Purpose and Overview

The graphics capabilities of GraFORTH can be divided into three main groups:

Two-Dimensional Graphics (or "Graphics of the First Kind") includes commands that plot points, draw lines, and fill rectangular areas on the screen, using a variety of colors and options.

Character Graphics (or "Graphics of the Second Kind") includes using and creating new character sets, displaying text with different sizes and colors, and defining completely new shapes and pictures in terms of character sets and displaying these shapes using a special block printing function.

Three-Dimensional Graphics (or "Graphics of the Third Kind") includes creating and displaying three-dimensional color images at high speed for animated effects.

This chapter will discuss two-dimensional graphics. We'll start by talking about what the Apple itself is capable of, and how GraFORTH uses these capabilities. We'll show you how to plot points and draw lines, and then undraw them again, effectively removing them from the screen. We'll discuss color and the drawing modes (ORMODE and EXMODE) and how they affect the drawing process. We'll also talk about using Turtlegraphics, which is especially useful for creating certain kinds of graphics displays.

Apple Graphics

The Apple screen display, whether it be text or graphics, is made out of the same units, called pixels. A pixel (abbreviated form of 'picture cell') is the smallest unit, or dot, which may be turned on or off of the surface or the screen. There are 53,760 of these smallest units which make up the entire screen, arranged in a matrix 280 dots wide and 192 dots high.

The standard Apple text display divides the screen into 24 horizontal lines, each 8 dots high. Seven of these 8 vertical dots are used to form the characters, while the eighth is used to separate the lines from one another. Horizontally, the screen is divided into 40 columns, each 7 dots wide. Five of these 7 horizontal dots are used to form the character, while one on each side of the character is used for spacing between the characters. The ASCII values for the characters on the text screen are stored in a 1024 byte memory area. The hardware inside the Apple continuously reads the values from this area and places the appropriate characters on the screen.

The Apple graphics display allows you to turn on or off all 53,760 dots on the screen individually. There are two 'graphics pages' in memory reserved for this function, but because of the higher resolution, each requires 8192 bytes to be set aside. It is possible to alternate between the pages very rapidly for animation effects (GraFORTH does this automatically for 3-D displays), but the Apple display hardware cannot merge or blend the information on the two pages. These two high resolution pages are often called 'picture buffers'. Each dot on the screen represents one bit from the picture buffer. Seven of the 8 bits in each byte are displayed on the screen, with the last bit used in determining the colors of the other dots in that byte.

GraFORTH Graphics

While it is possible to use the Apple text display from GraFORTH (with the word TEXT), the usual display is the graphics display. To specify points on the graphics screen, GraFORTH uses 'Cartesian coordinates'. This is a straightforward way to select a point by naming the column and the row the point is in. The horizontal position is the X coordinate and the vertical position is the Y coordinate.

The range of screen coordinates for GraFORTH graphics is:

X from 0 (screen left) to 255 (screen right)

Y from 0 (screen top) to 191 (screen bottom)

Thus, the upper-left corner of the screen can be represented with X=0 and Y=0, or simply the X-Y pair (0,0).

Note: The GraFORTH graphics screen is 9 percent narrower than the maximum possible (256 points wide rather than 280) for the sake of operating speed. This is one factor that contributes to GraFORTH's fast line drawing.

The standard Apple text display still uses all 280 dots across the screen for 40 characters per line. The characters themselves, instead of being placed on a text screen by the Apple hardware, are "drawn" from the text page onto the graphics picture buffer. The full character space, 7 dots by 8 dots, can be used, and is used for lower case characters and special character styles.

Two-Dimensional Graphics Words

PLOT, LINE and FILL

For these examples, we don't want text scrolling all over our beautiful graphics, so let's establish a text window in the bottom part of the screen. These examples will keep the graphics above the text window and away from harm. To establish the window, type:

```
Ready 0 40 18 24 WINDOW
```

This sets a 40-column wide window from line 18 to the bottom of the screen. Now type:

```
Ready ERASE
```

This clears the text that was still above the text window.

Let's begin at the beginning, with plotting points. The GraFORTH word PLOT removes two numbers from the stack, interprets them as X and Y coordinates, and plots a point at those coordinates on the screen. The form for PLOT is:

```
<X-coordinate> <Y-coordinate> PLOT
```

This example will plot a point in the upper left corner of the screen:

```
Ready 0 0 PLOT
```

Here is another point, in the upper right portion of the screen:

```
Ready 200 25 PLOT
```

The word LINE, like PLOT, removes two numbers from the stack and interprets them as X and Y coordinates. LINE then draws a straight line from the last plotted point to the given coordinates. To draw a line, we use the last point we plotted as one of the endpoints. We simply give LINE the coordinates of the other endpoint:

```
Ready 50 100 LINE
```

This draws a diagonal line from the point (200,25) to (50,100). We can draw another line, by using PLOT and LINE together again:

```
Ready 100 10 PLOT  100 140 LINE
```

This draws a vertical line through the other line and almost into our text window.

Rectangular areas can be filled in quickly with the word FILL. FILL also removes X and Y coordinates from the stack. It treats the last plotted point as one corner of the area, and the given coordinates as the opposite corner. This example fills in a rectangular area on the right side of the screen:

```
Ready 120 125 PLOT

Ready 200 75 FILL
```

For both LINE and FILL, the "last plotted point" is always the point last used by a plotting word, whether it was PLOT, LINE, or FILL. Another word, POSN, removes X and Y coordinates from the stack to act as a "last plotted point" without doing any plotting. POSN can be used to determine the first endpoint of a line or one corner of an area. This example uses **POSN** to set the first endpoint of a line:

```
Ready 225 50 POSN

Ready 250 125 LINE
```

COLOR

Of course, GraFORTH can draw in colors, too! The color is set with the word COLOR. COLOR removes a number from the stack and uses it to select a color. The eight color numbers (0 through 7) are the same as those used by Applesoft Basic. Here is a listing of the graphics colors:

Color Number	Color
0	not used
1	Green (1)
2	Violet (1)
3	White (1)
4	not used
5	Orange (2) (depends on monitor)
6	Blue (2) (depends on monitor)
7	White (2)

The orange and blue colors may appear different shades on different color monitors. The colors can be divided into two groups. The numbers in parentheses represent the "group number" (either 1 or 2). Because of some Apple][hardware constraints, it may be desirable to use colors from the same group when drawing lines or areas close together. We'll show you an example of this in a bit. (The Apple][Reference Manual contains more information on the internal details of these constraints.)

If you don't mind a bit of typing, this example will display 6 diagonal lines in each of the visible colors:

```
Ready ERASE

Ready 1 COLOR 0 0 PLOT 100 100 LINE

Ready 2 COLOR 20 0 PLOT 120 100 LINE

Ready 3 COLOR 40 0 PLOT 140 100 LINE

Ready 5 COLOR 60 0 PLOT 160 100 LINE

Ready 7 COLOR 100 0 PLOT 200 100 LINE
```

With your color monitorr properly adjusted, the colors of these lines (from left to rigght) should be green, violet, white, orange, blue, and anothher brand of white. Note that the colored lines are not broken att all, as they are with some graphics displays (like Applesofft). GraFORTH draws all colored lines without breaks.

Lines and points can bee drawn over FILLed areas, but the colors will be affected:

```
Ready ERASE

Ready 5 COLOR

Ready 0 0 PLOT 100 100 FILL
```

This draws an orange reectangle in the upper left portion of the screen. Now let's draww a line of a different color through it:

```
Ready 6 COLOR

Ready 0 0 PLOT 100 100 LINE
```

Note that 6 COLOR speciifies blue, but because of the orange background, the line apppears white. Now let's try the same example again, this timme using colors from different color groups:

```
Ready ERASE  5 COLOR

Ready 0 0 PLOT 100 100 FILL

Ready 1 COLOR

Ready 0 0 PLOT 100 100 LINE
```

Whoops! You should see a series of small green rectangles along the diagonal. This is the result of the Apple][hardware limitations. The solution to avoiding this trouble is to simply use colors of the same group when lines or areas are superimposed or placed close together.

UNPLOT, UNLINE, and EMPTY

So far we've been using the word ERASE to clear the graphics from the screen. In GraFORTH, points, lines, and areas can be selectively erased. Let's ERASE the entire screen now and set the color back to white, then plot a few points:

```
Ready ERASE  3 COLOR

Ready 50 25 PLOT

Ready 100 25 PLOT

Ready 150 25 PLOT
```

Points can be individually removed with the word UNPLOT. UNPLOT has the same form as PLOT, however it erases the point at the given coordinates. (If there is no point there to begin with, nothing happens.) Let's use UNPLOT to erase two of the points we have on the screen:

```
Ready 50 25 UNPLOT

Ready 100 25 UNPLOT
```

Similarly, lines can be erased with the word UNLINE. This example draws two lines, then erases one of them:

```
Ready 0 0 PLOT 100 100 LINE

Ready 50 0 PLOT 150 100 LINE

Ready 0 0 UNPLOT 100 100 UNLINE
```

Rectangular areas created with FILL can be erased with the word EMPTY. Here we'll FILL two areas, and erase one:

```
Ready 25 75 PLOT 100 125 FILL

Ready 175 25 PLOT 225 100 FILL

Ready 25 75 UNPLOT 100 125 EMPTY
```

Points, lines, and areas must be UNdrawn using the same color they were drawn in. For example, all of the above objects were drawn with 3 COLOR set. The same color was still in effect when some of the objects were erased. Let's change the color and try erasing the remaining line and area:

```
Ready 1 COLOR

Ready 50 0 UNPLOT 150 100 UNLINE
```

Since 1 COLOR is set, the GraFORTH system assumes a green line is to be erased, and leaves a string of violet dots behind.

```
Ready 2 COLOR

Ready 175 25 UNPLOT 225 100 EMPTY
```

With 2 COLOR set, GraFORTH tries to erase a violet colored area, changing the white to green.

INVERSE and NORMAL

If you prefer to do graphics on a white background, you can do this with the word INVERSE. INVERSE simply draws the 'complements' of the selected color: white becomes black, black becomes white, green becomes violet, blue becomes orange, etc. To show the effects of INVERSE, let's first erase the screen, then enter INVERSE:

```
Ready ERASE

Ready INVERSE
```

Notice that the "Ready" on the last line is now displayed in "inverse": black characters on a white background. Since only the word "Ready" was printed after executing INVERSE, it is the only thing displayed in inverse. Now type:

Ready HOME

Since HOME clears the text window, now everything inside the text window is in inverse. Now type:

Ready ERASE

ERASE has "erased" the entire screen to white. Let's draw the six colored lines again:

Ready 1 COLOR 0 0 PLOT 100 100 LINE

Ready 2 COLOR 20 0 PLOT 120 100 LINE

Ready 3 COLOR 40 0 PLOT 140 100 LINE

Ready 5 COLOR 60 0 PLOT 160 100 LINE

Ready 6 COLOR 80 0 PLOT 180 100 LINE

Ready 7 COLOR 100 0 PLOT 200 100 LINE

Note that the colors of the lines have all changed. From left to right, the colors are now violet, green, black, blue, orange, and another black.

We'll eventually want to return to a normal black-background display. The word NORMAL causes GraFORTH to use the normal colors again, including good ol' black:

Ready NORMAL

Ready ERASE

ORMODE and EXMODE

GraFORTH has two different "drawing modes", called "ORMODE" and "EXMODE". Amazingly enough, these modes are set with the GraFORTH words ORMODE and EXMODE. The 'default' mode (the mode GraFORTH uses when a mode is not specified) is ORMODE. The philosophy behind ORMODE is that the plotting words put dots of the specified color on the screen regardless of what is already on the screen. With EXMODE however, a drawing command will put points on the screen only where points are not already plotted. If some points to be plotted are already plotted, those points will instead be turned off.

A couple of examples will be helpful here. Let's first FILL an area, then draw an overlapping line in ORMODE:

Ready 100 50 POSN 150 100 FILL

Ready 50 50 POSN 200 100 LINE

The line goes straight through the middle of the rectangle. Watch what happens when we try to erase the line:

Ready 50 50 POSN 200 100 UNLINE

The line was erased, but it neatly chopped the rectangle in half, too. Using EXMODE, anything that can be done can also be undone. Let's do the same example again, this time in EXMODE:

Ready ERASE EXMODE

Ready 100 50 POSN 150 100 FILL

Ready 50 50 POSN 200 100 LINE

The line is white, except where it passes over the white background of the rectangle. Here it is changed to black. Now to erase the line, we want to make the white sections black, and the black trace through the rectangle white. And this is exactly what happens with regular plotting in EXMODE. We can erase the line by telling GraFORTH to draw it again:

Ready 50 50 POSN 200 100 LINE

The line is erased, and the rectangle is again intact. The key to understanding EXMODE is that if something is drawn once, it appears on the screen. If it is drawn again, it disappears, leaving the screen as if the object had never been drawn.

EXMODE works equally well with colors. In this example, a green line is drawn through the rectangle, the white rectangle is erased, then the line is erased:

Ready 1 COLOR 50 50 POSN 200 100 LINE

Notice that the line is violet inside the rectangle.

Ready 3 COLOR 100 50 POSN 150 100 FILL

The line is now completely green, as if the rectangle never existed.

```
Ready 1 COLOR 50 50 POSN 200 100 LINE
```

EXMODE and ORMODE can be combined with INVERSE and NORMAL along with the six colors to produce a wide variety of color and pattern combinations, more than we could hope to fully explore here. We suggest that you experiment further with these various combinations, to see how they can work best for your applications.

GPEEK

Your programs can determine whether or not a given point on the screen has been plotted with the word GPEEK. GPEEK removes X and Y coordinates from the stack, looks to those coordinates on the screen, and places a non zero number stack if the point there is "on" (not black) or a zero if the point is "off" (black). The following example draws a line, then checks two points, one on the line and one off:

```
Ready 3 COLOR 0 0 PLOT 100 100 LINE

Ready 50 50 GPEEK .
2

Ready 200 10 GPEEK.
0
```

Turtlegraphics

Turtlegraphics is also available from GraFORTH. Turtlegraphics is a somewhat different way of specifying how to draw lines in GraFORTH. Imagine a tiny turtle sitting on the middle of the screen with ink on his tail. Wherever he moves he draws a line behind him. We can tell him to turn to the left or the right, and we can tell him to walk forward a given distance leaving a straight line behind him. (For the mathematicians among us, this way of drawing lines could be considered as using "relative polar coordinates".)

The Turtlegraphics words in GraFORTH are found on the system disk in a text file called "TURTLE". We can compile these words into the dictionary by typing:

```
Ready READ " TURTLE "
```

We can see the words added to the dictionary by typing LIST. A few of the words are used by the other words: TURTLE.X, TURTLE.Y, and TURTLE.ANG are variables, and TURTLE.WALK is called by both MOVE and MOVETO.

Let's "initialize" Turtlegraphics by typing:

```
Ready TURTLE
```

TURTLE resets graphics mode, erases the screen and sets a text window along the bottom four lines, then sets 3 COLOR (white) and positions the turtle in the center of the screen, facing toward the top.

MOVE

The word MOVE moves the turtle in the direction it is pointing, drawing a line. The form is:

```
<distance> MOVE
```

The distance is measured in pixels, or dots. To move the turtle 50 pixels, type:

```
Ready 50 MOVE
```

TURNTO

The turtle can be turned to a certain angle with TURNTO. TURNTO has the form:

```
<angle> TURNTO
```

The angle given is in degrees, and increasing angles are in a clockwise direction. Zero is straight up, 90 is to the right, 180 is facing down, and 270 is to the left. Let's move the turtle in our example to face to the right (to 90 degrees), then move it 75 pixels:

```
Ready 90 TURNTO

Ready 75 MOVE
```

TURN

The word TURN turns the turtle clockwise from its current direction a given angle. The form is the same as for TURNTO, but TURN is a relative turn from the turtle's current direction. The following example now turns the turtle 45 more degrees clockwise, then moves the turtle 50 pixels:

```
Ready 45 TURN

Ready 50 MOVE
```

MOVETO

Lastly, MOVETO moves the turtle directly to a specified X,Y position on the screen without drawing any line. The form for MOVETO is:

```
<X coordinate>  <Y coordinate>  MOVETO
```

MOVETO is similar to POSN in that it simply establishes a new point on the screen, but MOVETO also updates the turtle's position for further Turtlegraphics commands. We can move the turtle to the upper-left corner of the screen, turn it to face to the lower-right, then move it back to the center, drawing a line, with the following commands:

```
Ready 0 0 MOVETO

Ready 127 TURNTO

Ready 160 MOVE
```

Examples

The advantage of Turtlegraphics is that shapes can be drawn in different sizes and facing different directions with little work. For example, to draw a square, you can type the following:

```
Ready TURTLE

Ready 50 MOVE 90 TURN 50 MOVE 90 TURN

Ready 50 MOVE 90 TURN 50 MOVE
```

A faster way is to repeat the words in a loop:

```
Ready TURTLE

Ready 4 0 DO 50 MOVE 90 TURN LOOP
```

This line can be put into a word definition and used at any time:

```
: SQUARE
  4 0 DO
      50 MOVE
      90 TURN
  LOOP ;
```

Now the square can be drawn starting at any point on the screen and turned any direction:

```
Ready TURTLE

Ready 0 100 MOVETO SQUARE

Ready 55 100 MOVETO 30 TURNTO SQUARE

Ready 120 100 MOVETO 60 TURNTO SQUARE

Ready 190 100 MOVETO 90 TURNTO SQUARE
```

(Note: The GraFORTH word SIN is used to compute sines of angles used in Turtlegraphics. If you have an applications program that uses angles, the word SIN can be very helpful. SIN removes a number from the stack and uses it to select and return a scaled sine value. The table repeats for every 128 numbers, and returned values range from -128 to 127.)

CHAPTER SEVEN: CHARACTER GRAPHICS

Chapter Table of Contents: Page

Purpose and Overview

GraFORTH can do weird and wonderful things with the characters displayed on the screen. Text can be reverse scrolled, down the screen. Characters can be made much larger, and displayed in color. Different character styles, or 'fonts' can be selected and even created in GraFORTH. Entire images can be defined within a character font and rapidly printed as a block of "characters" for animated displays.

In this chapter we'll show you how to make use of each of these features and give you some suggestions for incorporating them into your own programs.

Special Output Characters

Besides the special input characters (ConTRoL-I, ConTRoL-O, etc.) discussed in Chapter 4, GraFORTH also uses two special output characters, ConTRoL-L, and ConTRoL-K. These characters are usually printed from within a program, instead of entered at the keyboard. (They can be typed from the keyboard, but GraFORTH will try to read them as characters in a GraFORTH word.)

ConTRoL-L (Apple ASCII number 140) erases the screen inside the text window. Printing a ConTRoL-L is equivalent to executing the word HOME.

ConTRoL-K (Apple ASCII number 139) causes a reverse line feed, so that subsequent printing will be one line higher. If printing is already on the top line of the text window (the vertical tab equals the top window margin), then the display will scroll in reverse, moving text down the screen.

Changing Character Size and Color

GraFORTH has the unique ability to print characters in 8 different sizes using the word CHRSIZE. CHRSIZE removes a number from the stack to select the character size. Valid numbers are from 0 to 8. Character size 0 specifies the usual GraFORTH character display. Character sizes 1 through 8 cause the characters to be "drawn" onto the screen using GraFORTH's color graphics capabilities. Character size 1 is the same size as character size 0, and the others are 2 through 8 times larger.

Let's introduce some of these features through examples. First, we'll set everything back to normal by typing:

```
Ready ABORT
```

Now let's erase the normal sized characters from the screen and select a larger character size:

```
Ready HOME 2 CHRSIZE
```

(Erasing the screen with HOME is a normal but not required step in changing character size. If HOME is not used before changing size, in some cases not all entered characters will be printed.)

The "Ready" prompt is now twice its normal size! You will notice that the large character sizes take a longer time to print, and that if allowed, scrolling is much slower than it is when using the standard character size. Also, the screen is actually 9% narrower than the standard size, since the graphics features are used to print them.

The large characters can also be displayed in color! Type:

```
Ready HOME 1 COLOR
```

This will clear the screen, then make the text green. We cleared the screen again because combining two colors of text on the screen can have some unusual effects of its own. To see these effects, type:

```
Ready 2 COLOR
```

Now hit the <return> key a few times to cause the text to scroll. The "Ready" prompt that was green gets overwritten with the violet, but does not scroll. Only text of the current color and of the current size will behave as expected with text commands.

Obviously, when the characters are larger, fewer characters can be displayed on the screen. When you select a new character size with CHRSIZE, GraFORTH automatically sets the text window size to the correct limits, to keep the text on the screen. Below is a table relating character sizes to the number of characters that can be displayed, and indicating whether or not colored text is possible for that character size:

Size	Columns	Rows	Color?
0	40	24	No
1	32	24	Yes (with funny effects)
2	16	12	Yes (with better effects)
3	10	8	Yes
4	8	6	Yes
5	6	4	Yes
6	5	4	Yes
7	4	3	Yes
8	4	3	Yes

You might want to try the following to see GraFORTH's largest character size in color. First type ABORT to get yourself back to a predictable place, then type:

```
Ready HOME  8 CHRSIZE  5 COLOR
```

A mammoth orange "Ready" prompt will appear, split across two lines, with a huge lumbering cursor! Allowing time for the text to scroll, now enter:

```
Ready INVERSE
```

After another scroll, the display changes to inverse. Obviously, you wouldn't want to enter a long program this way! Large character sizes work very well for program or game displays, but weren't really intended to be used for input. The fastest way out of our current situation (besides hitting <reset>) is to type:

```
Ready ABORT
```

After the text scrolls once more, the ABORT is executed, and things are back to normal.

Font Selection

The character "style" used in a text display (the actual set of shapes of the characters displayed) is called a character 'font', or character set. The Apple][contains an uppercase-only character set stored in its hardware. GraFORTH uses this when TEXT mode is selected. However, GraFORTH's usual graphics display instead uses a character set from memory. This character set is stored in a binary file on the GraFORTH system diskette, and is read into memory when GraFORTH is first booted.

The disk actually contains several character sets, and any of them can be used for text display. The character set files on disk are:

```
CHR.SYS
CHR.STOP
CHR.SLANT
CHR.GOTHIC
CHR.BYTE
CHR.STUFF
CHR.MAXWELL
```

(The last two are special character sets used for 'character graphics', and do not work well for a text display. We'll show you how to work with these in a bit...)

In memory, a character set occupies 768 bytes. There are 96 printable characters, and each character uses 8 bytes in the character set. These 8-byte blocks are actually graphics "pictures" of each character. When GraFORTH is booted, it loads CHR.SYS into memory starting at location 2048. Whenever it displays a character, it looks up the "picture" of that character from this area of memory, and places it on the screen.

Character sets elsewhere in memory can also be used for the screen display. Let's load another character set from disk into a free area of memory. The location 2816 is the beginning of a large free area of memory. We'll use a standard DOS call to load the file in:

```
Ready CR 132 PUTC PRINT " BLOAD CHR.BYTE,A2816 " CR
```

The disk whirs a bit, and the character set is loaded. To use this character set for the display, the word CHRADR is used. CHRADR stands for CHARacter ADdRess, and it is used to select the memory location of the current character set. The form is:

```
<address of character set> CHRADR
```

We loaded the character set into memory starting at location 2816, so this is the address we give to CHRADR:

```
Ready 2816 CHRADR
```

All printing will now use the new character set. The characters that were already on the screen in the old character set, however, are unchanged. Characters from different character sets can be displayed on the screen at the same time. However, if the screen is scrolled, these characters will be reprinted a line higher, using the newest character set.

The ASCII numbers for the printing characters range from 160 to 255. To display all of the printing characters in the set at once using PUTC, type:

```
Ready 255 160 DO I PUTC LOOP
```

You may want to load the other character sets into memory to see what they look like. You can load them into the same area of memory and overwrite CHR.BYTE, or you can use another free area of memory and select it with CHRADR. The memory map in Appendix B shows the free areas of memory. Therefore, it is possible (and easy!) to have several character sets in memory at once, quickly changing from one to another. Care should be taken, however, to avoid overwriting a portion of the GraFORTH system. Remember that each character set occupies 768 bytes of memory.

Usually, you will want to return to the system (CHR.SYS) character set. The GraFORTH word CHRSET returns the address of this character set, 2048. Thus, to switch back to this display, you can type:

```
Ready CHRSET CHRADR
```

(Of course if you want to, you can overwrite this area of memory with another character set, too.)

The CHAREDITOR

On the GraFORTH system diskette is a file called CHAREDITOR. This program enables you to read in character sets, examine and modify character shapes, create large block images that are stored as a series of characters, and save the new character sets to disk again.

CHAREDITOR is one of the larger programs, so it would be a good idea to LIST the dictionary and FORGET any words you may have added before loading in CHAREDITOR. To load the program in, type:

```
Ready READ " CHAREDITOR "
```

To run CHAREDITOR, type:

```
Ready HOME RUN
```

Notice that we cleared the screen before running the program. CHAREDITOR does not automatically clear the screen. This is so that any graphics images on the screen can be retained and used within the CHAREDITOR, allowing you to "pull" images and shapes from other programs into your GraFORTH character sets.

You will see a list of commands to the right, the prompt "Enter command:" near the bottom of the screen, and a flashing dot in the upper-left corner. This flashing dot is the "drawing cursor" and will be used for creating your own character shapes.

Selecting and Displaying the Character Set

The character editor works with one character set at a time. To get an understanding of things, let's start by looking at the system character set that starts at location 2048. The editor uses single-letter commands. To specify the address of the desired character set, press "A" for Address. You will then see the prompt:

```
Enter Charset
Work Area Address : 2816
```

The input cursor is flashing over the "2816". This is the default address, the address used if you do not specify one. You can keep this address simply by pressing <return>. However, we want to enter the address of GraFORTH's standard character set. Type "2048" over the top of the "2816" and press <return>. Now 2048 is the address of the character set used by the character editor.

Type "D" for "Display characters". You'll see a display across the bottom of the screen of all the characters in the character set, in inverse. To the left are the numbers 0, 32, and 64. These are index numbers. When manipulating character shapes in GraFORTH, character numbers in the range of 0 to 95 are used instead of the ASCII values (which range from 160 to 255 for printing characters). The first row of characters are numbered 0 through 31, the second row 32 through 63, and the third row 64 through 95.

Displaying a Block of Characters

If we want, we can take a sequential string of characters and display them in a rectangular block on the screen. Let's display the 6 characters "n" through "s" in a block that is 3 characters wide by 2 characters tall. To select a block of this size, press "B" for "Blocksize". You will be prompted:

```
Enter Block Horizontal Size :
```

Enter a 3 and press <return>. You will see:

```
Enter Block Vertical Size :
```

Enter a 2, press <return>, and you will get the regular "Enter command:" prompt back. Also notice that 4 more dots have appeared at the top of the screen, outlining our 3 by 2 character block.

Press "D" to bring the character set display back. Counting across the bottom row from the index number 64, you will find that the character "n" is character number 78. To display the block of 6 characters starting with "n", type "R" for "Read".

You will see:

```
Enter character number
to be read :
```

We want character number 78, so type "78". The 6 characters will appear in the block surrounded by the 4 dots.

You can also display blocks starting on other haracters, or use a different blocksize. When changing blocksiz, you may want to erase the block from the screen. To do this, imply type "E" for "Erase", then answer "Erase (Y/N) :" with a "Y".

We've been looking at a block of standard charcters, to show you how block printing is done. Now let's see som actual character graphics. To protect our precious system charcter set, press "A" and select an address of 2816 again, back nto open memory. Type "G" for "Get". This option allows you toload a character set in from disk. You will see:

```
Enter Load File Name :
```

Type "CHR.STUFF". This character set will loa into memory starting at the location 2816. Type "D" to diplay this character set. Except for a few punctuation smbols, those don't look much like characters! You can see piecesof the Insoft logo, parts of faces, and an assortment of lins which are actually pieces of a helicopter used in the GrFORTH demonstration program.

If you've changed the Blocksize, set it back t 3 characters horizontally by 2 characters vertically. Now ype "R" and read character number 78. A smiling face will apper in the upper left. By pressing "D" again, you can see thatthis face occupies the same six characters that the characters "n' through "s" occupied in the system character set. The othr three faces begin at character numbers 84 and 90. Just prss "R" and enter the character number to see them.

The Insoft logo uses a blocksize of 8 by 2 chaacters, and begins at character number 16. The three helicoptersuse a blocksize of 5 by 3 characters and begin at character numbes 33, 48, and 63. You will probably want to erase the block (witl "E") before changing the blocksize, so that part of the prvious image won't remain on the screen beside the new block.

Defining Your Own Shapes

To create your own shapes with the character editor, first select a blocksize for the image you want to draw. Erase the block if necessary. Here's where the drawing cursor comes in. By pressing the I, J, K, and M keys, you can move this cursor one pixel up, left, right, or down within the block. If you want to plot a point at the position of the cursor, press "P" for "Plot". To draw a line from the last plotted point to the cursor, press "L". Notice that "P" and "L" are actually PLOT and LINE commands, with the coordinates specified by the cursor. The character image is created by moving the cursor and drawing the points and lines that make up the image.

In addition, you can create character images in color. Press "C" for "Color" and enter the number of the color you want to work in. (When colored character images are displayed in GraFORTH, the colors may be different, depending on whether the image is drawn beginning on an odd-numbered column or an even-numbered column. This comes about as a result of the way the Apple][generates high-resolution color.)

If you plot a point that you didn't want, you can erase it by pressing "U",which UNPLOTs the point. Similarly, you can erase lines by pressing "Z". If the drawing cursor moves too slowly, you can increase its step size by pressing "X", then entering the number of pixels you want the cursor to move whenever you press a cursor-moving key (I, J, K, M). If your image isn't coming out the way you'd like....well, press "E" to erase it and try again!

Experiment with these keys to get a feel for creating images. All of the images in CHR.STUFF were created with the character editor. If you like, you can read an existing image from the character set and use the drawing keys to modify it.

When you've created an image that you want to save, first multiply the block vertical size by the horizontal size, to determine how many characters your image will occupy. Then press "D" to see the current character set, and choose a range of characters in the character set to write your image to. Press "W" for "Write". You will be prompted:

```
Enter character number
to be written :
```

Type the character number of the first character in the desired range. Your image will be written into the character set starting at that character. Press "D" again and you will see your image neatly dissected and placed in the character set.

Images from one character set can be copied to another using the CHAREDITOR "T" ("Transfer") option. You will be prompted for a "From" address, a "To" address, and a length. To copy an entire character set from one address to another, simply enter the address of the character set to be transferred, the address of where it is to go, and enter 768 for the length. Remember that character sets are 768 bytes long.

Transferring only part of a character set is a little trickier. Remember that each character occupies 8 bytes. Compute the "From" and "To" addresses based on the character number and the addresses of the character sets. The length is the number of characters times 8.

Saving a Character Set

After a new character set has been created, you can save it to disk to be used again later. To save a character set, press "S" for "Save". You will see:

```
Enter Save File Name :
```

Type the filename you've selected for the character set. Be sure that there are no files with that name on disk, unless you want to overwrite that file. Note that all of the character sets on the GraFORTH system disk begin with the prefix "CHR.". This is not a requirement; the prefix simply acts as a reminder that the file contains a character set.

When you want to leave the character editor, type "Q" for "Quit". If you want to begin work with another program, it would probably be best to FORGET the character editor first, since it takes up a lot of room in the word library. The word "X" is the first word in the character editor, so to delete the editor, type:

```
Ready FORGET X
```

Block Printing from GraFORTH

Printing blocks of characters is done directly from GraFORTH much the same way as in the character editor. A character set is loaded into memory, an appropriate blocksize is selected, and a sequential range of characters is printed in the block at the current horizontal and vertical position.

Let's display some of the same images we saw earlier in the character editor. First, load "CHR.STUFF" back into memory:

```
Ready CR 132 PUTC PRINT " BLOAD CHR.STUFF,A2816 " CR
```

You could now type "2816 CHRADR" to select the character set, but remember that this character set doesn't have much in the way of recognizable characters! It contains helicopter parts and other things. GraFORTH can recognize the characters fine, but the screen display is unusable. When we display a character image, we'll jump into the character set, display the image, then jump back out.

BLKSIZE

The block size in GraFORTH is set with the word BLKSIZE. The form for BLKSIZE is:

```
<horizontal size> <vertical size> BLKSIZE
```

As in the character editor, the horizontal and vertical size are measured in characters. BLKSIZE remains set until changed. The word ABORT does not reset BLKSIZE.

To prepare to see the smiling faces, set a blocksize of 3 characters wide by 2 characters tall:

```
Ready 3 2 BLKSIZE
```

PUTBLK

The word that actually puts the block of characters on the screen is PUTBLK. PUTBLK removes a number frrom the stack and uses it as the starting character number for the block to be displayed. Character numbers range from 0 to 95, as in the editor. The number of characters to be printed is determined by BLKSIZE. The position of the block on the screen iss set the same way text is positioned, with HTAB and VTAB, or thee other text positioning commands.

Let's block-print one of the faces in CHR.STUFF. For this example, type this entire line at oncee:

```
Ready HOME  2816 CHRADR  78 PUTBLK  CHHRSET CHRADR  12 VTAB
```

"HOME" clears the screen and positionss printing to the upper-left corner, "2816 CHRADR" sets the characcter set address for CHR.STUFF, "78 PUTBLK" actually printss the image, "CHRSET CHRADR" resets the system character set, and ""12 VTAB" gets the following "Ready" prompt down out of the way, sco that it won't overwrite the block just printed.

A smiling face should have appeared iin the upper-left corner of the screen.

To save on typing a bit, let's definee a couple of new words to help us in and out of the special charracter set. We'll call these words "IN" and "OUT":

```
Ready : IN  2816 CHRADR  HOME ;

Ready : OUT  CHRSET CHRADR  12 VTAB ;;
```

To display another face, we can simplly type:

```
Ready IN 84 PUTBLK OUT
```

Unlike text printing, PUTBLK does nott update the horizontal cursor position. Therefore, once a pprinting position has been established, several images can be drrawn sequentially in the same space. The following example prints the three helicopter images in the same space 100 times. Keep yoour eyes open; it's fast:

```
Ready 5 3 BLKSIZE

Ready IN 100 0 DO 33 PUTBLK 48 PUTBLKK 63 PUTBLK LOOP OUT
```

After changing the blocksize, the Insoft logo (which starts at character number 16) can be displayed centered on the screen:

```
Ready 8 2 BLKSIZE

Ready IN 5 VTAB 16 HTAB 16 PUTBLK OUT
```

We're being cautious about the display here because we're mixing the printing of block images using one character set with reading keyboard input using another. Most finished programs will have the changes planned out, so that the most effective mixing of character images and text display can occur.

To erase a character image, the word UNBLK is used. UNBLK simply erases a block in the current blocksize at the current printing position. The following example erases the Insoft logo we placed on the screen:

```
Ready 5 VTAB 16 HTAB UNBLK
```

The VTAB and HTAB determine the position of the block to be erased. Since UNBLK doesn't print any characters, we don't need to specify a character set.

Of course, character images can also be made larger by using CHRSIZE. This example displays the Insoft logo four times as large:

```
Ready IN 3 CHRSIZE 1 COLOR 16 PUTBLK 0 CHRSIZE OUT
```

EXMODE Character Graphics

Character sizes 1 through 8 will be drawn in "EXMODE" if EXMODE is set. This allows you to draw characters or character images over other graphics, then erase them, leaving the original graphics intact. However, EXMODE character graphics requires a few special considerations.

As GraFORTH displays characters on the graphics screen, it stores the ASCII values for those characters in the text screen area. If a character about to be printed is already in place on the screen, no high-resolution printing is done, since the character is already present. This saves much time in printing and scrolling.

However, when using EXMODE, you usually want to reprint the same characters in the same location to cause them to disappear again. Therefore, to unprint a line using EXMODE, you must first erase the text screen (this is the actual Apple][text screen, not the high resolution screen used by GraFORTH) to force a reprinting. To do this, you use the Apple][monitor's screen erase routine ("-936 CALL"), then print the same line in the same position. The following word definition is an example of using EXMODE character graphics. It draws a diagonal line, writes text over the line, then erases the text, leaving the line intact. It repeats this 4 times:

```
: EXMODE.DEMO
  ERASE
  1 CHRSIZE                 (Set up EXMODE character graphics)
  EXMODE
  0 0 PLOT 100 100 LINE     (Draw the line to be written over)
  4 0 DO                    (Loop 4 times)
      3000 0 DO LOOP        (Delay loop, to slow it down)
      5 VTAB
      5 0 DO                (Print the line 5 times)
          PRINT " This line can be erased " CR
      LOOP
     -936 CALL              (Erase the text screen)
  LOOP
  0 CHRSIZE ;
```

Summary

Output Characters

GraFORTH uses two special output characters: ConTRoL-L erases the screen inside the text window, and ConTRoL-K causes a reverse line feed, making the screen reverse scroll if the cursor is at the top of the text window.

Character Sizes

The GraFORTH word CHRSIZE uses a number from the stack to select a character size. Valid numbers are 0 through 8. Sizes 1 through 8 can be drawn in color using the word COLOR. Character size 0 is the normal text display.

Font Selection

Various character fonts can be used by BLOADing them into free memory and selecting that memory with CHRADR. GraFORTH's system character set begins at location 2048. The word CHRSET returns this address.

CHAREDITOR

The program CHAREDITOR is used to modify and save character shapes and images. Here is the normal sequence of events in the use of CHAREDITOR, with example entries:

1. Load and run the CHAREDITOR program:

```
Ready READ " CHAREDITOR "

Ready HOME RUN
```

2. Select a character set work address:

```
Enter Charset
Work Area Address : 2816
```

3. (optional) Load a character set:

```
Enter Load File Name : CHR.STUFF
```

4. Select a block size (single characters are always 1 by 1; images may be larger):

```
Enter Block Horizontal Size : 3
Enter Block Vertical Size : 2
```

5. Draw the image or character using the described sketching keys.

6. Write your image or character into the character set:

```
Enter Character Number to be Written : 90
```

7. Save the modified character set to disk:

```
Enter Save File Name : CHR.TEST
```

Block Printing from GraFORTH

Displaying character graphics from GraFORTH usually involves the following steps:

1. Load a character set into memory:

```
Ready CR 132 PUTC PRINT " BLOAD CHR.STUFF,A2816 " CR
```

2. Select the character set:

```
Ready 2816 CHRADR
```

3. Choose an appropriate blocksize:

```
Ready 3 2 BLKSIZE
```

4. (optional) Select a character size and color:

```
Ready 2 CHRSIZE  1 COLOR
```

5. Position the cursor and draw the block:

```
Ready 5 VTAB 2 HTAB 90 PUTBLK
```

Since PUTBLK does not advance the cursor, several blocks may be drawn on top of one another without having to reposition the cursor. The word UNBLK erases a block at the current position of the given blocksize.

EXMODE Character Graphics

Character sizes 1 through 8 may be drawn using EXMODE. This way, characters can be displayed over other graphics without erasing them. However, to erase a line printed in EXMODE, the text screen must first be erased with "-936 CALL" before the line is reprinted.

Conclusion

This chapter introduced GraFORTH's character graphics capabilities. So far we have covered the language features of GraFORTH, its point and line graphics, and now the set of graphics that manipulate characters and block images. Next chapter, we'll introduce the most amazing aspect of GraFORTH, its three dimensional color graphics capability. So hold on to your keyboard, here we go!

CHAPTER EIGHT: 3-D GRAPHICS

Chapter Table of Contents:

Purpose and Overview

Perhaps the most exciting aspect of GraFORTH is its high-speed 3-D graphics capabilities. GraFORTH can manipulate up to 16 three-dimensional shapes simultaneously. In this chapter we'll discuss how to use these features.

We'll begin with an overview of how 3-dimensional shapes are accessed and manipulated, and give you some introductory examples. We'll then explain the various 3-D parameters and discuss the image "format" in detail. We'll show you how to use the IMAGEDITOR to create your own 3-D images, then discuss 3-D display methods. Lastly, we'll discuss two very useful programs for developing and manipulating your 3-D image files.

3-D Graphics at a Glance

To display a 3-D object in GraFORTH, the "image" information describing the shape of the object is first loaded into a free area of memory, then commands are entered which tell the GraFORTH system where the image is in memory, and how the image is to be displayed.

GraFORTH uses an internal array to store the current information about all of the 3-D objects being displayed. The array stores the locations in memory of the actual images and the display parameters (position, rotation, size, etc.). A number (from 0 to 15) is used to refer to each object, and to select which object is currently being manipulated.

To view a 3-D image, let's first make sure things are back to normal:

```
Ready ABORT
```

and set a text window so that text doesn't scroll over our 3-D images:

```
Ready 0 40 20 24 WINDOW ERASE
```

Now let's load an image from disk into a free area of memory. The binary file "XYZ" on the GraFORTH disk contains an image of three arrows, each a different color, and each pointing a different direction. This is the same object that was used in the PLAY demonstration in Chapter 1.

```
Ready CR 132 PUTC PRINT " BLOAD XYZ,A2816 " CR
```

Before we can view "XYZ", we have to initialize the internal 3-D graphics array. Since we're starting from scratch, enter the word OBJERASE. OBJERASE clears the array, and should be used when beginning all 3-D programs.

```
Ready OBJERASE
```

Now we want to assign a number to the object we're about to view. Remember that GraFORTH can handle up to 16 objects at a time. The word OBJECT is used to specify which object to manipulate. OBJECT removes a number from the stack, and uses this number to select the current object. Let's give the image "XYZ" the number 0 in the array:

```
Ready Ø OBJECT
```

For our example, we will want the shape to be drawn automatically after each entered command. To do this, the word AUTODRAW is used. AUTODRAW removes a number from the stack. If this number is 1, then the currently selected object will automatically be drawn after each graphic command. If the number is 0, then automatic drawing will not occur. (Entering the word DRAW will draw the objects when AUTODRAW is not in effect.) Let's turn on automatic drawing with AUTODRAW:

```
Ready 1 AUTODRAW
```

We've initialized the array, set object number Ø, and turned on automatic drawing, but we haven't specified where the current object is in memory. The word OBJADR is used to specify this address. We loaded the object into memory starting at 2816, so this is the number we give to OBJADR:

```
Ready 2816 OBJADR
```

At this point (because AUTODRAW is turned on) the image will appear on the screen. Right now it looks like a single arrow with a line through it, but that's only because we're seeing it head-on.

GraFORTH has 12 separate words for controlling the position, size, and orientation of 3-D objects. We'll introduce these words properly in a bit, but to give you a taste, let's rotate the image a little for better viewing:

```
Ready 14 YROT
```

Now it's beginning to come into view, and you can see parts of all three arrows. Let's move it a little more:

```
Ready 16 XROT
```

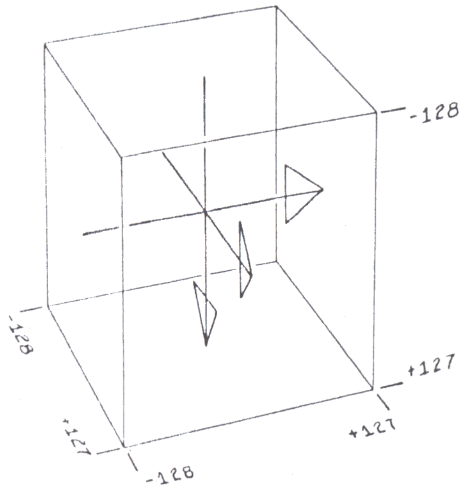
and add a little perspective:

```
Ready 6 SCALZ
```

3-D Image Format

Just as two-dimensional graphics use Cartesian coordinates labeled X and Y, three-dimensional graphics use a Cartesian coordinate system with the three directions labeled X, Y, and Z. The arrows in "XYZ" represent the three directions, or three 'axes'. X is a point along the horizontal, from left to right. Y is a point on the vertical, from top to bottom. Z is a point from rear to forward, pointing at the viewer.

The points that make up a 3-D image are expressed as three numbers, one for each of the X, Y, and Z coordinates. The valid range for each of these numbers is -128 to +127. Each arrow lies on an axis, with two coordinates equal to zero, and the ends of each arrow reaching from -128 to 127. At the center of the cube, where all three arrows meet, the three coordinates are all equal to zero.



The above diagram shows the limits for each of the three coordinates. Note that these limits define a "cube of space", 256 units along each side. All 3-D objects reside in this space. When more than one object is being displayed, each object has its own 3-D space, though these spaces may overlap or even coincide on the screen.

Image Parameters

Once an image has been loaded into memory and selected with OBJECT and OBJADR, it can be rotated, positioned, scaled, and translated in a number of ways.

Rotation

An image can be rotated around any axis, using XROT, YROT, or ZROT. XROT rotates the image around the X-axis, YROT around the Y-axis, and ZROT around the Z-axis. Each of these words removes a number from the stack and rotates the image to the selected angle. Angles are specified in units between 0 to 256 rather than degrees. An entry of 0 to YROT (or for that matter, XROT or ZROT) rotates the image around to a normal position facing the viewer. An entry of 64 rotates to 90 degrees, 128 rotates to 180 degrees, and so forth, until 256, which (like 360 degrees) is the same as zero: a full revolution.

Earlier, we used XROT and YROT to tip the image a bit so that we
could get a better view. We can also use a loop and cause the
image to rotate a full circle. The following word definition
executes YROT repeatedly, with an increasing rotation value:

```
: YSPIN
  260 0 DO
    I YROT
  4 +LOOP ;
```

```
Ready YSPIN
```

When YSPIN is finished, the object has a Y rotation of 0. To get
it back to our previous view, we enter the appropriate value for
YROT again:

```
Ready 14 YROT
```

XROT and ZROT can, of course, be manipulated in identical ways.

Scaling

The image can be changed in width or height with the words SCALX
and SCALY. Both of these words remove a number from the stack to
select the given X or Y scale. The valid range is from -31 to
+31. Numbers outside of this range will be "folded back" into
the range. When the 3-D object array is initialized with
OBJERASE, SCALX and SCALY are set to 16. Try these examples with
"XYZ":

```
Ready 25 SCALX

Ready 8 SCALY

Ready 4 SCALX
```

Setting a scale of zero causes the object to have no "thickness"
at all:

```
Ready 0 SCALX
```

Negative scale numbers reverse the image:

```
Ready -8 SCALX
```

Note: This reverse scaling is useful in unexpected ways. For
example, if you are creating the image of a bird, you only need
one wing image. The other wing is simply the first with one
negative scale number to reverse the image.

Here's a programming example of scaling:

```
Ready : SQUASH 12 -12 DO I SCALX LOOP ;

Ready SQUASH
```

Since for most graphics applications you will want to change both
the X and Y scale to change the total size of the object, the
GraFORTH word SCALE is provided. SCALE has the same form as
SCALX and SCALY. It simply sets both SCALX and SCALY to the same
value:

```
Ready 5 SCALE

Ready 12 SCALE
```

Three-Dimensional Perspective

There is a fourth scaling word in GraFORTH, SCALZ. SCALZ doesn't
change the size of the object in the same way that the other
scaling words do; instead it changes the perspective of the
object. Entries for SCALZ are also in the range -31 to 31. The
default value for SCALZ is zero, which doesn't provide
perspective views. (The front of a cube, for example, will be
the same size as the back.) If you enter a nonzero number for
SCALZ, perspective will be provided. If the entry is positive,
the front of the object will be larger than the back. If the
entry is negative, "reverse perspective" occurs, a most unusual
phenomenon! You may wish to try the following examples:

```
Ready  20 SCALZ YSPIN

Ready -10 SCALZ YSPIN

Ready   0 SCALZ YSPIN
```

Note: When SCALZ is nonzero, images take about 20% longer to
draw in exchange for the perspective features.

Also, SCALZ uses a fast algorithm that closely approximates true perspective. However, if you are displaying an image that has ends of lines meeting at the middle of a line, and you are using large amounts of perspective, the image may begin to distort. If this happens, break the image up into a series of shorter lines, so that all endpoints meet other endpoints, rather than meeting a line itself.

Position

Three-dimensional images can also be placed anywhere on the screen with the words XPOS and YPOS. XPOS and YPOS remove a number from the stack to determine the X or Y position on the screen of the center of the 3-D cube. Especially if the scale is large, to avoid screen wrap-around, ample room must be left on either side for the edges of the images. The valid entries for XPOS are 0 to 255; valid entries to YPOS are 0 to 191. The default values are 128 for XPOS and 96 for YPOS, which is the center of the screen.

To move the image around, let's first make it a bit smaller, to avoid wrap-around, then try a few different positions on the screen:

```
Ready 5 SCALE

Ready 50 XPOS

Ready  40 YPOS

Ready 200 XPOS
```

We can cause the feared wrap-around by placing the object close to one of the edges:

```
Ready 5 YPOS
```

Now let's move the image back to a more reasonable position:

```
Ready 96 YPOS
```

Translation

Translation occurs when tthe object is moved, not on the flat video screen, but within its own 3-dimensional space. In GraFORTH, objects can be translated along the X, Y, or Z axis with the words XTRAN, YTRRAN, and ZTRAN. When using translation, you must keep the image iinside the confines of its "cube of space". If you do not, tthen "3-D wrap-around" will occur, because GraFORTH cannot rrepresent points outside of its cube of 3-D space.

Our current image, "XYZ" already reaches to the edges of its space on all three axes. We can translate it, but wrap-around will occur immediately:

```
Ready 5 XTRAN
```

For some examples of trannslation, let's first load another 3-D image, one that doesn't ffill its space. We'll load and set up the image "HOUSE":

```
Ready ERASE

Ready CR 132 PUTC PRINT "" BLOAD HOUSE,A3000 " CR
         1 AUTODRAW
Ready 1 OBJECT  3000 OBJAADR
```

The image of a house shouuld appear. Let's get a better view:

```
Ready 20 XROT

Ready 10 YROT

Ready  8 SCALZ

Ready 10 SCALE
```

Now the house can be trannslated. It can be moved about a bit before causing wrap-arounnd. (In the next section, you'll see how to determine the true siize of an object from the IMAGEDITOR.)

```
Ready -50 ZTRAN

Ready  50 ZTRAN

Ready -25 XTRAN
```

Just for fun, try using YSPIN with the house, now that it has been translated away from the center of its space:

```
Ready YSPIN
```

Object Color

You noticed that each of the three arrows in "XYZ" was a different color. Images can be created with or without colors specified. If no color is specified, then the object's color can be determined when it is drawn later, using OBJCOLOR. OBJCOLOR removes a number from the stack to select the color of the current object. The usual GraFORTH color numbers are used.

The house does not have a set color, so we can set its color with OBJCOLOR:

```
Ready 1 OBJCOLOR

Ready 5 OBJCOLOR
```

Note that 3-D graphics, like two-dimensional and character graphics, can be done in either INVERSE or NORMAL, and either ORMODE or EXMODE, producing a wide variety of graphics effects. We encourage you to try some 3-D graphics commands with various combinations of display modes.

At the end of this chapter is a discussion of the program PLAY, which enables you to set all of these parameters (except for OBJCOLOR) into motion. PLAY is very useful in getting an intuitive feel for exactly what each of these parameters does.

The Image Editor

On the GraFORTH system disk is a file called IMAGEDITOR, which contains a program enabling you to create your own 3-D images. To use the image editor, first delete any new words on the word library to make room, then type:

```
Ready ABORT

Ready READ " IMAGEDITOR "
```

(NOTE: The image editor is a fairly large program. On non-language card systems, loading the image editor will move the top of the word library into the same memory used by the text editor program. If the editor is loaded into memory, it will overwrite the top of the word library, forcing you to reach for the power switch, as the GraFORTH system will become inoperable. After using the image editor, remember to FORGET the program before using the text editor.)

Now run the program:

```
Ready RUN
```

You will see a list of commands to the right and a prompt: "Enter command:". The image editor works with one 3-D image at a time.

Address and Image Selection

As in the character editor, you must select a work area address (or use the default address). To select an address, press "A" for "Address". You will see the prompt:

```
Enter File Address :
```

followed by the number "2816". (You should be getting pretty familiar with that number!) If you want to use another area of memory, enter that address. For this example, just hit <return>, and the address 2816 will be selected.

If you are doing these examples sequentially, the image "XYZ" will still be in memory at 2816. If you've turned the Apple off since that time, you will need to load it again. Type "G" for "Get" and enter the filename "XYZ". The file will be loaded into memory.

Getting a Good View

If the image was already in memory, it won't appear until you rotate it or move it on the screen. Images can be rotated, positioned, and scaled from the image editor.

To rotate the image, type "R". You will see:

```
Rotate [ X (num) to Z (num) ] :
```

For this command enter the letter of the axis you want to rotate around followed by the angle you want to rotate. For this example, type "Y16". The image will rotate around the Y-axis. Type "R" again and enter X16. Now you can see the arrows well.

To scale the object, type "S". You will see the prompt:

Scale [(num), or X,Y,Z (num)] :

To scale X and Y simultaneously, simply enter a number. To scale one of the coordinates, type X, Y, or Z, and then the scale number. Since we're keeping the image in the corner of the screen, it's best to keep the scale small. The scale is initially set to 8.

To change the position of the object, type "P". You will see:

Position [X (num) or Y (num)] :

Enter an X or a Y followed by the desired screen position. The image has an initial screen position of X=64 and Y=48. : X 50

You can choose a color for the image, if the color is not already set in the image file. Press "C" for "Color" and enter the desired color number. You can also choose between EXMODE and ORMODE views. Press "M" for "Mode", then enter "X" for EXMODE or "O" for "ORMODE".

Image File Entries

Now type "L" for "List" to see the numbers that make up the image. You can press <return> to see all of the entries or press ConTRoL-C to stop. Remember that, as explained above, GraFORTH uses Cartesian coordinates, a system of three numbers for each defined point.

Each entry in the IMAGEDITOR listing has the following information:

1. Whether the point is to be (M) moved to without drawing, or (D) drawn to from the previous line ending. (This means that each image file must begin with (M), not (D), since there are no previous lines at that time.)

2. What color should be used for the line. The color number (if present) is directly under the letter "C" in the heading. (If it is desired to use the word "OBJCOLOR" to specify object color, then don't make any color entries within the image file.)

3. The X, Y, and Z coordinates of the point (each coordinate lies within the range -128 to 127).

4. The address of the entry. Each entry occupies four bytes.

The last six lines of the image file can also be seen by pressing "E" for "Enter". We will use the "Enter" command in a moment to create our own 3-D shape. For now, press <return> to leave the "Enter" mode.

While using the image editor, you may want more screen space for text and less for image drawing, or vice versa. To accomplish this you can use "W" to move the text window up or down, position the image using "P", and scale the image using "S". The "List" and "Enter" commands will use as many lines as the text window allows.

Sometimes, while adjusting the image position, the image will "wrap around" on the graphics screen. If you want to clean up the screen, type "W" and reenter 14 or some other window top value. "W" clears the screen when it sets a new window.

Creating New Images

Now we will create our own image, a cube. First, we need to erase "XYZ". Press "Z", and you will see:

Erase File (Y/N) :

Type a "Y" to erase the file. The image won't disappear right away. (If the presence of the old image disturbs you, press "W" and enter 14 to cause the "Window" command to erase the screen.)

So that we will be able to see all sides of our object as it is created, enter a Z scale of 8 for perspective (press "S", then "Z8"). Now press "E" again. Notice that no file entries are listed, since we have erased them. You will see a prompt:

(M)ove, (D)raw, (-) Delete, (CR) Quit :

Since the first entry must be a move, type "M". You will be prompted for a color. Let's not use a color, so that later we can select its color with OBJCOLOR. Just press <return>.

You will then be prompted for X, Y, and Z values in turn. We're going to start with the point at the lower left front corner of the cube. X at the left is -127, so enter -127 and press <return>. Y at the bottom is 127. Enter 127 and press <return>. Z at the front is 127, so enter that and press <return>.

You still won't see anything drawn, because we have only defined a single point, and points aren't plotted in GraFORTH 3-D graphics, only lines. Now let's draw our first line. Type "D" this time instead of "M". Now enter an X value of 127 (remember the last entry was -127). We want the other two values to stay the same. In this "Enter" mode, to keep a previous value, just press <return>. The last value will be repeated. Press <return> for both Y and Z. Now a line will appear from left to right (from X = -127 to X = 127).

Now repeat the entry procedure, pressing "D" each time and changing only one number per entry, pressing <return> for the others:

Z to -127
X to -127
and Z to 127 again.

These entries will draw a square at the bottom of the image space. (If the view isn't very good, press <return> to leave "Enter" mode, change the rotation or the scaling, then press "E" to return to "Enter" mode.)

Note: If at any time you make an incorrect entry, just finish the entry, then press "-". "-" deletes the last entry in the file.

Now if we change Y to -127 and repeat the entire procedure, we will have most of the cube.

At this point three edges are still missing. Can you figure out how to draw the missing edges?

The solution is to (M)ove to each of the following locations, and (D)raw a vertical line (using Y) from bottom to top:

```
1. (M) X = 127,    Y = 127,   Z = 127
2. (D) X (same), Y = -127, Z (same)
3. (M) X (same), Y = 127,   Z = -127
4. (D) X (same), Y = -127, Z (same)
5. (M) X = -127, Y = 127,   Z (same)
6. (D) X (same), Y = -127, Z (same)
```

Saving the Image File

Now we can save our cube. Press <return> with no entry to leave the "Enter" mode, then press "K" for "Keep". You will be prompted:

Enter File Name to Keep :

Enter a file name here. The GraFORTH system diskette already contains a file named "CUBE". (It contains a cube identical to the one we just made here.) If you're using another disk, you can use the filename "CUBE" or another filename.

Three-Dimensional Display Methods

From within a program, the word DRAW is usually used instead of AUTODRAW to draw 3-D images. This way, several parameters can be changed at once before the next image is drawn. When AUTODRAW is off, executing DRAW causes the images to be drawn.

Aside from the mathematical methods (described in Appendix B), GraFORTH has a rather complex display method for 3-D images. In general, when a DRAW command is issued, the following events occur:

1. The drawing routines are directed at the graphics screen that is not currently being displayed, so that the drawing won't be seen.

2. The previous image on the invisible screen is "undrawn", using information stored when it was drawn.

3. The new image is drawn.

4. The display is switched to the freshly drawn screen.

This method guarantees high-quality animation images, since the entire process of drawing is concealed from the viewer.

You may wish to note that character graphics, discussed in the last chapter, also draws to both screens, so that character and 3-D graphics can be freely intermixed.

Redrawing Without Change

For maximum speed, an object is only redrawn by DRAW if a new command is issued to it. So in a program with several objects, only those that have been referenced since the last DRAW will be redrawn. Example:

```
0 OBJECT  16 XROT
3 OBJECT  24 YROT
DRAW
```

Only objects 0 and 3 will be redrawn when DRAW is executed.

If an object has been changed and then drawn, the images of the object on the two graphics screens will not be the same. If other objects are then repeatedly changed and drawn, causing GraFORTH to switch graphics screens, then the two unlike images of the object will be alternated, causing a back-and-forth type of residual motion.

Therefore, if several objects are being drawn independently, they should be referenced (using the word OBJECT), if not changed, to cause the image to be redrawn. This way, the images on both graphics screens will always be updated. For example,

```
1 OBJECT
```

causes a redraw of object 1 at the next draw command.

Erasing Individual Objects

The GraFORTH word OFF is used to "undraw" an object but not redraw it. Most objects stay on the screen after the last image entry to their tables. OFF selectively erases objects that are no longer needed. Subsequent commands to an object will redraw it. Here is an example of OFF:

```
Ready 3 OBJECT OFF
```

Overlapping Objects and UNDRAW

In a case where there are several overlapping objects, or objects are drawn over text, it is best to use "EXMODE", since this causes drawing and undrawing to occur without destroying the screen's original contents. Alternatively, if all the objects are in continuous motion, it may be desirable to use the word UNDRAW.

UNDRAW simply erases a block of character spaces specified by BLKSIZE, just as UNBLK does. However, UNDRAW also causes the next DRAW command to not do an automatic line "undraw" before drawing the next image. This way, you can use UNDRAW to erase the 3-D images yourself. Using UNDRAW is frequently faster than the automatic line undraw that is carried out by DRAW.

For example, let us say we have an image in the center of the screen (at X = 128, Y = 96) that extends 20 plotting points in radius around this point. Remember that numbers entered to BLKSIZE refer to characters, not points. Text characters of size 0 are 7 points wide and 8 points high. So an entry to BLKSIZE of 6 by 5 will cover an area 42 by 40 points, large enough for our sample image. Remember that UNDRAW, like UNBLK, is controlled by VTAB and HTAB. Let's set the blocksize, then position and execute an UNDRAW before the next DRAW:

```
Ready 6  5 BLKSIZE

Ready 18 VTAB  17 HTAB UNDRAW DRAW
```

Remember also that UNDRAW, like PUTBLK and UNBLK, doesn't advance HTAB across the screen as for printing. Once positioned, UNDRAW can be used repeatedly over the same area.

Other Effects

If you wish to prevent undrawing of the images (for special effects), simply use UNDRAW, but place the undraw block away from the image. For speed, select a blocksize of 1 by 1 in this case.

It is also possible to prevent screen sequencing altogether, using SEQUENCE, so that the process of drawing may be observed. SEQUENCE removes a number from the stack. If this number is a 0, screen sequencing is turned off. If the number is 1, screen sequencing is turned back on. This example will stop screen sequencing:

```
Ready 0 SEQUENCE
```

Usually used with "0 SEQUENCE", the word "SCREEN" selects which graphics screen to display. The screens are numbered 0 and 1. This example displays screen number 1:

```
Ready 1 SCREEN
```

PROFILE

There is another program on the GraFORTH system disk used for creating 3-D images, called PROFILE. PROFILE acts as a sort of graphics "lathe", creating images that are cylindrical in nature from a set of points defining the profile of the image. The file "CHAL" on disk contains the image of a chalice, and is an example of the kinds of images that can be created with PROFILE.

To run PROFILE, first make sure that there is room on the word library by FORGETting any extra words, then type:

```
Ready READ " PROFILE "

Ready RUN
```

Setting Parameters

You will see the PROFILE heading and some instructions. We're going to use PROFILE in this example to create a simple cone. The first question asked is:

```
Enter number of polygon sides :
```

This determines how smooth the cones circumference will be. For a perfect circle, you would ideally want to enter an infinite number of sides. Unfortunately, your Apple does not contain an infinite amount of memory! For this example, enter a 20.

The next prompt reads:

```
Enter Object File Address :
```

with a good ol' 2816 already selected for you. Images created with PROFILE can easily use a lot of memory. Usually you will want to use the area of memory beginning at 2816 or the space above the word library. (To find this address, print the value of PRGTOP after loading PROFILE, and add about 50 or 100 to this address for extra space.) For this example, just press <return> to keep the address 2816.

Entering Data from the Keyboard

Now you will see:

```
Data from [K]eyboard or [D]isk ?
```

You can either enter the profile coordinates directly from the keyboard or use a text file that contains the coordinates. Here we will enter the coordinates directly. Press "K" for "Keyboard". You will see:

```
Enter X,Y pair (end = "E") :
```

This is where you actually enter the coordinates. The Y coordinate is the vertical position in the profile. The valid range is -128 to 127. The X coordinate can actually be considered a radius, since it determines the distance from the edge to the center of the object. Its valid range is also -128 to 127, but negative entries are identical to positive ones, so only numbers from 0 to 127 need be used.

We're going to start our cone as a single point, and work down. The top of the cone is at Y = -128, and the radius (X) is zero. As we move down with increasing Y values, we'll also steadily increase the radius. Make the following entries:

```
Enter X,Y pair (end = "E") : 0,-128
Enter X,Y pair (end = "E") : 32,-64
Enter X,Y pair (end = "E") : 64,0
Enter X,Y pair (end = "E") : 96,64
Enter X,Y pair (end = "E") : 127,127
Enter X,Y pair (end = "E") : E
```

The last entry must be "E". For a few seconds, the phrase:

```
Generating image file (824 bytes) . . .
```

will appear on the screen as PROFILE computes the points that make up the cone, then the screen will be erased and the cone will appear. Notice that the cone has 20 vertical lines around its circumference. This is because we selected 20 polygonal sides. There are 4 circles around the cone and a point at the top. These are because we made 5 profile entries. At the bottom of the screen will be the message:

```
Enter object file name :
```

This is so you can save the 3-D object to disk. If you want to save the cone to disk, enter a filename and press <return>. If you don't want to save the image, just press <return> and the program will end.

Entering Data from Disk

As discussed earlier, PROFILE can also read a list of coordinates from a disk file. The textfile "BIGCHAL" contains a list of coordinates that describes the profile of a chalice. You may wish to see this list at some point. When PROFILE is no longer in memory, you can enter the text editor, get the file BIGCHAL, and list it. You will see a list of numbers similar to the one we entered to make the cone, but longer. Note that the last

entry in the file is "E", marking the end of the list. For now though, let's run PROFILE again, this time using the textfile BIGCHAL instead of keyboard entries. RUN the program, select 8 polygon sides, the address 2816, then "D" to read data from disk. You will then be prompted:

```
Enter Data File Name :
```

Enter the name "BIGCHAL". The disk will whir for a bit, then the message:

```
Generating image file (2724 bytes) ...
```

will appear. After a pause, the chalice will appear on the screen. As before, you can either save the 3-D image to disk, or press <return> to exit.

Memory Considerations

Because PROFILE can generate very large image files rapidly, image size checking has been added to help prevent overwriting important parts of memory.

Usually you will use one of two areas of memory for the 3-D image file when using PROFILE: either the free space from locations 2816 to 5887, or the space above the top of the word library. If you select an address between 2816 and 5887, PROFILE will prevent the image from extending beyond location 5887.

If you select an address greater then 5887, then PROFILE assumes the image is above the word library. It then checks for the presence of a language card. If you are using a language card, PROFILE will allow images to extend to location -16385, immediately below the Apple][I/O area. If you do not have a language card, PROFILE prevents the image from extending beyond location -26113, immediately below DOS.

If the image is too large to fit in the provided space, the image will not be created or drawn, and the following message will appear:

```
Not enough room here.
(Requires nnnn bytes.)
```

with nnnn being the actual number of bytes the image requires.

Notice that if the starting address you select is in a "safe" area of memory, then PROFILE will prevent the image from clobbering important information. However, if you select an address in the middle of something important, you'll find yourself having to reboot the system from scratch....

PLAYing Around

The program PLAY was briefly introduced in Chapter 1. PLAY was designed for you to "play" with a 3-D image, manipulating its rotation, scale, translation, and position parameters. Any or all of these parameters can be set into motion, giving you a rapid intuitive "feel" for what each of the parameters does. And PLAY is a lot of fun!

Note that PLAY, like IMAGEDITOR, uses the same memory as does the text editor on non-language card systems. Be sure to forget any extra words in the word library (PLAY is rather a large program), then type:

```
Ready READ " PLAY "

Ready RUN
```

The instructions are fairly self-explanatory. Once the image is loaded and you begin "playing", you can select a parameter with one of the number keys. To set the parameter in motion, press one of the arrow keys. The right arrow increases the parameter value; the left arrow decreases it. By pressing several number keys and arrow keys alternately, you can set a number of parameters in motion at once.

If any one parameter gets out of hand, you can press "F" to "freeze" its motion, leaving it at the current value. You can also press "D", to bring it back to its "Default" value.

If you want to pause everything, just press ConTRoL-S. The display will pause, and a flashing cursor will appear in the upper-left corner. Just press any key to resume. If you want to bring everything to a complete halt, press ESC. All motion will stop and all parameters will be set back to their default values. Finally, typing "?" will display the instruction screen again, and "Q" will quit the program.

Let's answer the start-up questions and get things moving:

The first prompt you will see is:

```
Image in [M]emory or on [D]isk?
```

If you already have an image in memory, press "M". If you want to load an image from disk now, press "D". For this example, press "D". Next is the now-famous address question:

```
Enter image address :
```

again with the number 2816 waiting for you. If you want to use the address 2816, just press <return>; otherwise enter the address you want. Press <return> for this example. If you selected to load an image from disk a moment ago, you will then see:

```
Enter image filename :
```

Type the name of the file you want to load. Let's load the file "HOUSE". Lastly:

```
Press Return to begin...
```

The screen will be erased and the image will appear. Along the right side are the values for each of the parameters. When you press a number key, the selected parameter will also be displayed on the bottom line with its current value and increment. Pressing the arrow keys will change the increment and set the object in motion.

You'll also see a question mark in the lower right corner. This is just to remind you that the instructions can be displayed at any time by typing "?".

With PLAY, it's very easy to get some of the parameters out of bounds, causing screen or "space" wrap-around. It doesn't hurt anything, and it can sometimes produce rather amusing effects!

Conclusion

We've now looked at all three kinds of graphics: two-dimensional graphics, character graphics, and three-dimensional graphics. With the information presented in these chapters, you can incorporate a wide variety of animated color graphics effects into your own programs, then use SAVEPRG to produce a system that boots and runs them automatically!

The next chapter explains how you can create music and sound effects with GraFORTH. (We'll also mention another program you may be interested in...) So without any further delay, on to chapter 9!

CHAPTER NINE: MUSIC WITH GRAFORTH

Chapter Table of Contents: Page

Introduction

GraFORTH has a sophisticated music synthesizer that plays through the Apple][built-in speaker. Notes may be played in nine distinct voices (not simultaneously). These features allow you to incorporate music or sound effects into your applications or game programs.

The two GraFORTH words that control the synthesizer are VOICE and NOTE.

VOICE

The GraFORTH word VOICE selects one of 9 voices in which to play notes. VOICE removes a number from the stack, and uses it to select a given voice. Here are the VOICE numbers and their meanings:

Number	Voice
-6 to -1	Selects a constant 'duty cycle' for the note, producing a note that is constant in volume. -1 = 50% duty cycle, -2 = 25% duty cycle, -3 = 12.5% duty cycle, etc. Smaller duty cycles decrease volume and increase the amount of high-frequency energy in the note.
0	Note begins at 50% duty cycle, then decreases to 0%. The note seems to die away.
1	The note begins at 0%, increases to 50%, then decreases again.
2	The note begins at 0%, then increases to 50%. The note seems to increase in volume.

NOTE

The GraFORTH word NOTE actually causes a note to be played. NOTE removes two numbers from the stack to select pitch and duration, then plays the note. The form for NOTE is:

```
<pitch>  <duration>  NOTE
```

The valid numbers for pitch and duration are in the range 2 to 255. Larger numbers for duration produce longer notes. Larger numbers for pitch produce lower pitched notes.

Let's play a couple of notes. The voice used if one has not been selected is voice 0. This example plays an "A" two octaves below middle A:

```
Ready 124 255 NOTE
```

Let's try a different note:

```
Ready 62 128 NOTE
```

This plays a note an octave higher for half as long. Now let's change the voice and play the same note:

```
Ready -1 VOICE

Ready 62 128 NOTE
```

Notice the change in tone quality. Experiment with the different voices to hear their differences.

Determining Duration and Pitch

The duration of a note is directly related to the size of the duration number. 255 can be considered a whole note, 128 a half note, 64 a quarter note, and so forth. Of course, if you want to play notes at a faster tempo, simply use smaller numbers.

Here is a table relating notes to the pitch numbers which produce them:

Note	Octave 1	Octave 2	Octave 3	Octave 4
A	248	124	62	31
A#	234	117	58	29
B	221	110	55	27
C	209	104	52	26
C#	197	98	49	24
D	186	93	46	23
D#	175	87	43	21
E	166	83	41	20
F	156	78	39	19
F#	147	73	36	18
G	139	69	34	17
G#	131	65	32	16

Useful Music Words

If you don't want to look up the pitches for each note, you can use the following program to generate the table and store it in a string array called "PITCH". Each element of PITCH, instead of containing a character, contains the pitch value for a note.

```
50 STRING PITCH

: COMPUTE.NOTES
  24870
  48 0 DO
     DUP 100 / I PITCH POKE
     DUP 18 / -
     DUP 1655 / -
  LOOP DROP ;

Ready COMPUTE.NOTES
```

Running COMPUTE.NOTES generates the table in PITCH. Now the pitch values for the 48 notes (numbered 0 through 47) can be found by reading the value from the proper element of PITCH. For example, the pitch value for the note 3 in the table (a "C" from the first octave) can be found in position number 3 in PITCH:

```
Ready 3 PITCH PEEK .
209
```

To play this note as a half note, you can enter:

```
Ready 3 PITCH PEEK 128 NOTE
```

You can also define a short word that retrieves the pitch value for you:

```
Ready : GETPITCH  PITCH PEEK . ;

Ready 3 GETPITCH
209
```

This word can be used with NOTE:

```
Ready 3 GETPITCH 128 NOTE
```

Since the notes are now numbered from 0 to 47, we can play all of the notes in the scale by using a loop:

```
Ready 48 0 DO I GETPITCH 32 NOTE LOOP
```

With a little patience, we can put together a song! The following word definition plays the first phrase from the "Happy Birthday" song:

```
: HAPPY.B
  12 GETPITCH 50  NOTE
  12 GETPITCH 50  NOTE
  14 GETPITCH 100 NOTE
  12 GETPITCH 100 NOTE
  17 GETPITCH 100 NOTE
  16 GETPITCH 200 NOTE ;
```

For longer tunes, repeating the words GETPITCH and NOTE will waste a lot of space. We wanted to show here how simply the tunes can be constructed. A much more efficient method is to store the numbers in memory or on the stack, and read them and play the notes from a loop.

Postscripts

Note: The quality of the synthesizer is higher than can be demonstrated with the Apple][built-in speaker. The use of a large external speaker is recommended for serious music work. See the Apple][Reference Manual or your local dealer for connection information.

For two-part music applications, the Electric Duet, also written by Paul Lutus, is available from Insoft. The Electric Duet plays 2 simultaneous notes through either the Apple speaker or an external amplifier, and can be used to play music directly from your GraFORTH programs. It contains a full feature music editor with the ability to transpose both note pitch and duration. Music can be directed to either the internal speaker or the Apple][tape output jack. The suggested price of the Electric Duet is only $29.95. For more information, contact Insoft or your local Apple dealer.

CHAPTER TEN: FINAL WRAP

We've made it! You have now been introduced to the GraFORTH system, from language features to complex graphics. From here on out, you will probably be using this manual more as a reference guide than as a tutorial; therefore, we suggest you get acquainted with the appendices. You will find the Word Library listings invaluable, and the Index very helpful for finding those definitions you've forgotten. The technical data section covers very useful information we suggest you at least browse through, and the GraFORTH diskette file listing and ASCII code tables are excellent references when you need them.

Please note that if you are using or intend to use GraFORTH to develop software for re-sale, we would like to talk with you. Insoft represents fine software (such as this!) for Apple, IBM, Atari, NEC and other popular microcomputers. Our royalty rates are among the best in the industry, and our support team is second to none. Let us show you why using our team of professionals makes good sense!

If you decide to market software on your own, please call us for information on a license agreement to use GraFORTH. There is <u>no fee</u> for this license, however, we do have a few restrictions on <u>how</u> it is marketed (We'll show you how to lock GraFORTH so that only <u>your</u> program can be run.) Either way, please contact:

Michael Brown
Insoft
10175 SW Barbur Blvd. Suite 202B
Portland, Oregon, 97219
(503) 244-4181

You now have a graphics system that is quite nearly limited only by your imagination! We hope you enjoy learning and using GraFORTH as much as we have enjoyed the opportunity to bring it to you!

APPENDIX A: WORD LIBRARY LISTING

The following is a list of the words in the GraFORTH word library. The list includes the word name, a "before and after" stack picture, the page number in the text where the word is first introduced, and a brief description of what the word does.

The stack picture shown represents relevant numbers on the top of the stack as letters. The top of the stack is to the right, as indicated by a dash. Three dashes represent an empty stack. How words use the stack can usually be inferred simply from the stack picture.

The word descriptions here are not meant to be comprehensive. For more information on each word, we suggest you refer back to the text, using the page numbers provided.

GraFORTH WORD LIBRARY LISTING

Word Name	Before	After	Page
"	- - -	- - -	3-13

A set of quotes surrounding text causes the text to be compiled into the program. Used with PRINT, ASSIGN, and READ.

Word Name	Before	After	Page
$LIST	- - -	- - -	5-30

Lists words in word library with hexadecimal addresses.

Word Name	Before	After	Page
'	- - -	a -	5-30

a = address of the word that follows ', and prevents that word's execution.

Word Name	Before	After	Page
(	- - -	- - -	4-14

Indicates the beginning of a program comment, to be passed over by the GraFORTH compiler.

Word Name	Before	After	Page	
*	m n -	p -	3-10	p = m * n (multiplication)
+	m n -	p -	3-6	p = m + n (addition)
+LOOP	n -	- - -	3-20	Marks the end of a loop structure, using n as a loop value increment.
,	- - -	- - -	5-32	Compiles a single byte within word definitions.
-	m n -	p -	3-10	p = m - n (subtraction)
->	(not applicable)		5-8	Causes the next variable reference to store the top stack value into the variable, rather than placing the variable value on the stack.
.	n -	- - -	3-6	Prints n.
/	m n -	p -	3-10	p = m / n (division)
:	- - -	- - -	3-14	Marks the beginning of an executable word definition.
;	- - -	- - -	3-14	Marks the end of a word definition.
<	n m -	p -	3-23	p = 1 if n < m, otherwise p = 0.
<=	n m -	p -	3-23	p = 1 if n <= m, otherwise p = 0.

Word Name	Before	After	Page	
<>	n m -	p -	3-23	p = 1 if n <> m, otherwise p = 0.
=	n m -	p -	3-23	p = 1 if n = m, otherwise p = 0.
>	n m -	p -	3-23	p = 1 if n > m, otherwise p = 0.
>=	n m -	p -	3-23	p = 1 if n >= m, otherwise p = 0.
ABORT	- - -	- - -	7-3	Restarts GraFORTH from scratch. The screen is erased, character size of 0, color of 3, all stack pointers initialized to 0.
ABS	n -	m -	3-10	m = absolute numeric value of n.
AND	n m -	p -	3-23	p = 1 if both n and m are nonzero, otherwise p = 0.
AREG	(variable)		5-31	Value of AREG is placed in processor A register before a CALL. After CALL, contents of A register are loaded back into AREG.
ASSIGN	a -	- - -	5-12	Places following quoted text into memory starting at address a.
AUTODRAW	n -	- - -	8-3	If n is nonzero, 3-D objects will automatically be drawn after every graphic command. If n is zero, this feature is turned off.
AUTORUN	n -	- - -	5-26	If n is nonzero, the top word library word will automatically execute at every return to the system. If n is zero, this feature is turned off.

GraFORTH Word Library Listing

Word Name	Before	After	Page
BASE	(variable)		5-22
Value determines what base numbers are accepted and displayed in.			
BEGIN	- - -	- - -	3-29
Provides a program return point for the words REPEAT and UNTIL.			
BELL	- - -	- - -	3-3
Beeps the Apple speaker.			
BINARY	- - -	- - -	5-22
Sets number input and output to base two.			
BLKSIZE	h v -	- - -	7-12
Selects a blocksize of h characters horizontally by v vertically for use by PUTBLK, UNBLK, and UNDRAW.			
BYE	- - -	- - -	5-32
Exits GraFORTH to Apple monitor.			
CALL	a -	- - -	5-31
Loads processor registers from AREG, XREG, YREG, AND PREG, calls machine language routine at address a, then stores register values.			
CASE:	n -	- - -	3-32
Selects and executes nth following word from list of words numbered starting from 0.			
CHRADR	a -	- - -	7-6
Selects a as address of current character set.			
CHRSET	(variable)		7-6
Value is address of default character set (2048).			
CHRSIZE	n -	- - -	7-3
Selects character size for subsequent character printing using PRINT, WRITELN, PUTC, and PUTBLK.			

GraFORTH Word Library Listing

Word Name	Before	After	Page
CHS	m -	n -	3-10
n = - m (change sign)			
CLEOL	- - -	- - -	5-4
Clears from the cursor position to the end of the current line.			
CLEOP	- - -	- - -	5-4
Clears from the cursor position to the end of the text window.			
CLOSE	- - -	- - -	5-24
Causes DOS to close any open files.			
CLRKEY	- - -	- - -	5-20
Clears the Apple][keyboard strobe so that a key can be read with GETKEY.			
COLOR	n -	- - -	6-6
Selects the color for line and large character drawing.			
CR	- - -	- - -	3-13
Prints a carriage return (ASCII value 141).			
DECIMAL	- - -	- - -	5-22
Sets number input and output to base ten.			
DO	m n -	- - -	3-19
Initializes a loop, using n for an initial value and m as an ending value.			
DRAW	- - -	- - -	8-15
Causes all 3-D objects referenced since the last DRAW to be drawn, using 3-D display methods.			
DROP	n -	- - -	3-7
Discards n from the stack.			
DUP	n -	n n -	3-7
Makes a copy of n on the stack.			

Word Name	Before	After	Page	
EDIT	- - -	- - -	4-2	Loads from disk (if necessary) and runs the appropriate text editor.
ELSE	- - -	- - -	3-27	Separates the two controlled areas in an IF - ELSE - THEN construct.
EMPTY	x y -	- - -	6-8	Erases a rectangular area from the last plotted point to (x,y).
ERASE	- - -	- - -	5-4	Erases both graphics screens.
EXMODE	- - -	- - -	6-10	Causes plotted points to turn on corresponding screen locations that are off, and turn off locations that are on.
FILL	x y -	- - -	6-4	Fills a rectangular area from the last plotted point to (x,y).
FORGET	- - -	- - -	3-17	Truncates the GraFORTH library back to the word that follows FORGET.
GETC	- - -	n -	5-20	Gets a single character from the keyboard, placing its ASCII value on the stack.
GETKEY	- - -	n -	5-20	Reads the keyboard without waiting, returning an ASCII value. Values over 128 are valid. Should be followed by CLRKEY.
GETNUM	a -	n -	5-14	Converts text string at address a into a number. Unsuccessful conversions return 0.

Word Name	Before	After	Page	
GPEEK	x y -	n -	6-12	Examines point at screen coordinates (x,y). n is nonzero if point is turned on, or 0 if point is turned off.
GR	- - -	- - -	3-38	Reestablishes normal GraFORTH input and output, and sets the graphic display mode.
HEX	- - -	- - -	5-22	Sets number input and output to base 16.
HOME	- - -	- - -	5-4	Erases the screen inside the text window and sets HTAB and VTAB to the upper left corner of the window.
HTAB	h -	- - -	5-3	Sets the column for subsequent printing.
I	- - -	n -	3-19	Returns the current innermost loop value.
IF	n -	- - -	3-25	If n is nonzero, words between IF and THEN (or IF and ELSE) are executed, otherwise execution continues after THEN (or between ELSE and THEN).
INVERSE	- - -	- - -	6-9	Complements the color for all text and graphics displays (including black-on-white text).
J	- - -	n -	3-20	Returns the loop value for the next outer loop.
K	- - -	n -	3-21	Returns the loop value for the third outer loop.
LINE	x y -	- - -	6-4	Draws a line from the last plotted point to (x,y).

GraFORTH Word Library Listing

Word Name	Before	After	Page
LIST	- - -	- - -	3-3
Lists the words in the GraFORTH word library.			
LOOP	- - -	- - -	3-19
Marks the end of a loop structure, incrementing the loop value and looping back to the word after DO if the loop value is less than the ending value.			
MAX	m n -	p -	3-10
p = the greater of m or n.			
MEMRD	a -	- - -	4-13
Reads and compiles text in memory starting at address a.			
MIN	m n -	p -	3-10
p = the smaller of m or n.			
MOD	m n -	p -	3-10
p = remainder after dividing m by n.			
MOVMEM	a b n -	- - -	5-30
Moves a block of n bytes from address a to address b.			
NORMAL	- - -	- - -	6-9
Resets normal color (white-on-black text) display.			
NOTE	p d -	- - -	9-3
Sounds a note of pitch p and duration d in the current voice.			
OBJADR	a -	- - -	8-3
Selects a as address of currently selected 3-D object.			
OBJCOLOR	n -	- - -	8-10
Selects color of current 3-D object.			
OBJECT	n -	- - -	8-3
Selects which object subsequent 3-D commands will refer to.			

GraFORTH Word Library Listing

Word Name	Before	After	Page
OBJERASE	- - -	- - -	8-3
Initializes the 3-D image array. Should be used at the beginning of 3-D graphics programs.			
OFF	- - -	- - -	8-17
Causes the next DRAW command to undraw the 3-D object.			
OR	m n -	p -	3-23
p is bit-wise OR of m and m. (p is nonzero if either m or n is nonzero, otherwise p = 0.)			
ORMODE	- - -	- - -	6-10
Causes points to be plotted regardless of what screen locations are on or off.			
OVER	m n -	m n m -	3-7
Copies m to top of stack.			
PAD	- - -	a -	5-15
Returns the address (812) of a 120-byte string space.			
PEEK	a -	n -	5-6
Reads a single byte n from address a.			
PEEKW	a -	n -	5-6
Reads number n from address a.			
PICK	..m n -	..m p -	3-7
Copies the nth stack item to top of stack.			
PLOT	x y -	- - -	6-4
Plots a point at (x,y).			
POKE	n a -	- - -	5-6
Stores single byte n at address a.			
POKEW	n a -	- - -	5-5
Stores number n at address a.			

GraFORTH Word Library Listing

Word Name	Before	After	Page	
POP	- - -	- - -	3-22	Discards top return stack value.
POSN	x y -	- - -	6-6	Establishes a position for a "last plotted point" without plotting.
PREG	(variable)		5-31	Value of PREG is stored in processor status register before a CALL. After CALL, value of status register is stored back into PREG.
PRGTOP	- - -	a -	3-3	Returns the address of the top of the word library.
PRINT	- - -	- - -	3-13	Prints following quoted text.
PULL	- - -	n -	3-22	Moves top return stack value to data stack.
PUSH	n -	- - -	3-22	Moves top data stack value to return stack.
PUTBLK	n -	- - -	7-13	Draws a block of characters with present blocksize starting with character number n at the current cursor position.
PUTC	n -	- - -	5-19	Prints character with ASCII value n at the current cursor position.
READ	- - -	- - -	4-14	Reads and compiles text from file with following quoted filename.
READLN	a -	- - -	5-12	Reads a line from keyboard into string starting at address a.

GraFORTH Word Library Listing

Word Name	Before	After	Page	
REPEAT	- - -	- - -	3-31	Marks the end of the BEGIN - WHILE - REPEAT construct, causing execution to jump back to words following BEGIN.
RND	- - -	n -	3-10	n is a random number.
RNDB	- - -	n -	3-10	n is a random number from 0 to 255.
RUN	- - -	- - -	5-26	Executes the top word on the word library.
SAVEPRG	- - -	- - -	5-27	Saves current system to disk.
SCALE	n -	- - -	8-7	Sets the X and Y scales for the current 3-D object.
SCALX	n -	- - -	8-6	Sets the X scale (width) for the current 3-D object.
SCALY	n -	- - -	8-6	Sets the Y scale (height) for the current 3-D object.
SCALZ	n -	- - -	8-6	Sets the Z scale (perspective) for the current 3-D object. Faster drawing occurs with a SCALZ of 0.
SCREEN	n -	- - -	8-18	Selects display of the given graphics screen (0 or 1).
SEQUENCE	n -	- - -	8-18	If n = 1, automatic screen sequencing for 3-D drawing is enabled. If n = 0, sequencing is enabled. (Default=1)
SGN	m -	n -	3-10	n = 1 if m > 0, 0 if m = 0, -1 if m < 0.

Word Name	Before	After	Page
SIN	m -	n -	3-10
n is a scaled sine value for m, in the range -128 to 127, repeating for every 128 numbers.			
SPCE	- - -	- - -	3-13
Prints a space (ASCII value 160).			
STACK	- - -	- - -	3-5
Toggles the stack display on or off.			
STRING	- - -	- - -	5-9
Declares a string array with following name, setting aside number of characters specified before STRING.			
SWAP	m n -	n m -	3-7
Swaps position of top two stack values.			
TEXT	- - -	- - -	3-38
Reestablishes normal GraFORTH input and output, and sets text display mode (no graphics).			
THEN	- - -	- - -	3-25
Marks the end of an IF - THEN construct, where execution continues from.			
UNBLK	- - -	- - -	7-14
Erases a block with present blocksize at the current cursor position.			
UNDRAW	- - -	- - -	8-17
Erases a block and prevents the next DRAW from performing an automatic line undraw.			
UNLINE	x y -	- - -	6-8
Erases a line from the last plotted point to (x,y).			
UNPLOT	x y -	- - -	6-8
Erases a point at (x,y).			
UNTIL	n -	- - -	3-29
If n = 0, execution jump back to words that follow BEGIN.			

Word Name	Before	After	Page
VALID	- - -	n - -	5-14
n is nonzero if last GETNUM produced a valid number, otherwise n = 0.			
VARIABLE	- - -	- - -	5-7
Declares a variable with following name. Any preceding number is used as the variable's initial value.			
VOICE	n -	- - -	9-2
Sets the voice for subsequent NOTE commands. Valid numbers are -6 to 2.			
VTAB	n -	- - -	5-3
Sets the row for subsequent printing.			
WHILE	n -	- - -	3-31
If n is nonzero, execution continues after WHILE, otherwise execution jumps to words after REPEAT.			
WINDOW	L w t b -	- - -	5-3
Sets a text window with left margin L, width w, top margin t, and bottom margin b.			
WRITELN	a -	- - -	5-12
Writes text to screen from string at address a.			
XPOS	n -	- - -	8-8
Sets X-position of current 3-D object to n.			
XREG	(variable)		5-31
Value of XREG is placed into processor X register before a CALL. After CALL, value of X register is stored back into XREG.			
XROT	n -	- - -	8-5
Sets rotation of current 3-D object around X-axis to n.			
XTRAN	n -	- - -	8-9
Translates current 3-D object along X-axis by n.			

GraFORTH Word Library Listing

Word Name	Before	After	Page
YPOS	n -	- - -	8-8
Sets Y-position of current 3-D object to n.			
YREG	(variable)		5-31
Value of YREG is placed into processor Y register before a CALL. After CALL, value of Y register is stored back into YREG.			
YROT	n -	- - -	8-5
Sets rotation of current 3-D object around Y-axis by n.			
YTRAN	n -	- - -	8-9
Translates current 3-D object along Y-axis by n.			
ZROT	n -	- - -	8-5
Sets rotation of current 3-D object around Z-axis by n.			
ZTRAN	n -	- - -	8-9
Translates current 3-D object along Z-axis by by n.			

APPENDIX A: WORD LIBRARY BY SUBJECT GROUP

Numeric Operator Words

CHS	ABS	SGN	RND	RNDB
MIN	MAX	POKEW	POKE	<>
=	>	<	>=	<=
OR	AND	PEEKW	PEEK	SWAP
DROP	POP	I	J	K
PULL	PUSH	DUP	OVER	PICK
MOD	/	*	+	-
SIN	BASE	DECIMAL	BINARY	HEX
MOVMEM	VALID	GETNUM		

Program Branching or Control Words

+LOOP	LOOP	DO	REPEAT	WHILE
UNTIL	BEGIN	IF	THEN	ELSE
BYE	STACK	FORGET	VARIABLE	RUN
AUTORUN	ABORT	READ	MEMRD	;
:	CASE:	(	CLOSE	EDIT
PRGTOP	SAVEPRG	->		

Input/Output Operator Words

HOME	CLEOP	CLEOL	GETC	GETKEY
CLRKEY	PUTC	.	$LIST	LIST

Text Display Function Words

VTAB	HTAB	CHRADR	CHRSET	CR
SPCE	TEXT	WINDOW	PRINT	ASSIGN
"	STRING	PAD	READLN	WRITELN

General Graphics Words

GR	GPEEK	ORMODE	EXMODE	ERASE
COLOR	INVERSE	NORMAL		

Two-Dimensional Graphics Words

POSN	PLOT	UNPLOT	LINE	UNLINE
FILL	EMPTY			

Character Graphics Words

PUTBLK	CHRSIZE	BLKSIZE	UNBLK

Three-Dimensional Graphics Words

SCREEN	DRAW	SEQUENCE	UNDRAW	AUTODRAW
OBJECT	OBJADR	OBJERASE	OBJCOLOR	SCALE
SCALX	SCALY	SCALZ	XPOS	YPOS
XTRAN	YTRAN	ZTRAN	XROT	YROT
ZROT	OFF			

Miscellaneous Words

CALL	PREG	AREG	XREG	YREG
,	'	NOTE	VOICE	BELL

APPENDIX B: TECHNICAL DATA

Table of Contents

GraFORTH Memory Map

0 to 255	$0000 to $00FF	6502 Page Zero. See Page Zero listing below.
256 to 511	$0100 to $01FF	6502 Stack
512 to 767	$0200 to $02FF	GraFORTH Line Input Buffer
768 to 811	$0300 to $032B	3-D Matrix scratch-pad area
812 to 935	$032C to $03A7	Compiler Stack, PAD String Area
936 to 975	$03A8 to $03CF	Graphics Horizontal Color Buffer
976 to 1023	$03D0 to $03FF	DOS Link Area
1024 to 2047	$0400 to $07FF	Text Display Screen (used for graphics also)
2048 to 2815	$0800 to $09FF	Primary character set storage area
2816 to 5887	$0B00 to $16FF	>>> User Free Space <<<
5888 to 6655	$1700 to $19FF	Image position and rotation data (See the Image Data listing below.)
6656 to 7679	$1A00 to $1DFF	Graphics address lookup tables
7680 to 7935	$1E00 to $1EFF	Data stack
7936 to 8191	$1F00 to $1FFF	Return stack
8192 to 16383	$2000 to $3FFF	Graphics screen 0
16384 to 24575	$4000 to $5FFF	Graphics screen 1
24576 to -32256	$6000 to $8200	GraFORTH System as delivered (Address approximate)

GraFORTH Memory Map

Without Language Card

-30720 to -28673	$8800 to $8FFF	Text editor file area (when used)
-28972 to -26113	$9000 to $99FF	Text editor program (when used)
-26114 to -16385	$9A00 to $BFFF	DOS 3.3
-16384 to -12289	$C000 to $CFFF	Apple][hardware I/O
-12288 to -1	$D000 to $FFFF	Apple][ROM area (Basic, Monitor)

With Language Card

-30720 to -18945	$8800 to $B5FF	Text editor file area (when used)
-18944 to -16385	$B600 to $BFFF	Text editor program (when used)
-16384 to -12289	$C000 to $CFFF	Apple][hardware I/O
-12288 to -1	$D000 to $FFFF	DOS 3.3 and Monitor

GraFORTH Page Zero Map

```
000-031 ($00-1F) not used
032-079 ($20-4F) Apple ][ monitor use
080     ($50)    GraFORTH text     pointer 1 (2 bytes)
082     ($52)    GraFORTH text     pointer 2 (2 bytes)
084     ($54)    GraFORTH graphics pointer 1 (2 bytes)
086     ($56)    GraFORTH graphics pointer 2 (2 bytes)
096-127 ($60-7F) not used (some DOS uses)
128-255 ($80-FF) used by GraFORTH
```

Useful locations in Page Zero:

128	($80)	last plotted X position
130	($82)	last plotted Y position
156	($9C)	pointer to data stack
157	($9D)	pointer to return stack
218-255	($DA-FF)	page zero matrix work area

Image Data Map

There are three data sets:

5888	$1700	undraw
6144	$1800	interim
6400	$1900	draw

Each data set contains 16 data tables, one for each of the 16 possible objects. Each data table is 16 bytes long:

Function	Relative Byte
Flag (draw, nodraw)	0
XROT	1
YROT	2
ZROT	3
XTRAN	4
YTRAN	5
ZTRAN	6
XPOS	7
YPOS	8
SCALX	9
SCALY	10
SCALZ	11
OBJCOLOR	12
Image Address	13 and 14

Each table begins at a multiple of 16. Therefore to find the object color for object 3:

```
16 * 3  (object 3)  + 12  (object color offset )
+ 6400  (data table base address) = 6460
```

Three-Dimensional Mathematical Method

The three-dimensional display method used in GraFORTH][uses a system of matrices that are successively multiplied to provide the ultimate position for each line in the displayed image.

In the following diagrams, (X) through (Z) refer to rotation angles, and X through Z refer to cartesian scalar values.

Matrix 1:

Scale X	0	0
0	Scale Y	0
0	0	Scale Z

Matrix 2:

1	0	0
0	COS(X)	-SIN(X))
0	SIN(X)	COS(X))

Matrix 3:

COS(Y)	0	-SIN(Y))
0	1	0
SIN(Y)	0	COS(Y))

Matrix 4:

COS(Z)	-SIN(Z)	0
SIN(Z)	COS(Z)	0
0	0	1

This matrix transformation occurs once per image. Then the result matrix is used to transform each line position using this last matrix:

X+XTRAN	Y+YTRAN	Z+ZTRAN
0	0	0
0	0	0

After this, if a nonzero value has been selected for SCALZ, a perspective computation is made (in which case image drawing is about 20% slower). The plotting coordinates then are offset by the user-provided XPOS and YPOS values, and the line is drawn.

Image Table Format

There are four bytes for each line entry in the 3D data table. Three of these bytes are one-byte signed numbers having a range of -128 to 127, and one byte contains data about color and whether to position or draw a line:

For each entry,

Byte 1 bit 7 (high bit) is set if a line is to be drawn, clear otherwise. Bits 0-2 contain a color number 0-7 (if zero, no color change). Use of zero is recommended, this makes it possible to control image color from the program using OBJCOLOR.

Bytes 2-4 are X, Y, and Z positions within the 3D space.

The end of the image table is indicated by having the data byte (1) be equal to 255 ($FF).

Word Library Structure and Compilation

Each word entry in the library consists of three parts:

1. A "pointer location" containing the address of thhe next lower word in the word library.
2. The word name (ASCII characters with high bit sett).
3. The executable machine language code for the wordd.

The hexadecimal numbers displayed by $LIST are the aaddresses of the pointer locations. A number returned by tic (') is the address of the executable portion of a word.

During compilation, GraFORTH separates the input linne by spaces into individual words, then searches through the library for each word. For each word search, GraFORTH first reads the curreent value of PRGTOP to find the top of the word library. It then looks here to find a pointer containing the address of the top word withiin the word library. Beginning with this first word, it followss the pointers from word to word down through the library. At each wordd, a check is made to see if this is the word being searched for. If tthe word is not found, the search falls through to a routine which aattempts to convert the word into a number. If this routine fails, the "Not Found" error is given.

Program lines are compiled directly into 6502 machiine language in the memory immediately above the top of the word librarry. If the line is an "immediate" command, and not part of a word defiinition, the machine language code is executed, then promptly forgotten.. If the line is part of a word definition, the code is saved, not eexecuted, and the word library expands.

At execution time, calls to other words are made thhrough direct machine language jumps. This is a major factor in the speed of GraFORTH.

Appendix C: Disk File Directory

TYPE	FILENAME	LENGTH	REMOVE OK?
B	OBJ.FORTH	36	NO
B	OBJ.EDITOR1	11	YES, IF 64K
B	OBJ.EDITOR2	11	YES, IF 48K
T	CHAREDITOR	21	NO
T	IMAGEDITOR	24	NO
T	PROFILE	15	NO
T	TURTLE	4	NO
T	PLAY	22	NO
T	STRING WORDS	4	NO
B	CHR.SYS	5	NO
B	CHR.STOP	5	YES
B	CHR.SLANT	5	YES
B	CHR.BYTE	5	YES
B	CHR.GOTHIC	5	YES
B	CHR.STUFF	5	YES
B	CHR.MAXWELL	5	YES
T	QUERY	2	YES - DEMO
T	HEADER	12	YES - DEMO
T	MENU	8	YES - DEMO
T	GRAPHICS1	8	YES - DEMO
T	GRAPHICS2	8	YES - DEMO
T	GRAPHICS3	10	YES - DEMO
T	TEXTDEMO	12	YES - DEMO
T	FORTHDESC	17	YES - DEMO
T	FLEDERMAUS	12	YES - DEMO
T	PIANO	11	YES - DEMO
T	CLOCK	5	YES
B	TETRA	2	YES - 3D
B	XYZ	2	YES - 3D
B	BAT	2	YES - 3D
B	CUBE	2	YES - 3D
B	HOUSE	2	YES - 3D
B	CHAL	10	YES - 3D
T	BIGCHAL	3	YES - PROFILE

Appendix D: ASCII Characters & Equivalent Numbers

Set	High Bit	Clear		
DEC	HEX	DEC	HEX	CHAR
128	80	0	00	ConTRoL-@
129	81	1	01	ConTRoL-A
130	82	2	02	ConTRoL-B
131	83	3	03	ConTRoL-C
132	84	4	04	ConTRoL-D
133	85	5	05	ConTRoL-E
134	86	6	06	ConTRoL-F
135	87	7	07	ConTRoL-G **(Bell)**
136	88	8	08	ConTRoL-H **(Left Arrow)**
137	89	9	09	ConTRoL-I
138	8A	10	0A	ConTRoL-J
139	8B	11	0B	ConTRoL-K
140	8C	12	0C	ConTRoL-L
141	8D	13	0D	ConTRoL-M **(Return)**
142	8E	14	0E	ConTRoL-N
143	8F	15	0F	ConTRoL-O
144	90	16	10	ConTRoL-P
145	91	17	11	ConTRoL-Q
146	92	18	12	ConTRoL-R
147	93	19	13	ConTRoL-S
148	94	20	14	ConTRoL-T
149	95	21	15	ConTRoL-U (Right Arrow)
150	96	22	16	ConTRoL-V
151	97	23	17	ConTRoL-W
152	98	24	18	ConTRoL-X
153	99	25	19	ConTRoL-Y
154	9A	26	1A	ConTRoL-Z
155	9B	27	1B	ESCape
156	9C	28	1C	Reverse Slash
157	9D	29	1D	]
158	9E	30	1E	Up Arrow
159	9F	31	1F	
160	A0	32	20	SPACE
161	A1	33	21	!
162	A2	34	22	"
163	A3	35	23	#
164	A4	36	24	$
165	A5	37	25	%
166	A6	38	26	
167	A7	39	27	'
168	A8	40	28	(
169	A9	41	29	)

Set	High Bit	Clear		
DEC	HEX	DEC	HEX	CHAR
170	AA	42	2A	*
171	AB	43	2B	+
172	AC	44	2C	,
173	AD	45	2D	-
174	AE	46	2E	.
175	AF	47	2F	/
176	B0	48	30	0
177	B1	49	31	1
178	B2	50	32	2
179	B3	51	33	3
180	B4	52	34	4
181	B5	53	35	5
182	B6	54	36	6
183	B7	55	37	7
184	B8	56	38	8
185	B9	57	39	9
186	BA	58	3A	:
187	BB	59	3B	;
188	BC	60	3C	<
189	BD	61	3D	=
190	BE	62	3E	>
191	BF	63	3F	?
192	C0	64	40	@
193	C1	65	41	A
194	C2	66	42	B
195	C3	67	43	C
196	C4	68	44	D
197	C5	69	45	E
198	C6	70	46	F
199	C7	71	47	G
200	C8	72	48	H
201	C9	73	49	I
202	CA	74	4A	J
203	CB	75	4B	K
204	CC	76	4C	L
205	CD	77	4D	M
206	CE	78	4E	N
207	CF	79	4F	O
208	D0	80	50	P
209	D1	81	51	Q
210	D2	82	52	R
211	D3	83	53	S

Set	High Bit	Clear		
DEC	HEX	DEC	HEX	CHAR
212	D4	84	54	T
213	D5	85	55	U
214	D6	86	56	V
215	D7	87	57	W
216	D8	88	58	X
217	D9	89	59	Y
218	DA	90	5A	Z
219	DB	91	5B	[
220	DC	92	5C	Reverse Slash
221	DD	93	5D	]
222	DE	94	5E	Up Arrow
223	DF	95	5F	
224	E0	96	60	°
225	E1	97	61	a
226	E2	98	62	b
227	E3	99	63	c
228	E4	100	64	d
229	E5	101	65	e
230	E6	102	66	f
231	E7	103	67	g
232	E8	104	68	h
233	E9	105	69	i
234	EA	106	6A	j
235	EB	107	6B	k
236	EC	108	6C	l
237	ED	109	6D	m
238	EE	110	6E	n
239	EF	111	6F	o
240	F0	112	70	p
241	F1	113	71	q
242	F2	114	72	r
243	F3	115	73	s
244	F4	116	74	t
245	F5	117	75	u
246	F6	118	76	v
247	F7	119	77	w
248	F8	120	78	x
249	F9	121	79	y
250	FA	122	7A	z

Appendix E: Index

GRAFORTH MANUAL ERRATA

As with any manual as comprehensive as GraFORTH's, a few "bugs" managed to creep past our editors. Please make note of the following changes:

PAGE	CHANGE
3-23	The last paragraph is inaccurate. The bitwise AND of some nonzero numbers will produce a zero result. However, the word AND is usually used with number comparisons that yield a 1 or 0. If both the top stack value and the second stack value are 1 (representing "true") then the AND of the two numbers will also be 1. If either or both numbers are zero, then the AND will be zero.
3-31	The BEGIN...WHILE...REPEAT diagram has the =0 and <>0 reversed. The text for this section is correct.
4-12	The fifth paragraph should refer to Appendix B, not D.
8-17	The word OFF does not immediately erase the currently selected object. It causes the next DRAW command to erase the object, without redrawing it. Subsequent commands to the object wll redraw it. The following example erases a 3-D object: `Ready  3 OBJECT OFF DRAW`

Program Control Words

RUN

The GraFORTH word RUN automatically executes the top word on the dictionary. This can be a great convenience when loading and running programs from disk. By using RUN, you don't have to check what the top word on the dictionary is after compiling a file in order to run it. In addition, if the top word has a name something like:

```
SUPER.ZAPPO.ELECTRO.BLASTERS.APPLE.VIDEO.GAME,
```

using RUN can save a bit of typing, too....

AUTORUN

The word AUTORUN goes a step beyond this. AUTORUN removes a number from the stack. If this number is nonzero, then GraFORTH will automatically execute the top word on the dictionary every time program control is returned to the GraFORTH system level (i.e. whenever you expect to see a "Ready" prompt). DOS errors, GraFORTH or machine language errors, executing the word ABORT, or pressing the Reset key with the AUTORUN option on will all cause the top dictionary word to be executed. Here is an example to give you a feel for the way AUTORUN works:

```
Ready : TEST PRINT " AUTORUN IS ON!!! " ;
```

We've added this word to the top of the dictionary so that AUTORUN will have a very visible effect.

```
Ready 1 AUTORUN
AUTORUN IS ON!!!

Ready 3 5
AUTORUN IS ON!!!
[3]
[5]
Ready SWAP
AUTORUN IS ON!!!
[5]
[3]
```

```
Ready ABORT
```

(The screen clears.)

```
GraFORTH ][ (C) 1981 P. Lutus
AUTORUN IS ON!!!
Ready
```

Fortunately, the AUTORUN option can be turned off by typing:

```
Ready 0 AUTORUN

Ready
```

If the top dictionary word runs a "closed" program which never exits to the system level, the AUTORUN option effectively makes the GraFORTH language itself inaccessible. Any errors or ABORTs simply restart the program.

Saving the GraFORTH System

The GraFORTH language is stored on the system disk as an executable binary file with the name "OBJ.FORTH". As mentioned in Chapter 3, when the disk is booted, this file is automatically loaded and run.

The GraFORTH word SAVEPRG is used to create GraFORTH binary files similar to OBJ.FORTH. SAVEPRG saves the current GraFORTH system, including any new words added to the dictionary, as a binary file. Once created, this file can be BRUN at any time, bringing the modified GraFORTH system back into memory.

SAVEPRG is a powerful tool. You can save "customized" systems, with your favorite special-purpose words already in the dictionary when the system is booted. You can also save finished applications programs, in such a way that the program automatically starts up when booted. This is ideal for games applications, where the obvious presence of a "language" is neither needed nor desirable.

To use SAVEPRG, first compile the words to produce the "finished" system you want to save, then type SAVEPRG:

```
Ready SAVEPRG

SAVE FILE NAME :
```

This prompt asks for the filename you want the new system saved as. The GraFORTH system disk automatically BRUNs the file "OBJ.FORTH", so if you want this new system to boot automatically, you should name your file "OBJ.FORTH" too. Your file will then overwrite the supplied GraFORTH system. (Make sure you're using a copy of the disk and not the original!) You are then prompted:

```
AUTORUN (Y/N) :
```

This prompt asks whether or not you want the saved system to boot up with the AUTORUN option on. If you answer Yes to this question, then the new system will automatically run the top word on the dictionary, starting a program in motion. If desired, your program can later turn the AUTORUN option back off, returning access of the GraFORTH language to the user. If you answer the AUTORUN question with No, the new system will display the "Ready" prompt on boot-up, with immediate access to the language.

After answering this question, this disk whirs for a bit, saving the new system to disk.

Note: As discussed in Chapter 2, a slightly modified version of DOS is used with GraFORTH. Any system saved with SAVEPRG requires this version of DOS to be in memory. New systems should be saved to a copy of the GraFORTH disk, so that the special DOS will be present.

The GraFORTH system as supplied includes an additional word on the top of the dictionary which asks the demonstration prompt on boot-up. This word can be found in the disk file "QUERY". The system was saved with the AUTORUN option on, so that the demo prompt would come up automatically. When you answer No to the demo question, the word turns AUTORUN off (freeing the system), then FORGETs itself! This leaves the system in its "usual" state.

The GraFORTH system can be saved to disk without the demo prompt simply by using SAVEPRG with no additional words on the word library. (This should only be done to a copy of your disk, in case lightning decides to strike while the system is being written to disk.) Boot the disk, answer No to the demo question, then type:

```
Ready SAVEPRG

SAVE FILE NAME :OBJ.FORTH

AUTORUN (Y/N) :N
```

After the disk stops whirring, turn your Apple off, then on again. When the system boots, the demo prompt will be gone.

You can also put the demo prompt back into the system. Type:

```
Ready READ " QUERY "
```

This adds the word that asks the demo question to the top of the dictionary. Now type:

```
Ready SAVEPRG

SAVE FILE NAME :OBJ.FORTH

AUTORUN (Y/N) :Y
```

The system will be saved with the demo prompt back in.

Overlays

GraFORTH programs can automatically load and run other GraFORTH programs, and even delete themselves to free up memory. Program segments that overwrite each other in this way are often called "overlays". The GraFORTH demonstration programs use overlays extensively.

To execute an overlay, include a word in the first file that reads the overlay. Make the first line in the overlay FORGET the words already in memory, and the last line in the overlay file the word RUN. To be more specific:

When you need an overlay, execute a READ <filename>, where <filename> is the name of the overlay. This file will now be read into memory, but since the first line of the overlay contains a FORGET <wordname>, where wordname is the name of the GraFORTH word you wish to forget back to (inclusive), the original file (or portion thereof) will be removed. As reading of the overlay continues, it will now fill memory previously occupied by the original file.

We urge you to examine the demonstration file listings as an example of overlays. Since the FORGET at the beginning of each file does not cause an error if the word being forgotten does not exist, the demo files (or any overlay) can also be directly loaded and run.

Moving Memory

MOVMEM simply moves a block of memory from one location to another. MOVMEM removes three numbers from the stack. The form for MOVMEM is:

```
<source>  <destination>  <# of bytes>  MOVMEM
```

The <source> number is the starting address of the data to be moved. The <destination> is the address of where the block is to be moved to. <# of bytes> specifies how many bytes are to be moved. For example, to move 256 bytes from address 2048 to address 2816, enter:

```
Ready 2048 2816 256 MOVMEM
```

MOVMEM can be handy for relocating character sets and 3-D images in memory, as will be discussed in Chapters 7 and 8.

Retrieving Word Addresses

The word ' (an apostrophe, also called a "tic") places on the stack the address of the word that follows it, and prevents that word from being executed. Here is an example:

```
Ready ' ERASE
[30749]
```

The tic placed the address of the word ERASE on the stack, and prevented ERASE from being executed. Note that the tic is a word that looks forward down the input line, and retrieves the address when it is compiled, not every time it is executed.

The address returned by "tic" is always greater than the hexadecimal address shown with $LIST. This is because the $LIST address indicates the beginning of the word definition, and "tic" returns the address of the executing portion of the word. See Appendix B for more information on the word library structure.

Calling Machine Language Routines

Machine language programs in memory can be called directly from GraFORTH with the word CALL. CALL removes a number from the stack, interprets it as a memory address, then calls the machine language routine at that address. (The routine should end with an RTS (ReTurn from Subroutine) instruction to return to GraFORTH properly.) Machine language programs can be loaded from disk using the DOS command "BLOAD" into any free area of memory, then CALLed from GraFORTH.

Before a machine language CALL is made, values can be placed in the Apple processor's A, X, Y and P registers using the GraFORTH variables AREG, XREG, YREG and PREG. Before making the machine language CALL, simply place the desired values into AREG, XREG, YREG and PREG as you would any other variable. When CALL is executed, it loads the processor registers with the values from these variables before doing the call. (Note the importance of loading a proper value into PREG. If improper processor bits are set, GraFORTH will not operate!) After the routine has executed, the values of the registers are loaded back into the variables and can be read from GraFORTH, again, just as any other variable.

Here is a nice example, which uses CALL to read the game paddles. The Apple System monitor contains a routine at location -1250 for reading the game paddles. It expects to see the number of the game paddle (0 to 3) in the processor's X register. It returns a number from 0 to 255 (based on the position of the paddle) in the Y register. The following word reads the value of a game paddle by placing the top stack value in XREG, calling the paddle routine, then placing the value of YREG on the stack:

```
: READ.PADDLE
  -> XREG
  -1250 CALL
  YREG ;
```

(The Apple manuals warn that two consecutive readings of a game paddle can produce incorrect results, and suggest a short wait loop between readings.)

Compiling Number Tables

The word "," (comma) causes a number to be compiled as a byte directly into GraFORTH. Small assembly language routines can be compiled using commas, or number tables can be generated. Here is an example of a word that contains a number table of the visible high resolution colors. The numbers are stored as individual bytes following the word name in memory:

```
: COLOR.TABLE 1 , 2 , 3 , 5 , 6 , ;
```

These numbers correspond to the colors green, violet, white, orange, and blue. (Colors in GraFORTH will be discussed in detail in the next chapter.) Each number can be accessed by using the tic to retrieve the address of COLOR.TABLE, then adding an offset (0 to 4) to pick out the appropriate number with PEEK. Note that COLOR.TABLE is not an executable word!

The comma is the only GraFORTH word that assembles directly at the byte level, and some precautions are required to use it effectively. The comma should only be used within word definitions. Also, for internal reasons, the first byte of an assembly of code or data may not be greater than 127 (hexadecimal $7F), nor can it be equal to 10 ($A). Here are the reasons: 10 is a special reserved compiler flag, and a number less than 128 must follow each GraFORTH word name to mark its end. (For more information, see Appendix B for technical information on GraFORTH's dictionary link structure.)

Leaving GraFORTH (gently)

The GraFORTH word "BYE" can be used to enter the Apple][system monitor. The GraFORTH language begins at hex location $6000. To restart GraFORTH from the monitor, type "6000G".

Conclusion

That about wraps up the language features of GraFORTH. From here on out we'll be talking about the many types of graphics available with GraFORTH. (That is what you bought it for, isn't it?) The next chapter will cover basic point and line drawing in GraFORTH, as well as a discussion of the supplied TURTLEGRAPHICS. We'll get into the various modes, color selections and...

Well, that's the topic of chapter 6!

CHAPTER SIX: TWO-DIMENSIONAL GRAPHICS

Chapter Table of Contents: Page