000-5FFF and 2000, 3FFF respectively).

Figure 17 illustrates a good-size input array, and figure 18 shows the resulting output array.

Upon subroutine return or APPLE II line generation turn-on with the 0C command, an end of file mark (79) is placed in the output array.

The output array may be used in a number of different ways. Non APPLE II computers can use the array to generate images using line drawers suited to their particular display peripheral. The line drawing techniques section of this manual is a good place to start in generating these custom line drawer programs. The output array feature may also be used to save images on tape or disk. An image can be dumped and played-back later. The display smoothing section of this manualdescribes how output arrays may be used to reduce flicker in displays.

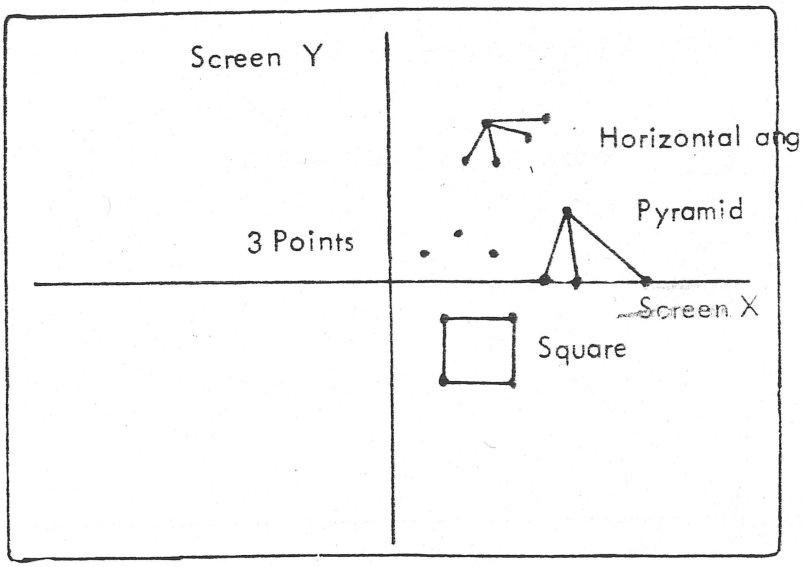



```
0D      ]  Turn on output
001F    ]  array at 1F00
0E      ]
FFFF    ]  Screen = 256x256
0000    ]  bits (decimal)
01      ]
0001    ]
0000    ]
0003    ]
02      ]
0002    ]
0000    ]
0003    ]  Pyramid
03      ]  clipped
8001    ]
0001    ]
8003    ]
02      ]
8001    ]
0000    ]
0004    ]
03      ]
8001    ]
0001    ]
8003    ]
02      ]
0001    ]
0000    ]
0003    ]
02      ]
8001    ]
0001    ]
8003    ]

01      ]
0001    ]
0004    ]
0006    ]
02      ]
0001    ]
0004    ]
0005    ]
03      ]
5001    ]  Horizontal
0004    ]  angles
0006    ]  (clipped)
03      ]
0002    ]
0004    ]
5005    ]
03      ]
0002    ]
0004    ]
0005    ]
0401    ]  Turn off clipper
01      ]
0001    ]
00fc    ]  Vertical
0007    ]  square
02      ]  (nonclipped)
0002    ]
00fc    ]
0007    ]
02      ]
0002    ]
00fd    ]
0007    ]

02      ]
0001    ]
00fd    ]
0007    ]
02      ]
0001    ]
00fc    ]
0007    ]
00      ]
0001    ]
8000    ]
0008    ]
00      ]
8001    ]  Pure points
c000    ]
0008    ]
00      ]
0002    ]
8000    ]
0008    ]
79      ]  End of file
```

Figure 17. An example input array (see next page for output).



Note: X, Y, Z, P, B, H = 00 Screen Height and Width = 256, Center = 0, 0

```
06]
2a
00
54
00]
06]
03
02
54
00]
06]
54
00
2f
00]
06]
03
02
2f
00]
06]
2f
00
2a
00]
06]
2a
00
03
02]]
```

The Pyramid

```
06]
15
54
19
65]
06]
19
65
1b
54]
06]
19
65
2f
5f]
06]
19
65
32
65]
06]
12
b8
24
b8]
06]
24
b8
24
ca]]
```

Horizontal Angles

Square

```
06]
24
ca
12
ca]
06]
12
ca
12
b8]]
0a]
0f
07]
0a]
17
0b]
0a]
1f
07]]
79]]
```

Square (continued)

3 Points

End of file mark

Figure 18. An example output array (generated by input array of Figure 17)

DISPLAY SMOOTHING

A smoothly flowing display is very desirable in any form of animation whether it be film, television, or 3D computer graphics. While film animators go to high frame rate cameras and television people go to high persistance phosphors, computer graphics users rely on double buffering, screen ping-ponging, adaptive screen erasing and array-based projections to achieve low flicker and realistic results. The A2-3D1 package has enough graphics commands to reduce flicker very substantially if used properly. Methods of smoothing displays will now be covered.

Two characteristics are important in display smoothness: frame-to-frame incremental difference and frame-to-frame discontinuity. Incremental difference refers to the degree of difference of two adjacent frames in an animation sequence. If there is little or no difference between frames, a very slow frame projection rate is sufficient for smooth animation. An animation of a clock's hour hand would look just as smooth at 5 frames per second as at one frame per minute.

Frame-to-frame discontinuity refers to actions during or between frame projection that tend to break up a sequence of frames' flow. Screen erasure is one example of a discontinuity. If a screen goes blank for a noticable time between frames, flicker is introduced, even if there is very little or nothing moving in the frame sequence.

The A2-3D1 program attacks the incremental difference problem through the use of hand-optimized assembly language and integer arithmetic. Fast projections are required for high frame rates, and assembly language is the fastest running language of all. There are a couple of things that can be done to improve frame rate:

1. Keep the scenes as well organized as possible. Large strings of start, continue, and ray points can effectively double 3D-to-2D conversion rate when compared to their start-continue pair equivalents.

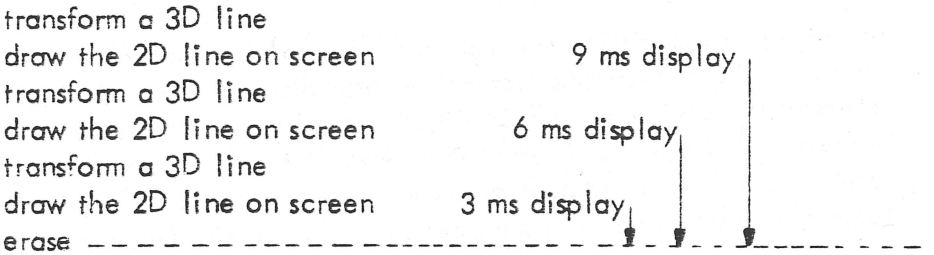
2. Don't use clipped projection on lines that don't need it. The clipping of tiny objects offers no display accuracy gain and reduces frame rates.

Frame-to-frame discontinuity can be a real problem on memory-based displays such as the APPLE II since it takes a long time to erase the screen's memory. A fast erase command helps smooth a display, and the A2-3D1 program has one. The 45 millisecond erase time (22 erases per second) is still quite slow on a human perception scale however.

The type of erase performed is also important in discontinuity reduction. An erase that seems to make all portions of the picture simultaneously fade-out is

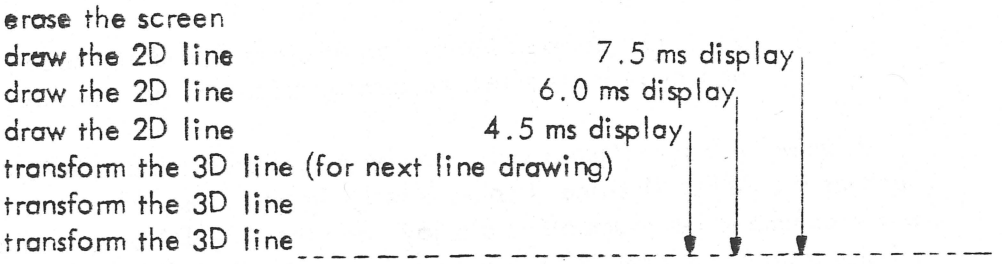
superior to one that sweeps from one side of the screen to the other or "folds the screen" like a set of blinds. This is due to the fact that the eye notices a few large sweeping motions more than many thousands of small fading motions. Increased intensity in the last-erased portion of the screen during a series of sweeping erases is also very noticable. Keeping these factors in mind, the A2-3D1's erase has been designed to fade the screen in a checkerboard pattern as quickly as possible.

The order of projection, transformation, and screen erasure also makes a difference in screen flicker. The standard projection sequence in 3D animation is:



This form of projection sequence has two problems. Lines in the display are on the screen for vastly different lengths of time, and the last lines are erased immediately after they are drawn. The first lines in the display will be very bright, and the last lines will be dimmly flickering. Screen flicker can be reduced by equalizing the amount of time lines are on the screen, and increasing the line erase after draw time.

This is where the array generating feature becomes useful to the APPLE II owner. By building an output array of the new frame while the old frame is on the screen, the screen erase can be delayed until the array of 2D screen lines is ready for projection. As an added bonus, only line drawing time is required between lines since the 3D-to-2D conversion has already been performed. The line generator generates the lines at a faster rate thereby reducing the time between display lines on the screen:



Array display can be automatically performed by putting an interpretive jump (0B command) to the output array on the end of the input array. The A2-3D1 program can interpret the input array's resulting output array and display the image by projecting lines. The price paid for the resulting smoothness is memory size. The output array must use a new area of memory.

The best screen discontinuity reductions are achievable through the use of both display pages of memory. This takes 16K for the display pages alone (on the APPLE II) and requires at least 32K of memory in an APPLE II system. The idea is to get one page ready with the next image while the other page is being displayed. When everything is ready, the display screen select command (07) is used to turn-on the ready page, thereby freeing the old page for an erase and subsequent image generation. The display screen select (07) command, erase screen high and low command (08) and the write screen select command (09) can be used to ping-pong between screens.

Variations of the double buffering scheme described above are also possible on other machines. The prerequisites include the ability to switch rapidly between two screen memories, and the availability of two screen memories. You must also be able to write into the non-displayed screen memory.

SELECTIVE ERASE AND EXCLUSIVE ORED LINES

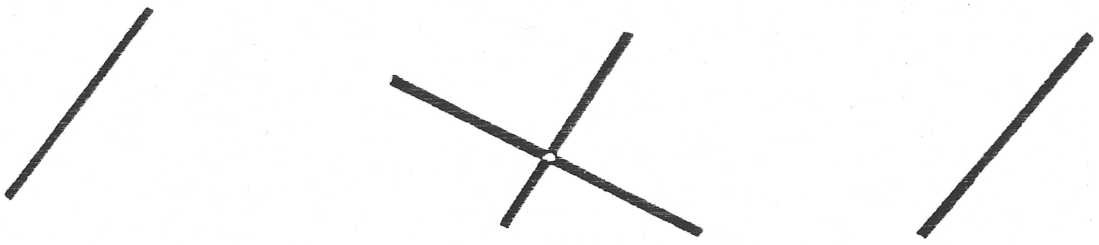
While memory-based display systems bring erase speed and line drawing speed disadvantages, they also bring a few advantages. The ability to read a screen image is one of these advantages. The A2-3D1 program is designed to make use of this advantage through an exclusive ored line and ored line drawing capability.

Plotting a point on the APPLE II requires a bit in a byte in screen memory to be turned on by setting it to one. One must be sure to leave the other bits in the byte undisturbed, however, as they represent other points that were plotted earlier. This requires byte reading, bit turn-on (by oring it with one), and byte write-back. Points are turned off in the same manner. Instead of turning a bit on, a bit is turned off. It is therefore possible to erase points and lines selectively. This enables extremely complex images to be drawn on a screen with one small simple object that freely moves about using selective line generation and erase of the small object. The whole scene needn't be regenerated so flicker is minimized.

All seems well and fine with this scheme until one notes that everything the small moving object moves across disappears. Selectively erasing an object in this manner not only erases the object itself, but every object with overlapping points.

It is possible to create a list of all the overlapped points and somehow restore them later, but there is a much cleverer way of obtaining nearly the same results: by using exclusive ors instead of simple ors to set and reset the points. If a point is plotted on black by exclusive oring it with a one, the point turns on. If plotted on white, the point turns off. The same exclusive or operation can therefore turn a point on or off, effectively toggling the point.

Turning points on and off are the components of a successful selective draw and erase capability, but there is more to this particular draw-erase pair than meets the eye. Figure 19 illustrates what happens when two exclusive ored lines intersect. Notice that the intersection point is turned off by the intersecting line when it is drawn and is restored when it is selectively erased.



a) original line b) intersecting line c) erased intersecting line

Figure 19. The affects of exclusive ored line drawing

The original line is actually restored to its original condition. If desired, the intersecting line could be made to sweep wildly across a screen filled with complex images without disturbing a single line.

The 0C command turns the exclusive ored line feature on (see the command sheets). Small objects may be projected and selectively erased by:

1. projecting the object into an output array
2. drawing the object with exclusive ored lines
3. drawing the object again with exclusive ored lines (thereby erasing it)

Output arrays needn't be used as the 3D converter performs the same sequence if called twice in a row, but there is no need to put the same image through the 3D-to-2D converter twice. The above method saves a great deal of time.

Experimentation with the exclusive ored line drawing function is necessary before using it extensively. Lines with intersections turned off give poor results in some applications.

Exclusive ored lines may also be used to draw black on white images. The screen must be initially filled with solid white using the 08 (erase/fill) command. Exclusive ored lines will turn points off as it draws across the white field.

USING THE A2-3D1 PACKAGE ON NON APPLE II SYSTEMS

The output array generating feature is provided specifically for non APPLE II 3D graphics users. As the output array generation section describes, an output array of start and end points of lines can be generated and the APPLE II 2D generator turned off altogether. By simply feeding the output array to a line drawing subroutine (see the section on line drawing methods in this manual), the image can be displayed on any display device.

Since the APPLE II line drawing routines aren't required in a non APPLE II application, this area of memory may be used for other programs. Addresses for the start point change as new versions of A2-3D1 are released, but you can find the start of the line drawer in memory by looking for the following sequence of bytes:

3, c, 30, C0, 6, 18, 60 hex repeated over and over again.

This is the line drawer's mask table and anything from this point on may be eliminated. Remember not to use any of the APPLE II line drawer commands (commands that dont generate output array entries) as these will call the APPLE II line drawer that no longer exists.

LINE DRAWING METHODS

The output array generated by the 3D-to-2D converter in the A2-3D1 package consists of start and end points of lines, so a display device that is capable of drawing lines is required to reconstruct the 2D image. Many display devices don't have this capability due to a lack of hardware or software support by the manufacturer. This section presents line drawing methods that will reduce this line drawing requirement down to the ability to plot a point. Software presented in this section is capable of calculating all the points that need to be turned-on on a screen to generate any given line.

A line on a display screen is a series of points that have been turned on. On a raster scan bit map (such as an APPLE II display screen) these points don't line up perfectly with an ideal line between the two end points because the bit map is arranged in a square matrix. The closest approximation to the line is generated. Software is required to determine which of these points should be turned on.

When writing programs that perform line drawing functions it is best to think of a line as a ratio of X movement to Y movement. A diagonal line may move one unit up for every three units across for example. In mathematical terms:

$$3Y=X$$

A computer plot of a line with a simple ratio of 3 to 1 is easy to visualize. The computer simply starts at the start point of the line, draws three dots across, moves up one unit, draws three dots across, moves up a unit, and so on, until the end point is reached. A computer can keep track of when to make the upward movement by setting up a variable in a program. A one can be subtracted from this variable every time a move to the right is made,and an upward movement and the addition of three to the variable can be performed during an upward movement. This variable is called the sum. The computer makes the upward movement when the sum is less than or equal to zero. Since the sum is being counted up by three and down by one in the above case, three times as many horizontal as vertical moves are made:

sum	line drawing action
3	move right and subtract 1 (change in Y) from sum.
2	move right and subtract 1 (change in Y) from sum.
1	move right and subtract 1 (change in Y) from sum.
0	move up and add 3 (change in X) from sum
3	move right and subtract 1 . . . . .

Notice that for every X (right) movement the change in Y is subtracted from the sum and for every Y(upward) movement the change in X is added to the sum.

A "sum-tracking" algorithm has just been described. This method works with more complex line ratios as well. A 15 to 17 ratio, for example, works as well as a 3 to 1 ratio. This method only requires simple adds, subtracts and conditional branches, so it is particularly well suited to assembly language implementations.

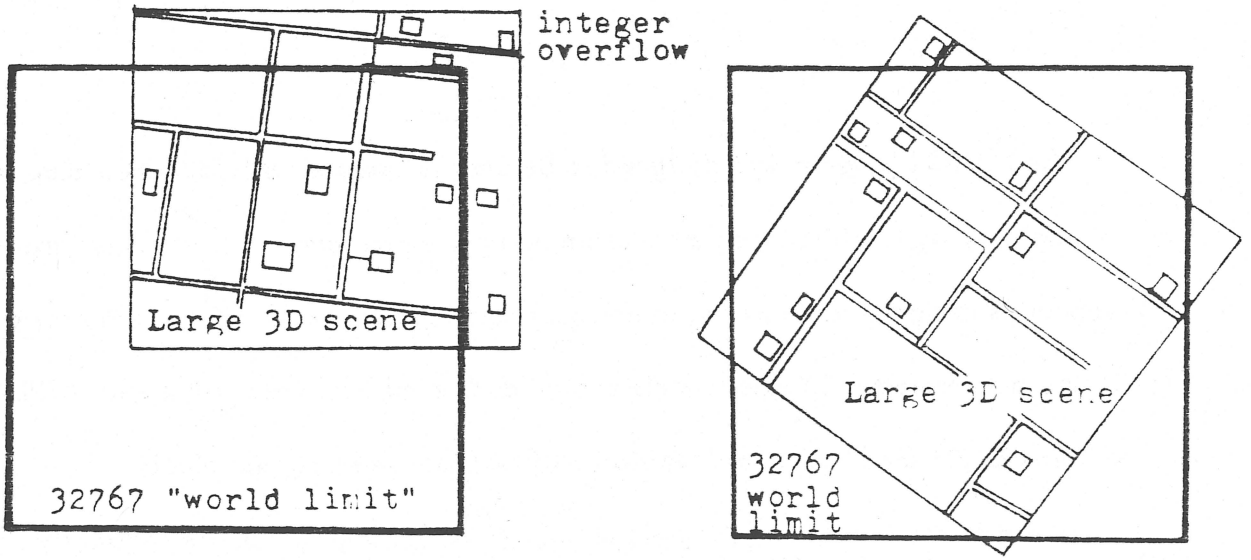
The BASIC program of figure 20 shows a simple implementation of the line drawer. It can be run in any BASIC and uses terminal input and print-outs for familiarization purposes.

```
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 20. BASIC line drawing subroutine.

SETTING - UP 3D SCENES

Although points in 3 coordinate space have a range of 32767 integer units in every direction, the actual range capability of the display program is limited by the viewer's position and direction of view. Since it is the world and not the viewer which actually moves, care must be taken to insure that the world always stays inside the 32767 unit limit. Too much translation and too big a data base size will cause integer overflow in the display program. Figure 21 a illustrates this condition. Figure 21 b shows that rotation can cause integer overflow due to the square world boundary. In most applications, limiting the scene to 10000 units in each direction will allow for good resolution, freedom of movement, and translation as figure 22 illustrates.



a) Overflow caused by large translation and large world size.

b) Overflow caused by rotation and a large world size.

Figure 21 World size and movement trade-offs.

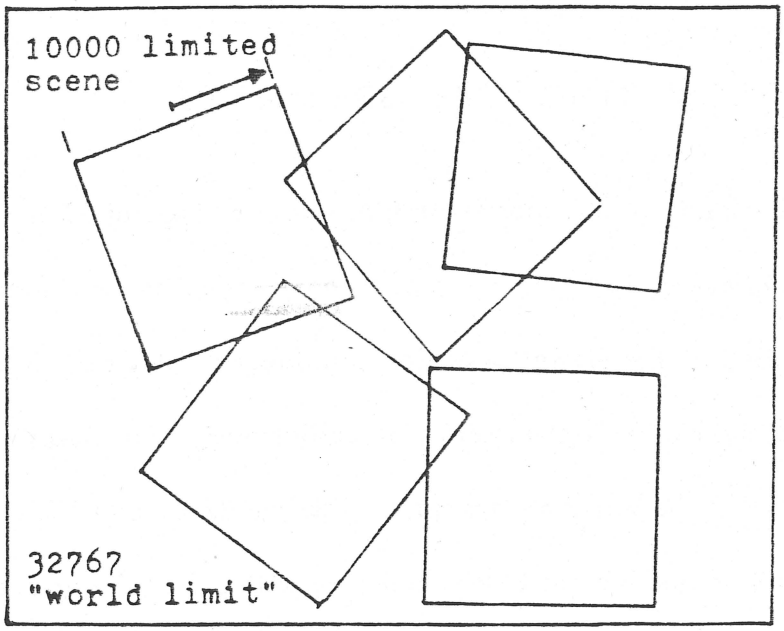


Figure 22 A reasonable movement/world size trade-off.

The simplest way to code a data base is to lay out the desired scene on a large piece of graph paper. Coordinates can then be loaded into the computer's memory in the proper format.

CONCLUSION

The A2-3D1 program was designed to be easy to use and will add a new dimension to engineering, architecture, simulation and game programs. The assembly language version is designed to be as fast-running as possible and trades-off accuracy for speed. If you are interested in super precision and don't mind slow projections, our APPLE II or general 3D Graphics BASIC version will augment this package nicely.

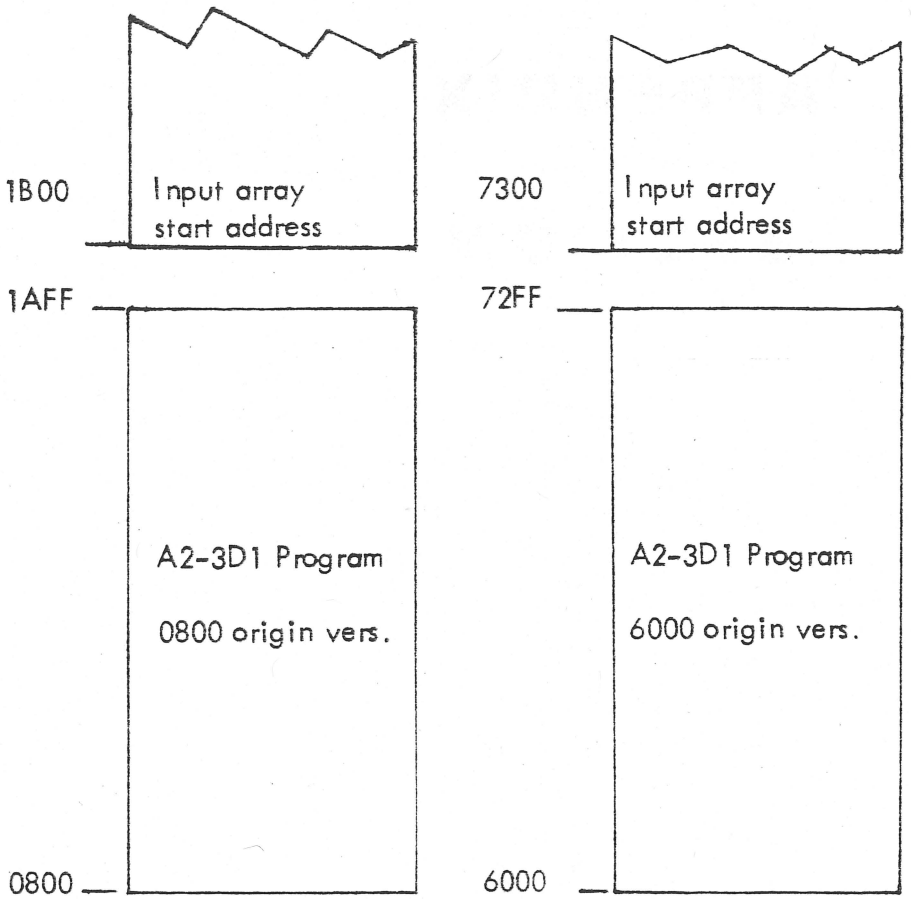
We are interested in hearing about your work in 3D graphics. We constantly try to improve, debug , and speed-up our 3D to 2D converters, and the more feedback from the field we get, the better our products will get. Once again, thank you for ordering the Sublogic A2-3D1 graphics package .

APPENDIX

SECTION

APPENDIX ONE

A2-3D1 MEMORY MAP



Entry Points (hex)

800	Normal entry
803	High Speed entry
806	Trig entry Sine
809	Trig entry Cosine
80C	Trig argument

6000	Normal entry
6003	High Speed entry
6006	Trig entry Sine
6009	Trig entry Cosine
600C	Trig argument

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

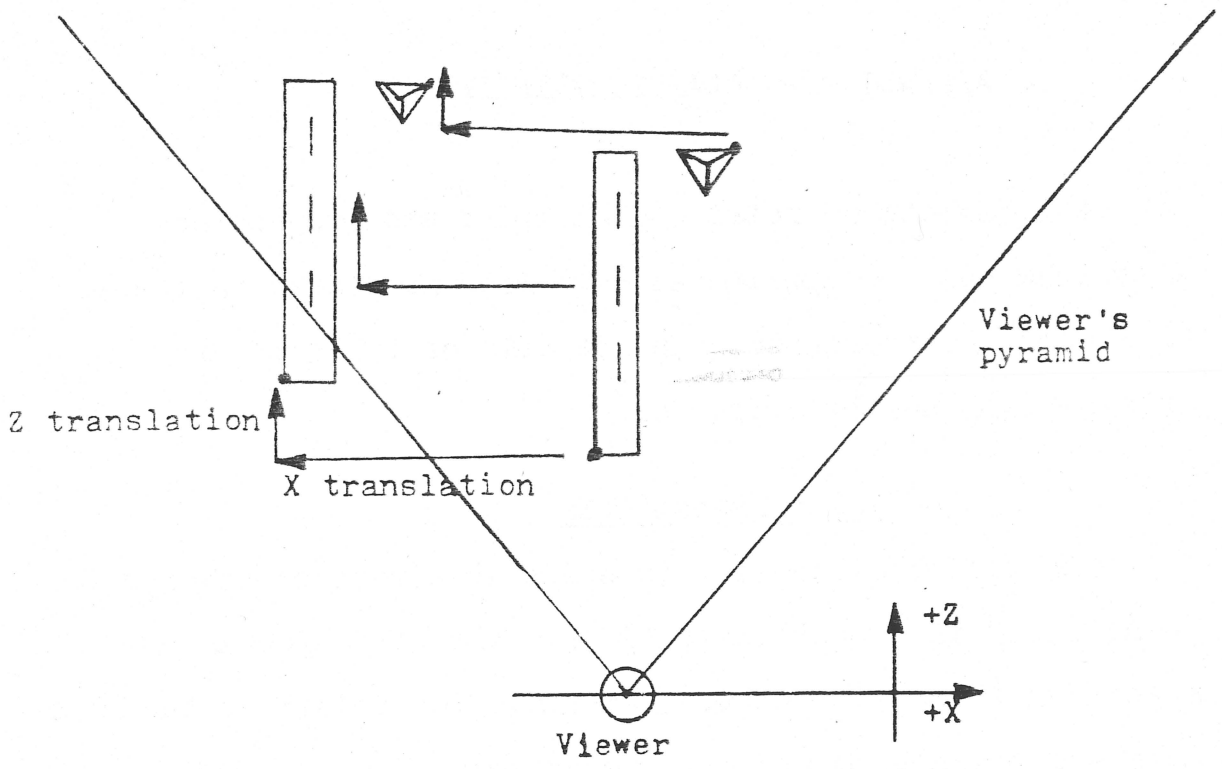


Figure A2-1. Point translation of a data base

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

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.

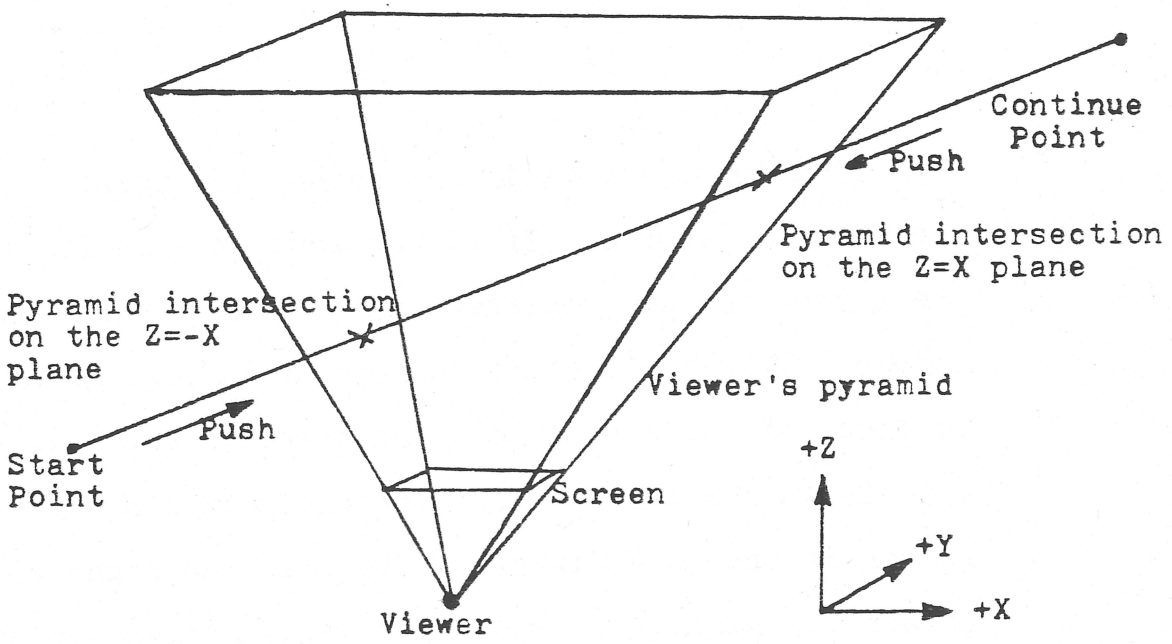


Figure A2-3. A line in need of clipping

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

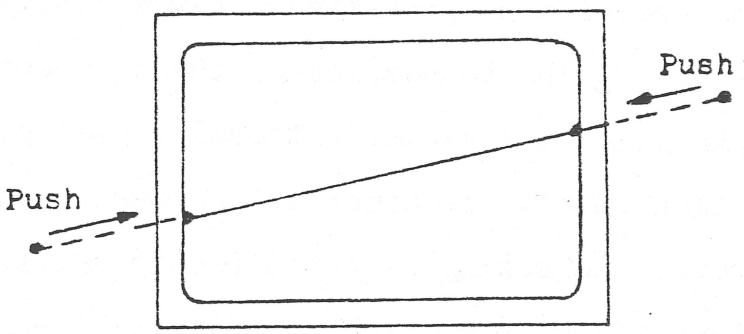
point is to the left of the screen and the end point is to the right. In this case, the codes are 1000 and 0100. The "and" of the codes is 0000 which means the line might be on the screen partially. The 1000 code indicates that the start point is to the left of the screen and must be pushed right while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

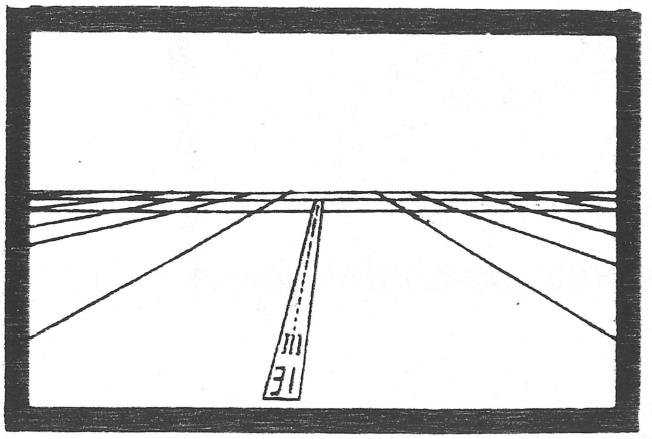
Integer arithmetic is, speedwise, far superior to floating point and was thus chosen for the assembly language 3D graphics package. Double precision 8 bit words are used for all space coordinates providing a range of 32767 units in each direction. The boundaries of the 3D scene, however, should be less 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, there is no interference from bad weather, and there is no risk of crashing.

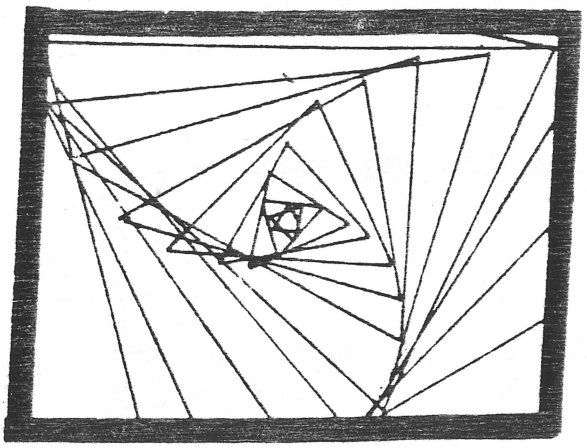
Pilots using a 3D graphics equipped simulator can learn more about the flight characteristics of an aircraft. Spins, steep dives, near crashes and generally pushing the plane to and beyond its limits are all safe maneuvers and a pilot can learn what to expect in any of these situations.



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.

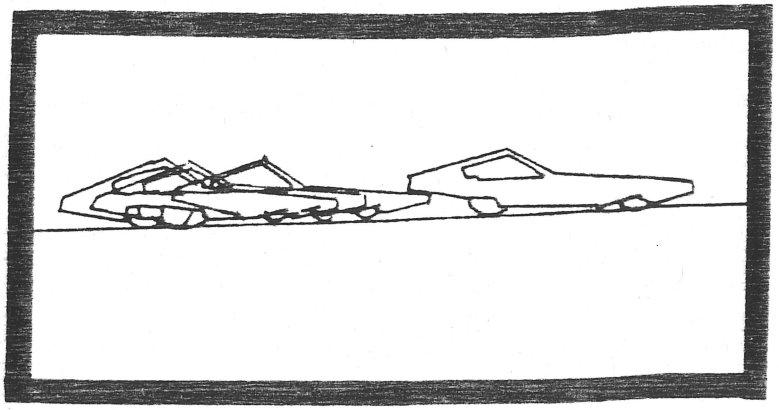
APPLICATION NOTE - - - COMPUTER ART

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:

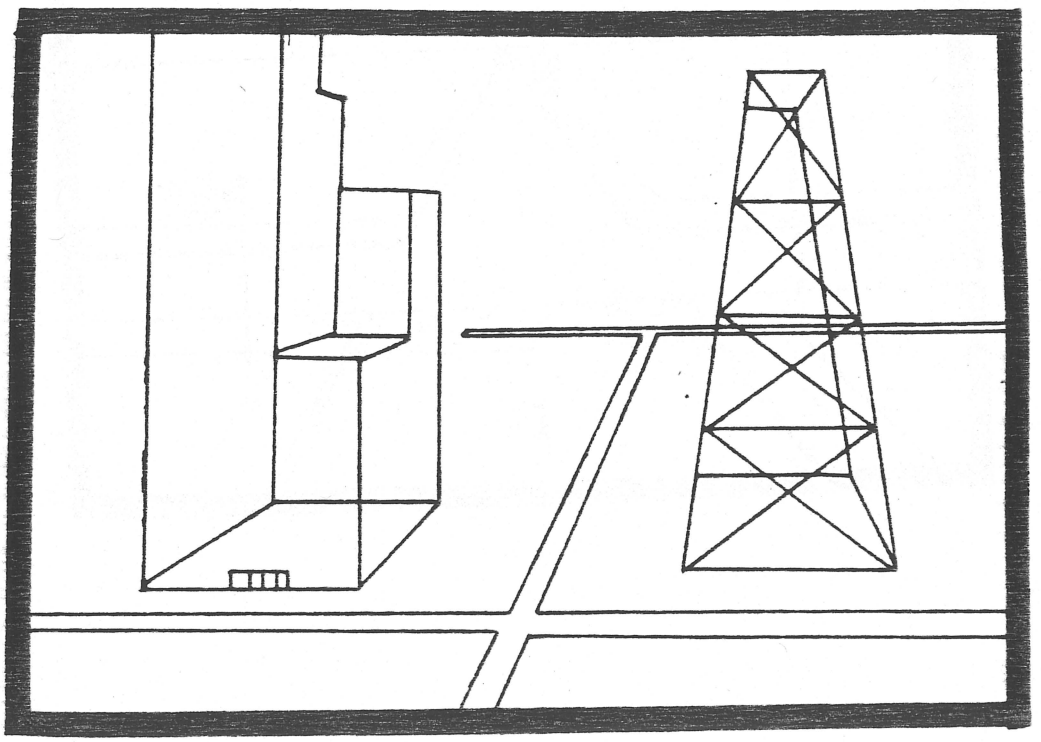
1. They are less expensive when implemented with a microcomputer based system

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.

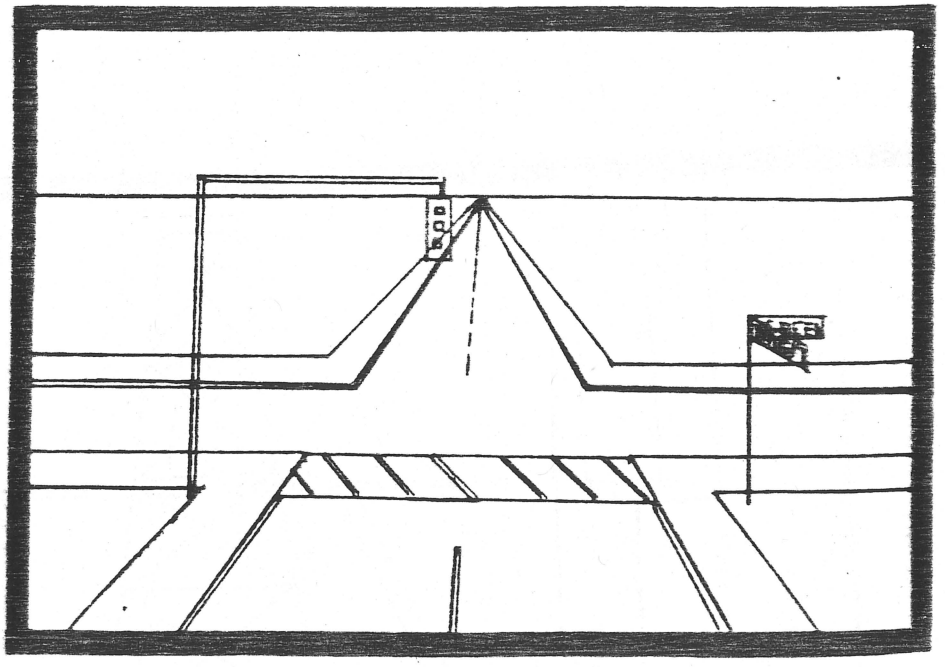


APPLICATION NOTE - - - DRIVING SIMULATION

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.

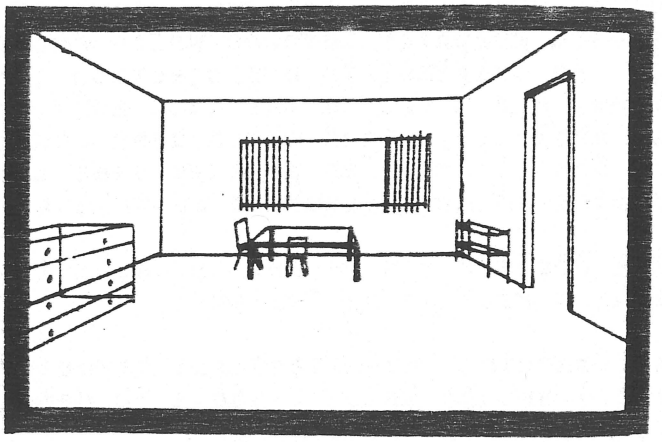
By projecting a training course on an inexpensive projection television in front of the driver, he can practice driving all day without an instructor. With todays rising gas and auto prices and dropping computer costs, this sort of simulation gets even more attractive.

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.



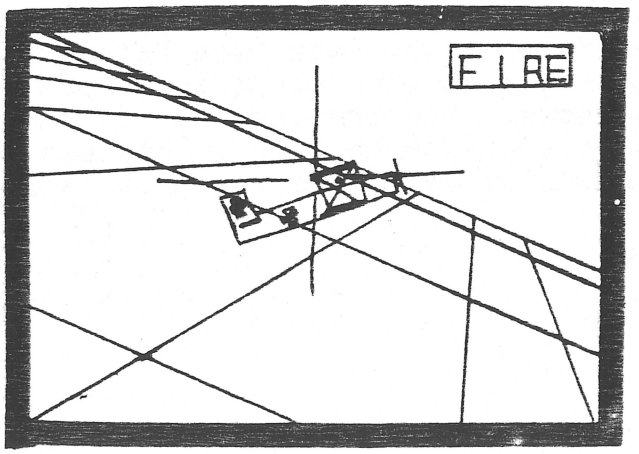
APPLICATION NOTE - - - ROOM LAYOUT VISUALIZATION

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.



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.

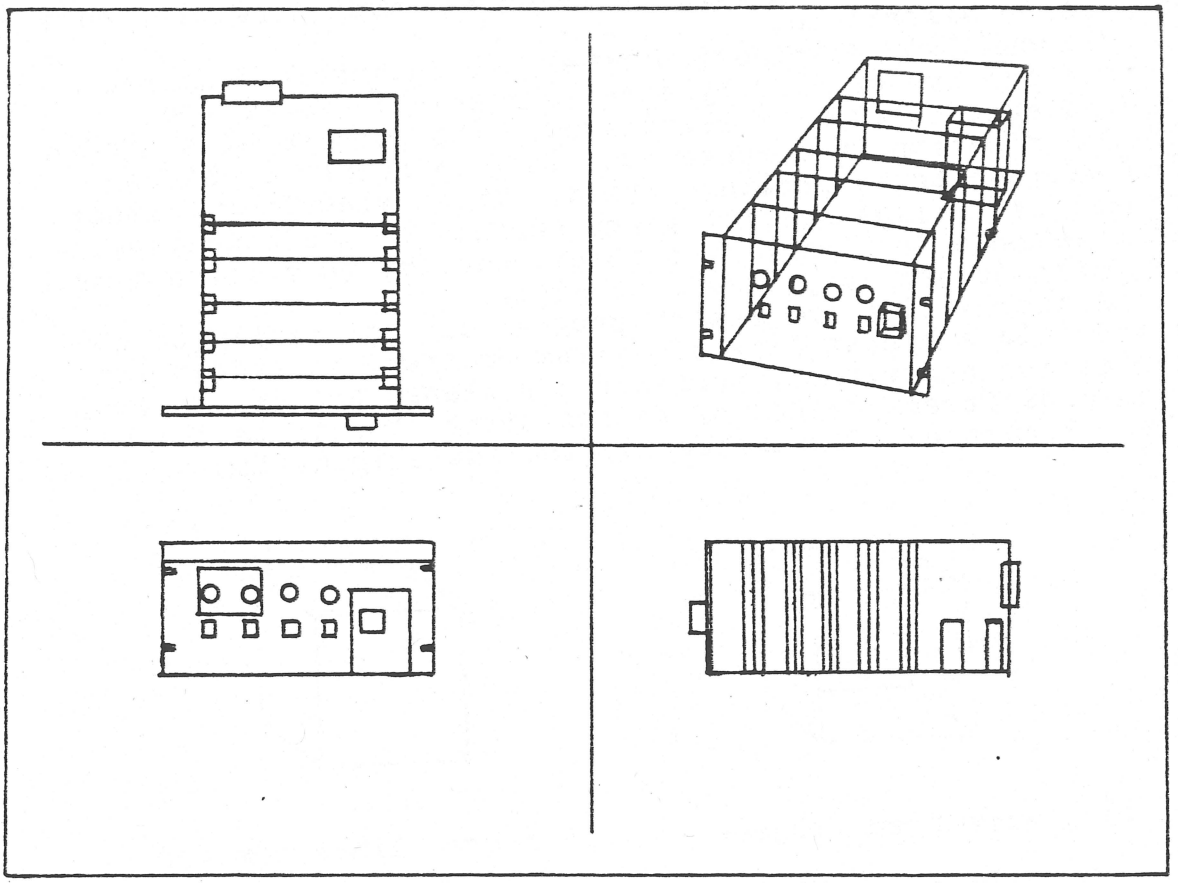


Figure a. An engineering drawing generated using 3D microcomputer graphics

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.

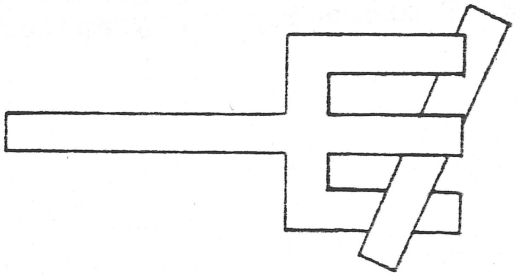


Figure b. Hidden surface conflict

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.

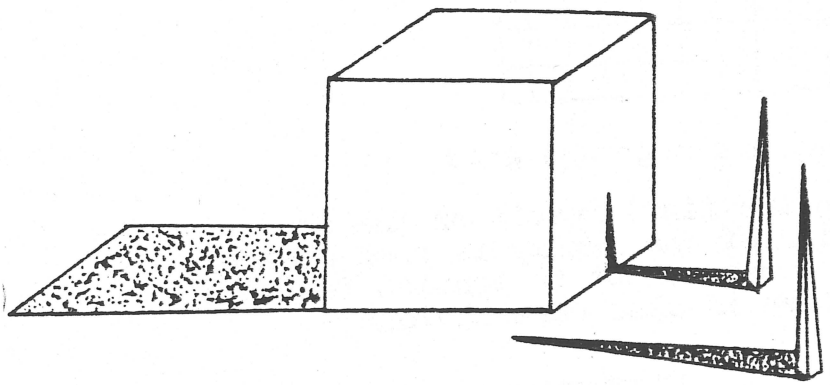


Figure c. Shadowing

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).

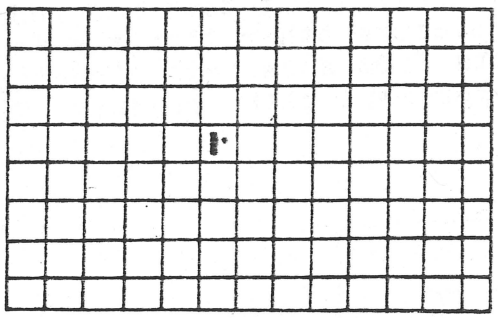
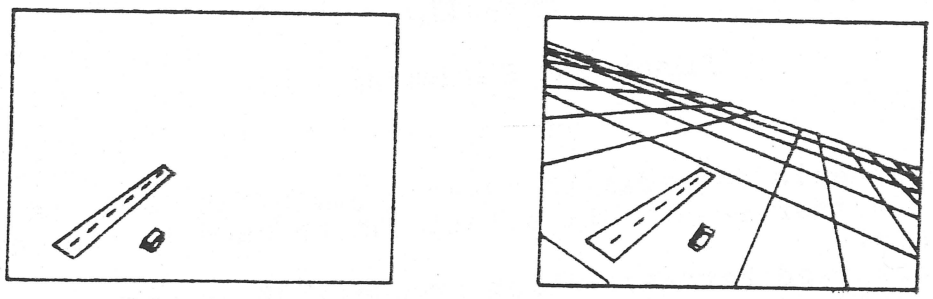


Figure a. A runway and a ground grid

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 - - - 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)

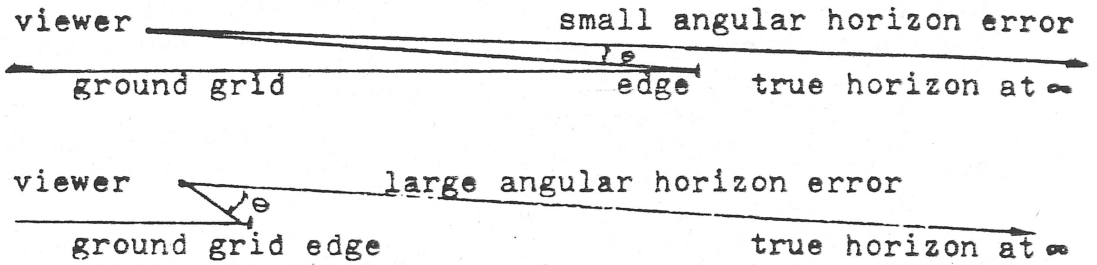


Figure a. Horizon error

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.

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)
SB= Sine (Bank)
SH= Sine (Heading)

CP= Cosine (Pitch)
CB= Cosine (Bank)
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":

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

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

APPENDIX FIVE

TAPE LOADING PROCEDURE

The A2-3D1 program comes on a standard APPLE II format tape. This tape may be read using the standard loader in the APPLE's monitor using a "R" command.

The tape loads from location 800 hex to 2FFF hex. There are actually two copies of the program that load. The 800 hex origin copy loads at its actual location, and the 6000 origin copy loads from 1C00 to 2EFF. The two areas from 1B00 to 1Bff and 2F00 to 2FFF contain the test cube data base. The 6000 origin copy must of course be moved(using the APPLE II's move command)to memory location 6000 to 73FF hex before use. After the load the memory looks like this:

2FFF 2F00	Test cube input array
2EFF 1C00	6000 hex origin A2-3D1
1BFF 1B00	Test cube input array
1AFF 0800	800 origin A2-3D1

Use this command to move the 6000 origin version to 6000 hex:

```
*6000<1C00.2FFFM
```

Use this command to initially load the tape:

```
*800.2FFFR
```

Consult the APPLE II REFERENCE MANUAL for tape recorder volume settings and loading methods for cassette tapes.

The tape contains two copies of the above-mentioned programs. The first copy is near the beginning of side one of the tape. The second copy is approximately a minute away from the first copy.

There are other programs and subroutines on the tape as well. The load and go manual for this package tells where they are and how they can be used. These are mostly BASIC programs.

APPENDIX SIX

USING THE INTERNAL TRIG FUNCTIONS

Control programs that move a viewer through space often require trig functions. BASIC trig functions are so slow, however, that one cannot seriously consider using them in a real-time simulation application. The A2-3D1 high performance trig generation routines may be used to lookup sine and cosine functions to solve this problem.

The A2-3D1's sine and cosine generator can generate a 16-bit sine or cosine in approximately 50 us. (20,000 cosines per second). This generator is table-driven.

Angles given to the sine/cosine generator must be in "pseudodegrees". A pseudodegree is defined as 1.40625 regular degrees. There are 256 pseudodegrees in a circle. The angle may therefore be expressed in one byte.

INPUT: The angle in pseudodegrees should be poked into memory location 2060 decimal (80C hex). Pseudodegrees are oriented exacly the same way as regular degrees, but there are only 256 of them instead of 360.

CALL: For sine generation the routine at 806 hex should be used. A BASIC CALL 2054 will perform a sine function.
For cosine generation, the routine at 809 hex should be used. A BASIC CALL 2057 will generate the cosine function.

OUTPUT: A 16-bit result will appear in byte-swapped order at location 80C, 80D hex in memory (2060,2061 decimal). The result is expressed in standard fractional notation. The 16-bit value will range from -32768 to 32767 thereby represent values from -1.0 to .99997. The value 25323 would therefore be equivalent to:

$$25323/32768 = \boxed{.7728}$$

Pseudodegrees are quite coarse, but very good approximations of half pseudodegrees and even quarter pseudodegrees are obtainable through linear interpolation. A circle with 512 or 1024 divisions is fine enough for almost any graphics application, and the trig generator is so fast that looking up two values wastes nearly no time at all.

APPENDIX SEVEN

PROGRAM FAMILIARIZATION

The user is referred to the A2-3D1 load and go manual for program familiarization purposes.

APPENDIX EIGHT

MULTIPLIER AND DIVIDER PATCH POINTS

The A2-3D1 graphics program is very heavily multiply and divide weighted. Significant speedups can be obtained through the use of a high performance multiplier or divider unit. The AMD 9511 or the TRW MPY 16 AJ are two chips that work well in this application. No high performance chips have been tried on the APPLE II at Sublogic Co. at this time, but if you have one on your system you may wish to give it a try.

There are two main problems in interfacing external hardware or software multipliers to the A2-3D1 package. Entry points may change as new versions are released, and each routine has many different entry points that are used in different ways. The following discussion will attempt to overcome these problems.

MULTIPLY ROUTINES: There are actually two multiplier routines in the A2-3D1 program. One is a large 16 x 16 signed twos complement multiplier, and the other is a small, fast 8x 8 multiplier. The 16 x 16 multiplier has many entries, but they all end up going to one routine called MULT. This routine multiplies MPYER by MCAND and leaves a 16-bit most significant bit result in registers X(msb) and A (lsb). The addresses are:

MCAND = 7A,7B byte swapped
MPYER = 78,79 byte swapped
RESULT = %X,%A (% signifies register)(%X=msb,%A=lsb)

The multiply should be arranged so that fractionals times integers yield correct results. In other words, 32767 x 32767 should equal approximately 32767. Likewise, 32767 x -32768 should yield approximately -32767. You may have to write a small overflow saturation routine to prevent the generation of a wrong answer when multiplying full scale negative by full scale negative numbers; mainly -32768 x -32768. This combination should saturate to 32767.

The current MULT address is 0B47, but this constantly changes. Find it by looking for the instructions A5,79,45,7B. The address with the A5 is MULT's address. MULT goes all the way to the RTS (60) about 108 bytes later. You can see exactly what MULT does by disassembling it. There should be enough room in this quarter K area for your chip's interface program.

The small multiplier is at address 10C3 (approximately). It begins with 49,FF,85,78. This routine performs %A x %X = %A where % signifies register. Again, these are fractional multiplies and 7F x 7F should equal about 7F.

In replacing either multiply routine you must be careful not to disturb any registers or zero page variables. If you do, restore them before you exit your routine.

DIVIDE ROUTINE: The divide subroutine resides at about C53 and begins with 09,00,30. This routine performs the following action:

%A,%X / MCAND = MPYER

where %A is the most significant and %X is the least significant half of a 16-bit number. MCAND and MPYER have the same addresses and byte-swapped characteristics as they did in the double precision multiplier section. Note that the significance of %X and %A is reversed in the multiply and the divide routine. In the multiply %X is most significant, while %A is most significant in the divide. The end of the divide routine is at about CE2 and is an RTS (60) instruction.

The divide handles fractionals in the same way as the multipliers, only in reverse. The following results should be obtained by your divider substitute:

32767/ 32767 = 32767 (decimal calculations)
12345/23485 = 17224
-125/589 = -6954

Basically, its like doing a division on your calculator and multiplying the fractional result by 32767.

The A2-3D1 program is setup so that the top of the fraction will never exceed the bottom. The result should therefore always be less than or equal to 32767, and will therefore never cause a 2-byte overflow problem.

IMPLEMENTATION: Its a good idea to disassemble the code at the locations to first find the subroutine, then to see how it works. Remember to leave the registers and zero page intact, and not to overwrite any routines other than the ones you are replacing. Also, it is wise to replace these routines in a step-by-step process. Don't replace them all at once, run the program, and hope it works.

We are interested in hearing about your custom applications. Performance figures are particularly desirable since we have no idea how fast one of the new high performance chips will make the 3D program run (other than our feeling that the 3D conversion process is about 80% multiply/divide weighted).

Implementing these patches on an APPLE II, using the standard Sublogic APPLE II screen driver, will probably double the projection rate at best. This is because line drawing (which requires no multiplies or divides) presently takes nearly half the image generation time.