Micro Technology Unlimited

K-1008-8 KEYWORD GRAPHICS PACKAGE

FOR COMMODORE PET AND CBM COMPUTERS

Micro Technology Unlimited
2806 Hillsborough Street
P.O. Box 12106
Raleigh, North Carolina 27605 USA
(919) 833-1458

TABLE OF CONTENTS

The Keyword Graphic Package was written be Martin J. Cohen, Ph.D. with assistance from Ron Holt, Larry Isaacs, and Gregory Yob. This manual was prepared by Hal Chamberlin and Larry Isaacs. We thank them all for a job well done!

1. OVERVIEW

The Keyword Graphics Package, or KGP for short, is a 6502 machine language program that extends the command repertoire of PET BASIC to include over 45 graphics commands for the Micro Technology Unlimited Visible Memory. Besides being much easier to use and understand than previous software packages for the Visible Memory, it is much faster since manipulation of the 64,000 picture elements in the display is done at machine language speed.

For example, if a solid vector between the coordinates 35,21 and 117,73 is desired, the BASIC statement:

```
130 LINE 35,21,117,73
```

would be inserted into the user's program (or by omitting the line number, the vector would be drawn as soon as the command was typed in). A caption located at X=220 and Y=123 could be generated simply by coding:

```
710 MOVE 220,123
720 CHAR "MARKET INDEX"
```

In addition the package provides some powerful advanced functions not found in other graphics packages such as automatic coordinate transformation (both translation and scaling), solid or dotted lines, keeping track of up to 4 different "display windows", subimage definition, and even a "scroll" command to facilitate the programming of animated displays.

1.1 GRAPHIC DISPLAY CHARACTERISTICS

The MTU Visible Memory is the graphics display device used by this software package. The display itself is very simple in concept consisting of 64,000 dots arranged in a rectangular array 320 dots wide by 200 dots high. Each of these dots may either be white (or green with the newer PET's), in which case it shows as a small point of light; or black, in which case it does not show up. Graphic figures are typically drawn by turning dots on to approximate the figure's outline which means that the figure is white against a black background. A simple command is provided for "drawing" black dots on a white background instead if the user prefers that mode of display.

With the Visible Memory installed, there are actually two sources of video that can be displayed on the PET screen. While programming, the user will want to see the normal PET character display. While running a graphics program, the Visible Memory should be shown instead. While debugging, it is convenient to be able to see both images superimposed (K-1008-6 Integrated Visible Memory only). Commands are provided to rapidly switch between the two video sources with a minimum of effort.

1.2 PLOTTING FEATURES

The most common use for graphics is plotting graphs and other images from mathematical functions or measured data. When the system is initialized, the origin of the plotting grid (the 0,0 point) is set to the lower left corner of the screen. X coordinates in the range of 0 through 319 and Y coordinates in the range of 0 through 199 are acceptable. For greater flexibility, a coordinate offset can be specified which has the effect of moving the origin. Independent X and Y scale factors can also be specified which will shrink or expand the image in either or both dimensions.

When plotting, a virtual grid having X and Y ranges of -32767 to +32767 is actually available. If a figure larger than the visible limits of the screen is being drawn, the visible part is still drawn correctly. Different parts of the overall image may be seen by resetting the X and Y offsets and then redrawing the entire image.

As an added convenience, storage and automatic handling of 4 display windows is provided. This allows the implementation of split-screen and other multiple display techniques with much of the housekeeping associated with switching among windows to be done at machine language speed.

Two different kinds of subimage capability are built-in. Subimages are useful for cases where shapes from a library of shapes are to appear in an image several times at different locations. Examples include various types of schematic diagrams and even ordinary text characters. The "vector-byte" type of subimage features compact storage of the shape definition (one byte per line segment) in exchange for restricted shape size and line angles. A default ASCII character set encoded in vector-byte form is a standard feature of the package. The "relative coordinate" type of subimage allows shapes nearly as large as the screen with lines at any angle to be defined at the expense of doubled storage requirements.

1.3 ANIMATION FEATURES

While any type of microprocessor driven stored image display is limited in its animation capability, two features of this package combine to provide animation from BASIC nearly as good as dedicated machine language programming. One of these, the vector-byte and relative coordinate subimage feature, means that entire images can be redrawn in different locations at machine language speed merely by specifying the new location of the subimage. The other is a "scroll" function which will move entire portions of the screen from one location to another without the need to erase and redraw them in the new location.

1.4 HOW TO USE THIS MANUAL

The first part of this manual is devoted to getting the customer started in the use of this package and the Visible Memory. Besides the Keyword Graphics Package itself, the distribution cassette has two demonstration programs recorded on it. It is strongly suggested that at least the first demonstration be run to convince yourself that the PET, Visible Memory, and Keyword Graphics Package are functioning correctly. The demonstration also gives an idea of the capabilities, appearance, and speed that can be expected of the user's own graphics programming.

The user's manual itself is basically divided into two parts. The first part, which is sections 4 and 5, is written as a tutorial which not only describes the graphics commands in detail but also teaches the reader many fundamental concepts of computer graphics in general. It should be read carefully by customers relatively inexperienced in graphics and at least skimmed by more experienced customers. The second part, which is sections 6-10, can be referred to directly by experienced users for concise descriptions of the commands and other system characteristics. The entire manual is written assuming a good knowledge of PET BASIC. If the reader is inexperienced with BASIC, the PET manuals should be studied in conjunction with this manual. In addition, machine language concepts and terms will be used occasionally. This should not be much of a problem for the inexperienced user as only the more advanced functions are involved.

2. LOADING THE KEYWORD GRAPHICS PACKAGE

The Keyword Graphics Package is a RAM resident program. Consequently, it must be loaded from tape (or disk) every time the PET is turned on or the memory it occupies is used for another purpose. After loading, it must be enabled to activate the additional BASIC commands.

2.1 HARDWARE REQUIREMENTS

The Keyword Graphics Package requires the following hardware:

16K Verison

1. PET or CBM computer with new[1] ROM's
2. PET cassette to read program tape
3. RAM memory from \$0000 tn \$3FFF (16K)
4. K-1007 and K-1008 Visible Memory or K-1008-6 Integrated Visible Memory addressed at \$4000, \$6000, or \$9000
5. K-1007-2 (old PET's) or K-1007-3 (new PET's) connector board

24K Verison

1. PET or CBM computer with new[1] ROM's
2. PET cassette to read program tape
3. RAM memory from \$0000 to \$5FFF (24K)
4. K-1007 and K-1008 Visible Memory or K-1008-6 Integrated Visible Memory addressed at \$6000 or \$9000
5. K-1007-2 (old PET's) or K-1007-3 (new PET's) connector board

32K Version

1. PET or CBM com,uter with new[1] ROM's.
2. PET cassette to read program tape
3. RAM memory from \$0000 to \$7FFF (32K)
4. K-1007 and K-1008 Visible Memory or K-1008-6 Integrated Visible Memory addressed at \$9000.
5. K-1007-2 (old PET's) or K-1007-3 (new PET's) connector board

NOTE 1: Systems with new ROM's can be identified by observing the display immediately after power-on. If the first line reads:
COMMODORE BASIC
then you have the new ROM's.

2.2 DISTRIBUTION CASSETTE CONTENTS

The following files are recorded on the program distribution cassette:

1. MTU KGP 16K	16K version of the Keyword Graphics Package
2. MTU KGP 32K	32K version of the Keyword Graphics Package
3. MTU KGP 24K	24K version of the Keyword Graphics Package
4. KGP DEMO	Generalized demonstration program
5. KGP EXDRAW	An illustrative figure drawing program

2.3 LOADING INTO 16K PET'S

The following describes how to load the 16K version of the Keyword Graphics Package. If you have a 24K or 32K PET and wish to use its greater memory capacity, skip to the next section.

1. Turn the PET off, then on to insure that BASIC is properly initialized.
2. The Visible Memory should be connected to the PET and able to run the demonstration and disgnostic programs in the K-1007 or K-1008-6 manual.
3. Enter the command: POKE 53,34 then a Carriage Return. This action reserves memory space for the Keyword Graphics Package.
4. Enter the command: NEW then a Carriage Return. This makes BASIC "digest" the effects of the previous command.
5. Place the K-1008-8 cassette in the recorder and make sure it is rewound. Then enter the command: LOAD "MTU KGP 16K" and then a Carriage Return.
6. When BASIC has found and loaded the file it will print READY. At this time, enter the command: NEW and then a Carriage Return. If this is the first time the package was loaded, see paragraph 2.7 for instructions on making a backup copy of the file to protect against loss.
7. Enter the command: SYS256*34 and then a Carriage Return which enables the graphic command processor. If the Visible Memory is not addressed at $4000, enter the command: VMPAGE 96 if it is at $6000 or VMPAGE 144 if it is at $9000. The KGP is now ready to use. If desired, go to section 3 for instructions on running the demonstration programs.

2.4 LOADING INTO 32K PET'S

The following describes how to load the 32K version of the Keyword Graphics Package. If you have a 24K PET or if your Visible Memory is not addressed at $9000, please refer to the next section which describes loading into a 24K PET.

1. Turn the PET off, then on to insure that BASIC is properly initialized.
2. The Visible Memory should be connected to the PET and able to run the demonstration and disgnostic programs in the K-1007 or K-1008-6 manual.
3. Enter the command: POKE 53,98 then a Carriage Return. This action reserves memory space for the Keyword Graphics Package.
4. Enter the command: NEW then a Carriage Return. This makes BASIC "digest" the effects of the previous command.
5. Place the K-1008-8 cassette in the recorder and make sure it is rewound. Then enter the command: LOAD "MTU KGP 32K" and then a Carriage Return.
6. When BASIC has found and loaded the file it will print READY. At this time, enter the command: NEW and then a Carriage Return. If this is the first time the package was loaded, see paragraph 2.7 for instructions on making a backup copy of the file to protect against loss.
7. Enter the command: SYS256*98 and then a Carriage Return which enables the graphic command processor. The KGP is now ready to use. If desired, go to section 3 for instructions on running the demonstration programs.

2.5 LOADING INTO 24K PET'S

The following describes how to load the 24K version of the Keyword Graphics Package. This version is for those who have expanded an 8K PET to 24K by the addition of a 16K memory expansion. It is also for people with 32K PET's who have elected to overlap the Visible Memory with the last 8K of PET memory in order to avoid changing the PET's internal addressing jumpers (see the K-1008-6 or K-1007 manuals).

1. Turn the PET off, then on to insure that BASIC is properly initialized.

2. The Visible Memory should be connected to the PET and able to run the demonstration and disgnostic programs in the K-1007 or K-1008-6 manual.

3. Enter the command: POKE 53,66 then a Carriage Return. This action reserves memory space for the Keyword Graphics Package.

4. Enter the command: NEW then a Carriage Return. This makes BASIC "digest" the effects of the previous command.

5. Place the K-1008-8 cassette in the recorder and make sure it is rewound. Then enter the command: LOAD "MTU KGP 24K" and then a Carriage Return.

6. When BASIC has found and loaded the file it will print READY. At this time, enter the command: NEW and then a Carriage Return. If this is the first time the package was loaded, see paragraph 2.7 for instructions on making a backup copy of the file to protect against loss.

7. Enter the command: SYS256*66 and then a Carriage Return which enables the graphic command processor. If the Visible Memory is not addressed at $6000, enter the command: VMPAGE 144 for Visible Memory at $9000 and then a Carriage Return. The KGP is now ready to use. If desired, go to section 3 for instructions on running the demonstration programs.

2.6 USE WITH COMMODORE DOS SUPPORT AND OTHER COMMAND PROCESSORS

When enabled, the KGP inserts itself into BASIC and looks at all of the commands entering the system before BASIC does. If a graphic command is recognized, it is acted upon then discarded so that BASIC never sees it. This technique is not unique however since Commodore DOS Support (also known as "the wedge") typically used with the Commodore disk and other command enhancers (such as the Programmer's Toolkit) also use it. To prevent conflicts and possible crashes, it is important that the KGP be loaded and enabled last in the sequence of bringing up the system. The procedure to use with the disk wedge is as follows:

1. Follow steps 1-4 of the appropriate loading procedure described in paragraphs 2.3, 2.4, or 2.5.

2. Load and enable DOS Support according to Commodore's instructions.

3. Complete the KGP loading instructions starting at step 5.

4. Any other command interpreter should be loaded and enabled between steps 2 and 3 above. (Only the BASIC Programmer's Toolkit (tm) has actually been tested; proper operation with others is not guaranteed.

2.7 MAKING A BACKUP COPY

In order to protect against wearout or loss of the K-1008-8 distribution tape and to optimize reading accuracy on the user's system, it is recommended that a backup copy be made of the KGP the first time it is successfully loaded. After copying, make sure the extra copyright notice label is sffixed to the copy cassette. To make a copy of the Keyword Graphics Package, do the following:

1. Follow steps 1-6 of the KGP loading procedure. Do not enable it however.

2. Type the command: SYS1024 to enter the machine language monitor.

3. If a cassette backup is to be made and you have a 16K PET, enter the following command: `.S "MTU KGP 16K",01,2200,4000` then start the tape in record mode.

4. If a cassette backup is to be made and you have a 32K PET, enter the following command: `.S "MTU KGP 32K",01,6200,8000` then start the tape in record mode.

5. If a cassette backup is to be made and you have a 24K PET, enter the following command: `.S "MTU KGP 24K",01,4200,6000` then start the tape in record mode.

6. If a disk backup is to be made and you have a 16K PET, enter the following command: `.S "0:MTU KGP 16K",08,2200,4000` (backup on drive 0)

7. If a disk backup is to be made and you have a 32K PET, enter the following command: `.S "0:MTU KGP 32K",08,6200,8000` (backup on drive 0)

8. If a disk backup is to be made and you have a 24K PET, enter the following command: `.S "0:MTU KGP 24K",08,4200,6000` (backup on drive 0)

9. Enter an X command followed by a Carriage Return to return to BASIC.

10. Copies of the demonstration program and EXDRAW may be made in the usual way since they are entirely BASIC programs.

11. Affix the extra Copyright label supplied with the distribution tape to the backup copy.

3. RUNNING THE DEMONSTRATION PROGRAMS

Two demonstration programs have been included on the distribution cassette. Their purpose is two-fold. First they allow the first time user to get the Visible Memory to do "something useful" immediately after unpacking and installing it with a minimum of effort. Second, the listings of the demonstration programs serve as examples of how to use the Keyword Graphics Package to perform a variety of graphics functions. While not every possible command or graphic technique is exploited in the demo programs, an effort was made to include as many as possible. In the descriptions, specific KGP commands are sometimes referred to. Consult sections 4 and 5 for detailed description of these and other commands.

3.1 LOADING KGPDEMO

To load and run the generalized Keyword Graphics Demonstration, do the following:

1. Load and enable the Keyword Graphics Package as described in sections 2.3, 2.4, or 2.5.

2. Enter the command: LOAD "KGPDEMO" , start the tape in read mode, and wait for PET BASIC to print READY.

3. Enter the command: RUN which should switch to Visible Memory video and then start the sequence of demonstrations.

3.2 DESCRIPTION OF KGPDEMO

The first program illustrates point plotting by drawing a circle (depending on adjustment of the PET's display monitor, it probably will not be perfectly round) with 250 individual dots. The parametric equations: X=COS(A) and Y=SIN(A) are used to generate X,Y pairs as a function of the variable, A, which varies from zero to 2*Pi. The MOVE command is used to put the "cursor" at the desired X,Y position while the WRPIX command actually plots the point. Note that offsetting and scaling of X and Y, which vary between -1 and +1, is necessary to produce the appropriate X and Y values for plotting. KGP can also be instructed to do the offsetting and scaling (with restrictions) automatically if desired.

The second program illustrates vector plotting by creating a 31 point star. Since the string of lines is connected, the DRAW command can be used to draw from the current cursor position to the next endpoint. However the first endpoint is a special case. To handle the first point, a variable called FP is initially set to 1. As each new endpoint is computed, the value of FP is interrogated. If it is found to be non-zero (which will only occur for the first point), a MOVE is done instead to position the cursor without drawing. After the first point is plotted, FP is set to zero thus allowing vectors to be drawn between all successive points.

The 31 point star is actually drawn 4 times in different graphics modes (GMODE) to illustrate their effect. It is first drawn in mode 1 which gives normal line plotting. Next it is "drawn" in mode 2 which is actually an erase mode since it draws black lines. Note that when two lines cross and one of them is erased that a small gap is left in the other line. This is a fundamental problem of all stored image (as opposed to refresh vector) graphic displays. The last two times the star is drawn in mode 0 which is "flip mode". In flip mode, the state of each point plotted is made the <u>opposite</u> of what it already is. Thus on a black background it produces lines as in mode 1. However when lines cross, the point of intersection is flipped twice resulting in a gap. Note that drawing the star the fourth time in mode 0 erases it (flips white back to black) but that erasing "repairs" the gaps at line intersections! This is a very powerful property of flip mode which will be explained later.

The third program illustrated how a fully labelled and captioned graph can be produced. First the Y axis labels are produced with a FOR loop, number conversion into a string variable, and the CHAR command to print the labels in the desired positions along the Y axis. Note that if the FOR loop had been written: FOR Y=-1 to 1 STEP .2 that after 10 iterations Y would not be precisely 0 because of round-off error in decimal fraction to binary floating point conversion. Thus rather than 0 being printed, something like -1.16415322E-10 would be printed instead. The captions are printed next by positioning the cursor with MOVE commands and then printing text with the CHAR command. Then the axes themselves are plotted with calibration marks for the Y axis. Finally the Fourier synthesis of the sound waveform of a particular organ pipe is plotted.

The fourth program illustrates the capability of KGP to keep track of and display multiple windows of text and graphics. Four windows of varying size and position are set up on the screen. Three of the windows act as miniature scrolling text displays and simply display a continuous stream of characters. The fourth window displays a portion of a somewhat larger graphic image. The program to generate this display services each window in round-robbin fashion but the KGP takes care of saving and restoring the "state" of each window while another is being serviced.

The last program illustrates advanced graphics functions and rudimentary animation. First the screen is filled with a checkerboard pattern. Use of the SCFLIP command is used to generate the white squares at high speed. Next a single letter "X" is set roaming about the screen. Hitting numbers 1 through 9 on the the PET's numeric keypad will change the direction that the X wanders. Hitting "0" on the keypad stops the program. Basically the program works by drawing the character twice in "flip" mode which will cause it to first appear and then disappear. The cursor coordinates are then updated according to the wander direction and the character is drawn and erased again. Since machine language in the KGP draws the individual lines making up the character, the entire process is performed fast enough to give an illusion of continuous motion.

3.3 LOADING EXDRAW

KGPEXDRAW is a simple interactive drawing program that can be used to draw any kind of figure on the screen. Its is not really a complete drawing application however since there are no provisions for storage and later recall of the image. Nevertheless, the code illustrates some of the techniques that would be involved in a real drafting application. To load and run the program do the following:

1. Load and enable the Keyword Graphics Package as described in sections 2.3, 2.4, or 2.5. (Simply type NEW if KGPDEMO has been run).

2. Enter the command: LOAD "KGPEXDRAW", start the tape in read mode, and wait for PET BASIC to print READY.

3. Enter the command: RUN which should switch to Visible Memory video, clear the screen, and display a menu of commands at the top of the screen.

3.4 OPERATING EXDRAW

EXDRAW is a program designed to illustrate some of the capabilities of the MTU Visible Memory. EXDRAW enables its user to draw and manipulate simple drawings on the VM screen. Although it is small enough to fit in a 16K PET along with the Keyword Graphics Package, EXDRAW can do the following:

1. Use a blinking graphics cursor to indicate any point on the VM screen. The cursor can be moved in any direction and the distance of the move can be set from 1 to 512 points. In addition, the cursor can be moved to the closest already existing point, wherever it might be.

2. Draw a line between any two points.

3. Connect any point with any already existing point.

4. Move any point on the VM screen to any other position - when this is done, all of the lines connecting to the point are automatically moved as well.

5. Delete any point - when this is done, all lines connecting to the point are automatically erased.

6. Delete any line.

7. Redrawing the screen - this can be done either solidly or in flip mode.

8. The contents of the data base describing the current drawing can be displayed at any time.

3.4.1 Moving The Cursor

All drawing is done using a graphics cursor. This cursor appears on the VM screen as a blinking crosshair, and is moved by using the numeric keys (1-9). The cursor can be moved by itself whenever the message:

```
ENTER COMMAND:  . L P D E R S V : * 1-9
```

is displayed at the top of the VM screen.

The direction of the move is the direction of the numeric key from the central 5-key. For example, pressing 6 moves the cursor to the right, and pressing 1 moves the cursor down and to the left. The distance the cursor moves is initially set to 8 pixels (a pixel is a picture element, or point on the VM screen), but can be changed by pressing the "." key and then pressing a digit from 0 to 9. If the value of the digit is N, the length that the cursor will henceforth move is 2 raised to the N power. Thus if a 0 is pressed following the ".", the cursor will move one pixel with each move, while if a 6 was pressed, the cursor will move 64 pixels each move.

There are two special move control keys. If the HOME-CLR key is pressed, the cursor moves to the center of the screen (this is also the initial position of the cursor). Of the "*" key os pressed, the cursor moves to the nearest point which has already been defined. This last feature is often very useful when constructing drawings in which many lines are to end at the same point. All that needs to be done when drawing a line (described in the next section) is to move the endpoint near the desired point. When "*" is pressed, the endpoint of the line will then be at the nearest point.

3.4.2 Drawing A Line

All lines drawn by EXDRAW (except for the "S" command described later) are drawn using the flip mode. That is, whenever a line crosses a point, the state of the point is reversed. Thus, if the point was off, it is turned on, and if the point was on, it is turned off. This method of drawing has the extremely nice feature that a line can be erased simply by drawing it again, and all the points crossed by the line will be restored to their original condition.

To draw a line, press "L". The line drawn will start wherever the cursor is. Move the cursor as described in section 2. The current position of the line will always be shown on the screen. This gives the effect of a so-called "rubber-band" line. To end the line, either press "RETURN" or enter another command which will be executed. This command can be another "L" which will cause another line to be drawn starting where the previous line ended. It is possible to change the length that the cursor moves with the "." command while drawing a line.

Note that to connect two already existing points, you can use the "*" key to move the cursor to the first point, press "L" to begin the line, move the cursor near the second point, press "*" to move the endpoint to the second point, and then press "RETURN" to finish the line.

3.4.3 Moving A Point

To move a point, press "P". The cursor will then be moved to the nearest point, and the message: MOVE A POINT will be displayed. You can then move the point by moving the cursor as described in section 3.4.1. Whenever the point moves, all lines which end at the point are automatically redrawn to reflect the new position of the point. To stop moving the point, either press "RETURN" or press any other command key, which will then be executed. Note that if the point moved to upon pressing "P" is not the one you want to move, by pressing "RETURN" you can then move the cursor to the correct point without moving the point incorrectly selected.

3.4.4 Deleting A Point

To delete any point, move the cursor near the point and press "D". The cursor will be moved to the point nearest it and the message: "PRESS Y TO DELETE POINT, OTHER NOT TO" will be displayed. If the cursor is at the correct point, press "Y". The point will then be deleted along with all lines ending at the point. The screen will then be redrawn to show the modified drawing. If the cursor is not at the correct point, press any key other than "Y". EXDRAW will then enter cursor moving mode without deleting the point.

3.4.5 Deleting A Line

To delete any line, move the cursor near the line and press "E". The line nearest the cursor (perpendicular distance) will be redrawn as a dotted line (it takes several seconds to search the list of lines to find the closest one) and the message: "PRESS Y TO DELETE POINT, OTHER NOT TO" will be displayed. If the correct line is dotted, press "Y". The dotted line will be deleted. If the correct line is not dotted, press any key other than "Y" and the line will be made solid again. EXDRAW will then enter cursor moving mode.

3.4.6 Redrawing the Screen

The screen can be redrawn by pressing either "R" or "S". If "R" is pressed, the screen is cleared and the image is redrawn in flip mode. If "S" is pressed, the screen is cleared and the image redrawn solidly, that is, points common to two or more lines are turned on rather than flipped. This drawing mode is useful when you want to see how the drawing really looks without the missing points caused by intersecting lines. Note that when altering an image, it should be drawn in flip mode while when considering a "final" version of the image the drawing should be done in solid mode.

3.4.7 Displaying the List of Points

In the course of deleting lines and points, it is possible to have a point in the data base with no lines connecting to it thus wasting storage space. The "V" command can be used to plot all of the points in the data base as single points on the screen. If a lone point is seen, it may be deleted with the "D" command described in section 3.4.4.

3.4.6 Displaying the Image Data Base

To display the numeric values representing the drawing, press ":". The following information will then be displayed on the PET's standard video display:

1. The current position of the cursor as: X1 Y1 -
2. Two internally used variables and the current move length as: IP NI MF -
3. The number of points defined as: POINTS: NP -
4. The data points themselves preceeded by their index
5. The number of lines as: LINES: NL -
6. The lines themselves as defined by the indices of their endpoints

At the end of the listing will be the message: PRESS ANY KEY FOR GRAPHICS. When any key is pressed, the Visible Memory display returns.

4. FUNDAMENTAL GRAPHICS COMMANDS

Because of the large number of commands in the KGP and their wide range of sophistication, they will be described in two groups; fundamental commands, and advanced commands. In addition, they will be described in their probable order of need rather than grouped by function. The appendix has a long and short form summary of the commands grouped by function.

4.1 FUNDAMENTAL SETUP COMMANDS

Before executing plotting functions in your program, it is necessary to switch to the Visible Memory display and set the operating mode of the KGP. In many cases defaults are provided to simplify usage.

4.1.1 Video Selection Commands

The VISMEM command is used to switch to Visible Memory display so that the graphic image may be seen. The PETMEM command is used to restore the normal PET character display. One problem with software switching of the display is that an error in a graphics program (or pressing the STOP key to interrupt program execution) will usually leave the Visible Memory image on the screen. Thus the PETMEM command will have to be entered to see the PET display and error message. The PETMEM command can also be abbreviated as]P which is useful for rapidly switching to PET video in such cases.

Two other commands are provided for K-1008-6 Integrated Visible Memory users. The PVMEM command will cause both the Visible Memory and the PET display to be shown together. This is useful while debugging complex graphics programs. The NOMEM command will completely blank the screen (i.e., neither video source is selected). Note that neither the PET video "screen" nor the Visible Memory is erased, the display is merely turned off. This is useful for blanking the screen while an image is being drawn "behind the curtain". These two commands will not work with the K-1007/K-1008 combination; they would simply select Visible Memory video or PET video respectively.

4.1.2 Display Mode Commands

Three commands are provided for selecting either a black background with white plotting or a white background with black plotting. The NRMDSP command selects a black background which is considered "normal" for CRT dispalys. The RVSDSP command selects a white background. FLPDSP changes to the opposite background color. Typically the background color is specified once at the beginning of the program and then left alone. If it is changed in the middle of the program, the new background color will only apply to subsequent plotting commands; it will not change the background color of what is already on the screen. Thus one would normally clear the screen following a background color change. The default is a normal display (black background) which is easier on the eyes and on the PET's picture tube.

4.1.3 Screen Clearing Commands

Two high speed screen clear commands are provided. A simple CLEAR will set the entire Visible Memory screen to the current background color in about 1/10 of a second. SCLEAR can be used to clear rectangular portions of the screen if desired. Following the SCLEAR command should be four numbers (or BASIC variables) which define the X and Y coordinates of the upper left and lower right corner of the rectangle to be cleared.

The order of the arguments is Xul,Yul,Xlr,Ylr. The X coordinates are expected to be in the range of 0 to 319 and the Y coordinates are expected to be between 0 and 199. If either Y has a fractional part, it will be truncated to an integer. X is a bit more complicated however. The first X, which defines the left edge of the cleared area, is reduced by the KGP until it is divisible by 8. The second X, which defines the right edge, is increased until it is one less than a value divisible by 8. Thus if the X coordinates were 66 (left) and 163 (right), the 66 would be reduced to 64 (the closest lower value divisible by 8) and the 163 would be increased to 167 (one less than 168 which is divisible by 8). This action will be referred to as "rounding to a multiple of 8" in the description of other commands that act this way. To avoid unexpected results it is a good idea to always make the left X coordinate a multiple of 8 and the right X coordinate always 1 less than a multiple of 8 in commands that do such rounding.

4.1.4 Plotting Mode Commands

Even with only two colors possible on the screen, there are three ways that points, lines, and characters can be plotted. The command GMODE followed by a single integer or variable selects one of these three modes. Mode 1 is used for most normal plotting. In mode 1, all plotted points are set to the opposite color of the background. Thus for a black background, white points, lines, and characters are plotted. In mode 2, all plotted points are the same color as the background, i.e., nothing shows up. However if an image previously plotted in mode 1 is replotted in mode 2, it is erased. The replotting must be exact for the erasure to be complete.

Mode 0 is a special case. In mode 0, the points plotted are made the opposite of their previous color rather than opposite the background color. The effect is the same as mode 1 if an isolated point or line is drawn on an unblemished background. However if a line crosses a previously plotted line, their point of intersection will revert back to the background color. If two identical lines are drawn in mode 0, the second line will effectively cancel out the first one and leave nothing but the background! The drawings below illustrate the effect of different drawing modes on a figure:

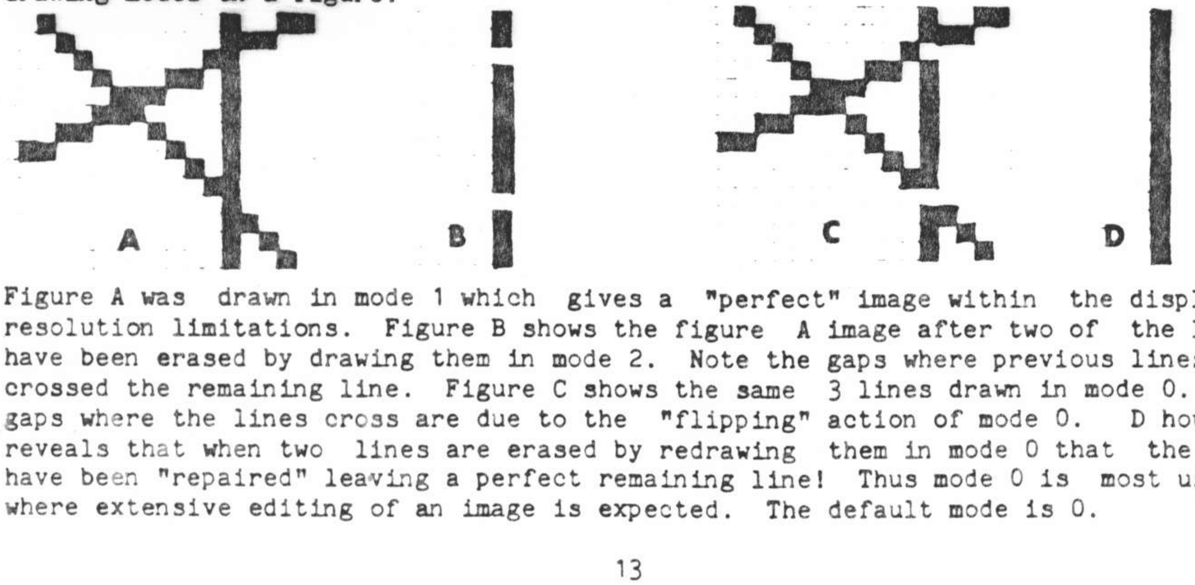


Figure A was drawn in mode 1 which gives a "perfect" image within the display's resolution limitations. Figure B shows the figure A image after two of the lines have been erased by drawing them in mode 2. Note the gaps where previous lines had crossed the remaining line. Figure C shows the same 3 lines drawn in mode 0. The gaps where the lines cross are due to the "flipping" action of mode 0. D however reveals that when two lines are erased by redrawing them in mode 0 that the gaps have been "repaired" leaving a perfect remaining line! Thus mode 0 is most useful where extensive editing of an image is expected. The default mode is 0.

4.2 PLOTTING COMMANDS

The KGP is capable of directly plotting points and drawing straight lines. The location of the points and lines on the screen are defined by X and Y coordinates. X has a legal range of 0 to 319 inclusive and Y has a range of 0 to 199 inclusive. If mixed numbers are given for coordinates, they are truncated before use (same effect as the BASIC INT function). The handling of out of range coordinates is described in the individual commands.

4.2.1 Positioning the Drawing Cursor

In many of the plotting commands the location of a "drawing cursor" is important. The MOVE command followed by X,Y coordinates is used to set the position of this cursor. The drawing cursor is simply a pair of numbers kept internally to the KGP, no real cursor shows up on the VM screen. Coordinates in the range of -32767 to +32767 are acceptable to the MOVE command. If they are outside of that range, an "ILLEGAL QUANTITY" error will occur.

4.2.2 Plotting Points

To plot a point at the current location of the drawing cursor, simply enter the command, WRPIX. The command name is mnemonic for "WRite PIXel". The actual effect of writing the pixel is dependent on the current background color (see sect. 4.1.2) and the current graphics mode (see sect. 4.1.4). With a normal black background and the graphics mode set to 1, WRPIX will unconditionally plot a white point. If the drawing cursor specifies a point location outside the 0-319,0-199 screen area, point plotting is skipped.

4.2.3 Plotting Lines

Two commands are provided for plotting lines between points. The LINE command will plot a line between any pair of endpoints without having to set the drawing cursor first. The format of the command is: LINE X_1,Y_1,X_2,Y_2 where X_1,Y_1 defines the location of the initial endpoint, and X_2,Y_2 defines the final endpoint. Note that when the command is completed, the drawing cursor will be set to X_2,Y_2. If part of the line is outside of the 0-319,0-199 screen area, only the visible part will be plotted. Note that the invisible part still requires plotting time even though no plotting is done. Thus a command like LINE -10000,-10000,10000,10000 will require a few seconds to plot even though only 200 of the 20000 points are actually visible.

The DRAW command works somewhat differently. It is followed by a single pair of coordinates, X_2,Y_2, and will draw a line from the drawing cursor position to the X_2,Y_2 point. When the drawing is complete, the drawing cursor is repositioned at the X_2,Y_2 point. This command is useful for drawing figures made of end-to-end connected lines. It is particularly useful for graphs where the plotted points are to be connected by straight lines for improved appearance. Again, only the visible portion of the line will be plotted.

4.3 CHARACTER DRAWING COMMANDS

The commands discussed up to this point are sufficient for plotting any kind of graphic image. However it is often desirable to caption and label the image with text characters. Only the simplest form of character plotting will be discussed here, i.e., normal character size and left-to-right reading. The standard character set contains definitions for all of the upper and lower case letters, the digits, and standard ASCII special characters. It does not have the PET graphics characters. The shapes of most of the characters are slightly different from the corresponding PET video characters since they are drawn in a 5 dot by 7 dot matrix rather than a 7 dot by 7 dot matrix. This allows one to draw as many as 53 characters on a line with equal or even superior readibility.

The CHAR command is used for displaying short pieces (one line or less) of text. It may be followed by a single numerical argument or variable in which case it will interpret the number as an ASCII code and draw the corresponding single character. The character is located such that the leftmost point of its baseline coincides with the current drawing cursor position. The character extends 7 coordinate units above the baseline and 5 units to the right of the X cursor position. Note that lower case characters with descenders (g,j,p,q,y) draw up to three units below their baselines. After the character is drawn, the X coordinate of the drawing cursor is incremented by 6 in preparation for another character. Thus the statement: CHAR 64 will plot a capital A at the cursor position and then increment the cursor position by 6 in the +X direction.

CHAR may also be followed by a BASIC string variable in which case the entire string will be printed. Movement from one character to the next is as described above. When the string has been drawn, the drawing cursor is positioned to the next available character space. One may also use multiple arguments with the CHAR command. Each argument is separated from the preceeding one by a semicolon (;) just as in BASIC print statements. (The comma (,) separator for tabbing to the next field is not available.) You may also mix numeric arguments and string variable arguments. Note that if the line of text becomes too long, it will simply run off the screen and the excess characters will not be seen.

Characters are actually drawn with line segments rather than by moving a 5x10 dot matrix into the display area. This means that the characters are plotted according to the same rules as lines with respect to the background color and the plotting mode. In particular, drawing a new character on top of an old one will not erase the old character, instead the two characters will be superimposed. This also applies to the space (code 32) character which in effect simply moves the cursor without erasing. The old character must be erased first by drawing it again with the graphics mode set to 2 or 0. Alternatively, a delete (code 20) may be printed which will erase the character cell (including descender) but it will be relatively slow. Note that the delete code does not move the cursor. A text type-in and display program therefore should detect blanks and replace them with a delete-blank sequence.

5. ADVANCED GRAPHICS COMMANDS

While the previously discussed commands are sufficient for most any kind of graphics application, many additional commands and modes are available to simplify the more complex applications. These were not discussed in the previous section to avoid confusing first time users.

5.1 SCALING AND OFFSETS

In most plotting applications the X and Y coordinates will not "naturally" be in the 0-319,0-199 range of the display. Of course the user program can always transform whatever coordinate representation is desired into the proper range but then plotting is slowed substantially by the additional BASIC language programming. To alleviate this, the KGP has the ability to transform all of the coordinates it receives at machine language speed.

5.1.1 Offset Command

Normally the origin of the coordinate system for plotting (i.e., the 0,0 point) is at the lower left corner of the screen. By using the offset command, the origin may be set anywhere desired. OFFSET followed by two numbers or variables separated by a comma will establish a value that will be added to all X coordinates and a value that will be added to all Y coordinates before they are used. Thus if the statement: OFFSET 160,100 is executed (and offsetting is enabled, see below), then the origin of the coordinate system will be at the center of the Visible Memory screen. Offsets may be anything in the range of -32767 to +32767 however any fractional part is truncated off. The default offsets are 0,0. Note that the drawing cursor position will become undefined immediately after execution of the OFFSET command.

5.1.2 Scaling Command

In addition to moving the origin, it is useful to be able to shrink or expand the image, that is, scale it. For maximum speed and to simplify the machine language coding in the KGP, scaling can be only by positive or negative powers of 2. In fact the scaling command expects the actual power of 2 to be specified rather than a multiplying factor. Thus for scaling up by a factor of 8 (i.e., coordinates are multiplied by 8), the scaling parameter would be 3 since $2^3=8$. For scaling down, the scaling parameter would be negative; to shrink by a factor of 8, -3 would be specified since $2^{-3}=1/8$. The SCALE command is used to establish the X and Y scale factors as a pair of scale factors separated by a comma following the command. As an example, the statement: 130 SCALE 2,-1 will expand the image by a factor of 4 in the X dimension and shrink the image by a factor of 2 in the Y dimension. The default value for both scale factors is 0 which gives no scaling ($2^0=1$). Note that the drawing cursor position will become undefined immediately after execution of the SCALE command.

When studying the effect of combined scaling and offsetting, it is important to realize that scaling is done before offsetting. In addition, all arithmetic is integer arithmetic. Thus the equations that are actually evaluated by the KGP when coordinate transformation is enabled are as follows:

```
Xdrawn=INT(Xoffset)+INT(INT(Xspecified)*2↑INT(Xscale))

Ydrawn=INT(Yoffset)+INT(INT(Yspecified)*2↑INT(Yscale))
```

5.1.3 Enabling Coordinate Transformation

After setting the offset and scaling parameters, it is necessary to enable coordinate transformation. The command XFFLG followed by a single number establishes the transformation mode. Mode 0 is no transformation and is the default. Mode 1 enables transformation. Note that transformation only applies to subsequent drawing; the current content of the screen is not transformed. Also note that the drawing cursor position becomes undefined when the coordinate transformation mode is changed.

5.1.4 Character Scaling and Rotation

Characters can also be scaled and rotated independently of graphic scaling. The CHSCALE command followed by a single number sets the scale factor for characters. The scaling parameter is interpreted in the same way as the graphics scaling parameter, that is, it is an integer power of 2. Generally only positive values are useful since so much of the detail of the character will be lost when negative values are used that text becomes unreadable. The automatic cursor movement associated with the CHAR command is also affected by the character scale factor so that proper character and line spacing is maintained. The default character scale parameter is 0 which gives normal sized characters.

Characters may also be rotated in 90 degree increments by use of the CHROT command. The single following argument must be in the range of 0 to 3. The amount of counter-clockwise rotation is equal to 90 degrees times the rotation parameter. Note that CHAR will update the cursor position correctly for rotations other than 0. The default rotation parameter is 0 which gives normal upright, right-reading characters.

Note that character scaling and rotation is always in effect, it need not be enabled before use.

5.2 DISPLAY WINDOWS AND BOUNDARY CHECKING

The KGP has the capability of maintaining up to 4 independent "viewing windows" which is useful for implementing "split screen" functions in application programs. Again, appropriate BASIC programming could perform the same function but the KGP is able to switch among the windows much faster; fast enough in fact to give the illusion of simultaneous action in each window.

A "window" is really nothing more than a set of 6 numbers stored in memory. Associated with each window is its current boundaries (4 numbers) and the position of its drawing cursor (2 numbers). When a different window is selected as the "current window", these 6 numbers are saved in the old window's memory and the 6 numbers for the new window are made current. Thus a user program can freely switch among the 4 windows without having to save or restore any data. (Note however that character scaling and rotation parameters are global and not saved with the windows.)

5.2.1 Window Definition

When the KGP is first loaded and enabled, all 4 windows are initialized with boundaries set to the screen boundaries (0,0,319,199) and the cursor position undefined. In addition, window 0 is selected for use. The WINDOW command followed by a single argument is used to select a window and make it current. Data about the previously selected window is saved. The argument must be in the range of 0 to 3 since memory space for only 4 windows is available.

Window boundaries may be set with the SETWIN command. SETWIN is followed by 5 numbers which specify the window number (0 to 3), the left boundary, the bottom boundary, the right boundary, and the top boundary respectively. Note that the boundary values are with respect to the untransformed screen coordinates, i.e., the left and right boundaries must be between 0 and 319 and the bottom and top boundaries must be between 0 and 199. If the specified boundaries are out of range or scrambled up, they will be forced to something that makes sense but the final result will be unpredictable. Note that if SETWIN refers to the currently active window that the new boundaries will become the current boundaries. This allows software simulation of more than 4 windows if needed. Setting the boundaries of a window makes the cursor position for that window undefined.

Some operations on windows will "round" the left and right boundary values to a multiple of 8 as described in section 4.1.3. Thus it is a good idea to always specify a left boundary that is a multiple of 8 and a right boundary that is one less than a multiple of 8.

5.2.2 Boundary Checking

As a default, the KGP checks the coordinates of every point plotted against the boundaries of the current window before actually plotting. If the point would be outside the boundaries, the point is not plotted. This applies to points plotted by the WRPIX command, the line drawing commands, and the character drawing commands. Obviously this constant checking slows down plotting substantially even though it is done in machine language. Plotting speed may be increased by giving the NOCHK command to turn off boundary checking. The BNDCHK command restores boundary checking.

When boundary checking is turned off, points, lines, and characters being plotted are not specifically checked against the window boundaries. They are checked sufficiently so that memory addresses outside of the visible area of the Visible Memory are not written into, thus protecting memory. The actual visible effect of plotting outside of the screen area when boundary checking is off is unpredictable.

5.3 CURSOR DISPLAY COMMANDS

Although a cursor is used internally to define the location of points, line endpoints, and characters, it does not show up on the screen. For interactive graphics programs, it is useful to have a display of the current cursor position. The GRACSR command can be used to display a crosshair cursor at the current drawing cursor position. The cursor is always drawn in "flip" mode so that it will show up regardless of the type of image, if any, it covers. Subsequent movment of the drawing cursor will not cause the displayed cursor to move however. Instead, the visible cursor must first be erased by executing the GRACSR command again with the old cursor coordinates, updating the drawing cursor coordinates, and then issuing another GRACSR to display the crosshairs at the new location.

For text entry applications, an underline cursor is probably preferred to point to where the next character will appear. The TEXCSR command will display an underline cursor at the drawing cursor position. Actually, the Y coordinate of the cursor will be one less than the cursor position. The left end of the underline is equal to the X coordinate of the cursor while the length to the right is 5 units times the current character scale factor. Like the crosshair graphic cursor, the KGP does not automatically update the displayed cursor. Thus before a character is actually displayed, the cursor should be erased by displaying it again (it is always drawn in flip mode), the character drawn which will move the drawing cursor, and then the new cursor position displayed with the TEXCSR command.

During debugging or simulating more than 4 windows, it may be helpful to be able to read what the current drawing cursor position is. The RDCSR command followed by two variable names will read the X and Y coordinates of the cursor position into those variables. Thus the statement: 235 RDCSR EX,WY will set the value of EX to the X coordinate of the cursor and set WY to the Y coordinate of the cursor. Note that the values returned have already been transformed if transformation is enabled.

5.4 ADVANCED TEXT DISPLAY FUNCTIONS

While the CHAR command is useful for printing short labels and captions on figures, it is awkward to use when multiple lines of formatted text are to be displayed. The AUTEXT command is similar to CHAR except that it handles end-of-line and end-of-page conditions just like the PET's own character display. Thus after plotting a character and moving the drawing cursor, a check is made to determine if the next character will go beyond the current right boundary. If so, the X coordinate of the cursor is set to the left boundary (plus 2 units for spacing) and the Y coordinate is decremented by 10 so that a new line of text is started. If when Y was decremented it comes within 3 units of the bottom boundary, the entire content of the current window, text, graphics and all, is moved up 10 coordinate units instead, that is, scrolled. These actions effectively make the Visible Memory into a scrolling text display with a screen capacity of 20 lines of 53 characters each. If the current character scale factor is other than zero, the numerical values listed above are suitably altered.

Besides the normal printable characters, both CHAR and AUTEXT recognize and interpret a number of special control characters. These along with their function are listed below:

PET KEY	ASCII CODE	DEFINITION
DEL	20	Erase the character at the current character position. (does not move the cursor)
Up-arrow	145	Move cursor up one text line.
Down-arrow	17	Move cursor down one text line.
Right-arrow	29	Move cursor right one character position.
Left-arrow	157	Move cursor left one character position.
Shift-1 - Shift-9	177- 185	Move one dot position in direction the digit is from 5 on the PET numeric keypad.

PET KEY	ASCII CODE	DEFINITION
Shift-D	196	Add one to CHSCALE, i.e. double the character size.
Shift-E	197	Subtract one from CHSCALE, i.e. halve the character size.
Shift-F	198	Add one to CHROT, i.e. rotate 90 degrees counterclockwise
Shift-G	199	Subtract one from CHROT, i.e. rotate 90 degrees clockwise
Shift-@	192	Try it!
Shift-$	164	Draw the contents of BASIC's string variable GR$ as a character definition. The definition contained in GR$ MUST end in a zero byte! Used for debugging user character shape definitions. See section 6.
CLR	147	AUTEXT only. Clears the current window and then sets the cursor for the next character to be in the upper left corner of the window.
HOME	19	AUTEXT only. Positions cursor as in CLR but does not clear the window.
RETURN	13	AUTEXT only. Sets the cursor to the left boundary and moves down one text line (or scrolls up if necessary).

5.5 MISCELLANEOUS ADVANCED COMMANDS

Numerous additional commands are provided that are useful for special functions, debugging, etc. The commands used for defining custom character sets are not included here, they are described in section 6.

5.5.1 Readback Commands

Commands are provided for reading the current state of several KGP parameters. These commands are most useful for debugging. RDGM followed by a single variable name will read the current graphics mode, either 0, 1, or 2, (see section 4.1.4) into the variable. RDPIX followed by a variable name will read the state of the Visible Memory bit under the cursor into the variable. The value returned will be 0 if the point is black, 1 if it is white, or 255 if it is outside the boundaries. This command in conjunction with the WRPIX command can be used to turn the Visible Memory into a 64,000 bit memory with individual bit addressing for special applications.

5.5.2 Dotted Line Command

In some cases it may be desirable to make the lines connecting endpoints dotted rather than solid. The DOTL command can be used to accomplish this easily. DOTL is followed by two arguments separated by commas. The first argument specifies the number of coordinate units the line is to be "on" (the dash length) while the second argument specifies the number of coordinate units the line is to be off (the space length). If the second argument is zero, solid lines are produced. The default of course is solid lines. Lines making up characters are always solid.

5.5.3 Area Plotting Commands

Three commands are provided for rapid area plotting functions. WCLEAR will clear the area defined by the current window's boundaries to the background color. In addition, it moves the drawing cursor to a position appropriate for plotting a character in the upper left corner of the window. Also if boundary checking had been turned off previously with the NOCHK command, it will be turned back on. Note that the left and right boundaries are "rounded to a multiple of 8" for clearing purposes (see section 4.1.3).

SCFLIP is used to "flip" the state of all of the display dots in a specified area of the screen. Thus white dots become black and vice-versa. Following the SCFLIP command should be four numbers (or BASIC variables) which define the X and Y coordinates of the upper left and lower right corner of the rectangle to be cleared. The order of the arguments is Xul,Yul,Xlr,Ylr. The X coordinates are expected to be in the range of 0 to 319 and the Y coordinates are expected to be between 0 and 199. If either Y has a fractional part, it will be truncated to an integer. The X values will be "rounded to a multiple of 8" (see section 4.1.3).

The most powerful area command is SCROLL which can be used to move entire areas of the screen from one position on the screen to another. The entire command format is: SCROLL Xt,Yt,Xul,Yul,Xlr,Ylr where each of the 6 arguments may be either numbers or variables. The basic function is to move the rectangle defined by Xul,Yul,Xlr,Ylr (upper left and lower right corner coordinates) to an area of the same size whose upper left corner is at Xt,Yt. Please note that the three X coordinates are "rounded to a multiple of 8" (see section 4.1.3). The move is destructive, that is, the source rectangle is set to the background color as its content is moved to the destination rectangle. The source and destination rectangles can also overlap in any manner desired and SCROLL will still work as expected. Xt and Yt may specify part (or all) of the image to be moved off the screen and it will function as expected.

5.5.4 Miscellaneous KGP Control Commands

The VMPAGE command establishes the address of the Visible Memory graphics device. The single argument should be the memory page number of the first byte in the VM. Normally, this command would be executed immediately after loading the KGP if the VM was in a non-standard location. However it can also be useful in custom systems with two or more VM's to switch from one "image plane" to another.

The GKILL command is used to disassociate KGP from BASIC. It must be executed if KGP is no longer needed and the user wishes to use the memory occupied by KGP for other purposes.

The most frequently used commands have a short abbreviated name (see section 7.1) in addition to their completely spelled out name. KGP can be set into a mode where only the abbreviated form is recognized by executing the GRSHRT command. The advantage of the abbreviated only mode is that searching for command names is faster thus any BASIC coding in the program will run about 25% faster under the abbreviated name mode. The]X command will restore full name mode. If the execution of a particular non-graphic routine dominates the run time of a program (such as a Fourier analysis subroutine), one can simply code a GRSHRT command at the beginning of the routine and a]X command at the end to speed up that particular routine without having to sacrifice readability of other portions of the program.

6. USER DEFINED CHARACTERS AND FIGURES

The Keyword Graphics Package has the ability to store up to 255 predefined "shapes" each of which may be recalled by giving its "ID" and drawn anywhere on the screen (according to the drawing cursor) at machine language speed. In fact the "characters" available through the CHAR and AUTEXT commands are nothing more than 94 of these stored shapes where the numeric value of each ASCII character addresses the appropriate stored shape. The advantages of using the shape table facility of the KGP to draw repetitive shapes are:

1. Compact storage of the shape data - as little as 1 byte per line segment.
2. All shape coordinates are <u>relative</u> meaning that the same shape can be drawn anywhere on the screen by specifying the location of the first point.
3. Intricate shapes are drawn at machine language speed, typically 10 to 100 times faster than doing the same thing in BASIC.

Normally, building up definitions for the stored shapes would be a complex process involving use of the machine language monitor but in the KGP several additional BASIC commands have been provided so that shape tables can be defined and redefined by a BASIC program.

6.1 SHAPE TABLE STRUCTURE

In order to understand how the shape table building commands are used, it is first necessary to become familiar with how the shape table itself is stored in memory. A shape table entry consists of a string of bytes beginning with its ID and ending with a zero byte as illustrated below:

$$\underset{\nearrow\ \text{ID byte}}{65},166,132,230,51,65,132,1,99,\underset{\nwarrow\ \text{Terminal 0}}{0}$$

In most cases there will be many different shape entries in the shape table, each one following the previous one as shown below:

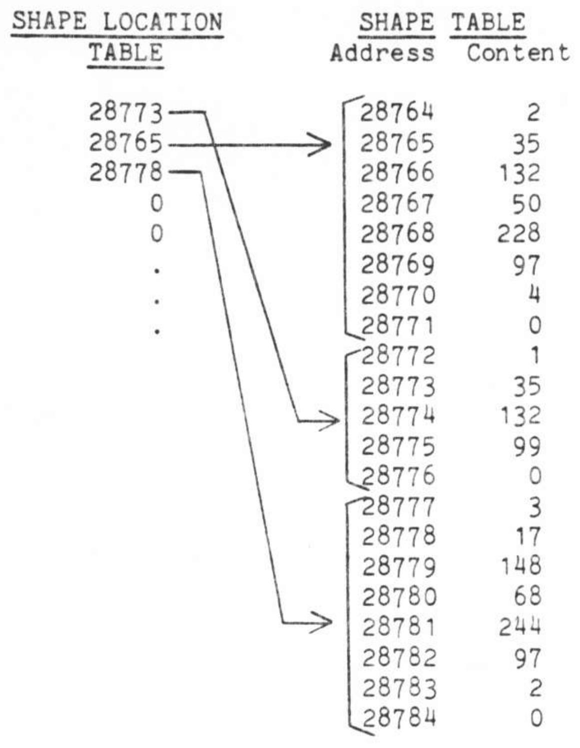
$$\underbrace{65,166,132,230,51,65,132,1,99,0}_{\text{Shape \# 65}},\underbrace{203,177,193,209,241,129,145,0}_{\text{Shape \# 203}},\underbrace{201,165,15,140,\ldots}_{\text{Shape \# 201}}$$

Since zero is not a legal drawing instruction byte in the shape table entry, it is easy to separate the entries simply by looking for zero instruction bytes.

When a shape is actually being drawn, the <u>shape interpreter</u> routine built into the KGP (i.e., the routine that executes CHAR and AUTEXT commands) receives the ID of the shape to be drawn and then "searches" the shape table for the matching ID. When a match is found, the drawing instruction bytes are interpreted one-by-one until the zero byte is found which terminates the shape. The drawing instruction bytes are described in section 6.2. If no match is found, nothing is drawn.

In reality it would be very slow even in machine language to search through hundreds of bytes of shape table entries looking for the requested ID. In order to speed up the search process, a seperate <u>shape location table</u> is used. The shape location table consists of 256 <u>pointers</u> each of which points to a shape table entry. Since each pointer is two bytes long, the shape location table is always 512 bytes long. Now when a CHAR or AUTEXT command wants to draw a shape, it simply goes directly to the IDth entry in the shape location table and then follows the pointer directly to the first instruction byte in the corresponding shape table entry. If an ID is requested for which there is no corresponding shape table entry, the pointer in the shape location table will be zero and no drawing will be performed.

The diagram to the right shows a shape table with only three simple shapes defined and the corresponding shape location table. Using this as an example, let's assume that a CHAR 2 command was executed. KGP would look into the 2nd entry in the shape location table and find a pointer to address 28765. At 28765 is the first drawing instruction byte for a simple "+" shape. (Note that the ID byte for the shape is still in the shape table. This is "left over" from building the shape location table and is not used in shape drawing.) Each instruction byte would be executed in turn to draw the "+" until the zero byte was encountered. At this point the CHAR command would be complete and execution of the user's BASIC program would continue with the next line. If a CHAR 5 was executed, KGP would look at the 5th entry in the shape location table and find that the pointer was zero. Since a zero pointer means that no shape is defined for that ID, nothing is drawn and the CHAR command would be complete.



SHAPE LOCATION TABLE	SHAPE TABLE Address	Content
28773	28764	2
28765	28765	35
28778	28766	132
0	28767	50
0	28768	228
.	28769	97
.	28770	4
.	28771	0
	28772	1
	28773	35
	28774	132
	28775	99
	28776	0
	28777	3
	28778	17
	28779	148
	28780	68
	28781	244
	28782	97
	28783	2
	28784	0

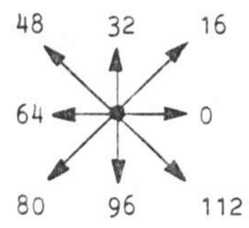
6.2 SHAPE INSTRUCTION BYTES

The shape instruction bytes tell the shape table interpreter how to draw the shape. The drawing cursor, which specifies the location of the shape, is used and updated in all shape drawing.

6.2.1 Vector Byte Instruction

For small, relatively simple shapes such as characters, vector byte instructions are the most efficient. Within a single byte one can generate a line segment or perform a move without drawing in any of 8 directions a distance from 0 to 14 coordinate units. The drawing below illustrates how to compute the vector byte value given the direction, the segment length, and whether a draw or move without draw is desired.

DIRECTION NUMBER



LENGTH NUMBER

Specifies length of move or draw. This number may be from 0 to 14.

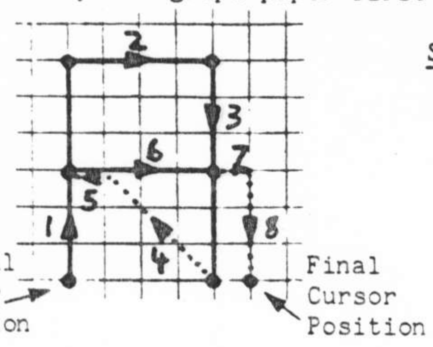
MOVE/DRAW NUMBER

Is zero for move.
Is 128 for draw.

The vector byte is simply the sum of the three numbers calculated above. Note that a move of zero distance to the right will be interpreted as an end-of-shape instruction.

As an example of how to code a shape definition using vector bytes let's try to code the shape definition for a squared-off letter "A". The first byte in the shape definition should be the ID number we wish to have refer to the "A". A logical choice would be the ASCII code for upper case A which is 65 in decimal.

Before trying to calculate the vector byte values, it is a good idea to draw out the shape on graph paper first as is done below:



SEGMENT	DIRECTION	LENGTH	MOVE/DRAW	SUM
1	32	6	128	166
2	0	4	128	132
3	96	6	128	230
4	48	3	0	51
5	64	1	0	65
6	0	4	128	132
7	0	1	0	1
8	96	3	0	99
End	0	0	0	0

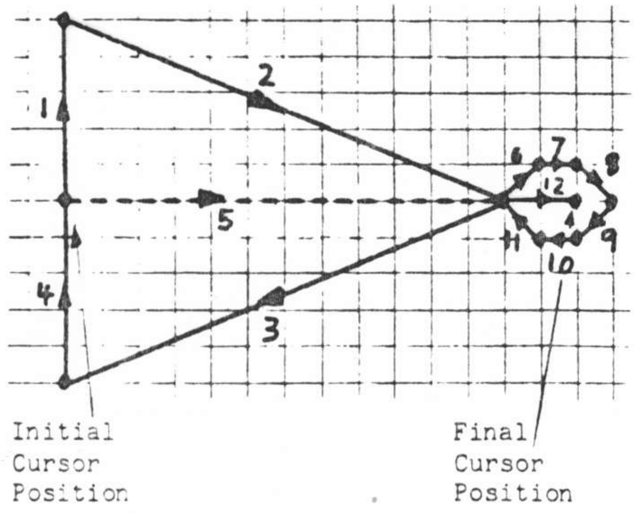
Note that an arrow is drawn over each line segment to indicate the direction of drawing and then each segment is given a number in order to avoid confusion. Next the direction number, length number, and move/draw number is computed for each segment. The actual sequence of vector bytes is the sum of the three numbers for each segment. Following the vector byte for the last segment is a zero byte to mark the end of the definition.

Note that a pair of moves is performed after the shape itself is drawn so that the cursor will be in position for the following character. If shapes are being defined for purposes other than text display, it probably will not matter where the cursor is left after drawing the shape. As an exercise, see if you can code the shape with fewer than 8 segments while maintaining the same final cursor position.

6.2.2 Relative Move and Draw Instruction Bytes

For larger shapes or shapes in which segments at odd angles are needed, relative move and relative draw instructions are available. Each of these are 3 bytes long. The first byte is a 223 for relative draw or 239 for relative move. The second byte is the X distance for the move/draw and the third byte is the Y distance. To allow moving or drawing in either direction, the number coded is the sum of the actual distance desired (within a range of -128 to +127) and 128. Thus if the cursor is to be moved 27 units to the left and 33 units upward, the entire instruction would be: 239,101,161 where the 239 is the relative move instruction code, the 101 is 128-27, and the 161 is 128+33.

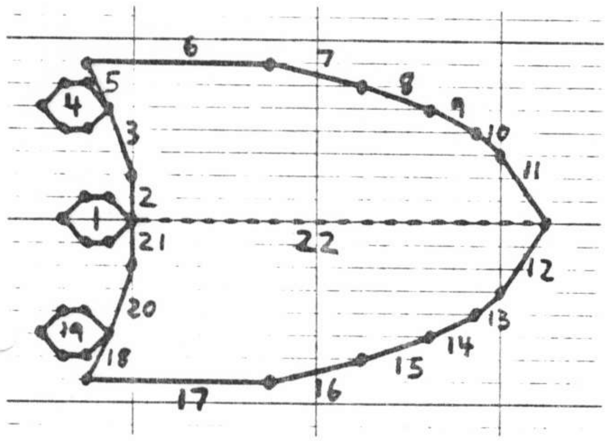
Note that a shape table entry can be any mixture of vector byte instructions and relative move/draw instructions that may be appropriate. The example below shows how one might define a logic gate symbol with such a combination:



	----VECTOR	BYTE----			-RELATIVE	M/D-	
SEG	DIR	LGTH	M/D	SUM	INS	X	Y
1	32	5	128	165			
2					223	140	123
3					223	116	123
4	32	5	128	165			
5	0	12	0	12			
6	16	1	128	145			
7	0	1	128	129			
8	112	1	128	241			
9	80	1	128	209			
10	64	1	128	193			
11	48	1	128	177			
12	0	2	128	130			

6.2.3 Additional Instruction Byte Codes

Several additional instructions are recognized in a shape definition. Perhaps the most useful is the 47 instruction which allows the definition of another shape to be used as part of this shape definition. This is somewhat like a "graphics subroutine" and is most useful for shapes that have two or more identically shaped "appendages". Following the 47 instruction byte is the ID of the shape that is to be used as a "subshape". When using subshapes, the cursor position at the end of a subshape is crucial since the remainder of the "main" shape is influenced by it. A good convention to follow when defining shapes that will be used as subshapes is to make the final cursor position the same as the initial cursor position. Subshapes may be nested, that is, a subshape definition may itself call upon another subshape (which makes it a sub-subshape) ad infinitum. (The maximum nesting depth is determined by the length of the 6502 micropreccessor's stack. A limit of 6 levels is reasonable.) The example shape definition below shows how subshapes can be used to shorten the definition of a logic OR-not gate symbol:



MAIN SHAPE DEFINITION

Seg	Vector Byte Sum	Rel move/draw Ins	X	Y	Subshape Ins	ID
1					47	203
2	162					
3		223	127	131		
4					47	203
5		223	127	130		
6	136					
7		223	132	127		
8		223	131	127		
9		223	130	127		
10	241					
11		223	130	125		
12		223	126	125		
13	209					
14		223	126	127		
15		223	125	127		
16		223	124	127		
17	200					
18		223	129	130		
19					47	203
20		223	131	129		
21	162					
22		239	146	128		
		0				

INVERSION BUBBLE DEFINITION

203,177,193,209,241,129,145,0

A similar instruction is the 63 instruction. Following the 63 is a string of shape ID's terminated by a zero ID. This is most useful in saving space when defining a shape that is actually a sequence of shapes (such as the letters of a company logo). Don't forget the zero instruction byte to terminate the definition; the zero ID only terminates the 63 instruction.

Two instructions make it possible for a shape definition to change the character (shape) scaling and rotation parameters. The 95 instruction will cause the next byte to be picked up, 128 subtracted from it, and the result added to the current character rotation parameter. The 175 instruction will cause the next byte to be picked up, 128 subtracted from it, and the result added to the current character scale parameter. There are probably very few practical uses for these commands but they are available.

The last two instructions are useful for debugging shape definitions of for maintaining "floating definitions" that are never really integrated into the KGP. The 79 instruction will cause the contents of a BASIC string variable to be used as a subshape definition. The two bytes following the 79 should be the PET ASCII code of the string variable's name. Remember that the subshape definition in the string variable must end with a zero byte. The 191 instruction is similar except that the actual memory address of the subshape definition (low byte first) follows the 191.

6.3 STORING SHAPE TABLE ENTRIES IN MEMORY

Now that we know what the byte values are for defining a shape, it is necessary to get them stored in a form that the KGP can use. In most cases the user will wish to add to the shape definitions already built into KGP so that the character plotting functions remain available. There are 298 bytes unused in the predefined KGP shape table that can be used for user shape definitions. This is enough for several dozen simple shapes or all of the examples given so far.

Three steps are necessary to add a shape definition to the shape table. The first step is to execute a CHINIT command followed by the number you wish to use as the ID of the new shape. The ID must be between 1 and 255 inclusive and should be an ID that is not already assigned to a character shape (if it is, that character shape will be effectively redefined). Currently unused ID's are: 1-10, 12, 14-16, 18, 21-28, 30-31, 124, 126-144, 146, 148-156, 158-163, 165-170, 172-176, 186-191, 200-239, 243-255. The second step is to enter a loop in which each byte in the shape definition is given as the argument of a CHDFC command. Thus if the shape requires 20 bytes (i.e., all vector bytes, relative move/draws, other instruction bytes, and the final zero) then the CHDFC command must be executed 20 times, once for each byte. The last step is to get the new definition added to the shape pointer table. This is accomplished by executing a CHBLD command which requires no arguments. Additional shapes may be added by following the preceeding three steps for each shape to be added.

As an example, lets add each of the 4 example shapes that have been discussed to the shape table. The BASIC program segment below will accomplish this:

```
100 DATA 166,132,230,51,65,132,1,99,0: REM FUNNY A
110 DATA 165,223,140,123,223,116,123,165: REM LOGIC INVERTER
111 DATA     12,145,129,241,209,193,177
112 DATA     130,0
120 DATA 47,203,162,223,127,131,47,203: REM OR-NOT GATE
121 DATA     223,127,130,136,223.132,127
122 DATA     223,131,127,223,130,127,241
123 DATA     223,130,125,223,126,125,209
124 DATA     223,126,127,223,125,127,223
125 DATA     124,127,200,223,129,130,47
126 DATA     203,223,129,131,162,239,146
127 DATA     128,0
130 DATA 177,193,209,241,129,145,0: REM INVERSION BUBBLE
200 CHINIT 200: REM DEFINE THE FUNNY A
210 FOR I=1 TO 9: READ A: CHDFC A: NEXT I
220 CHBLD
300 CHINIT 201: REM DEFINE THE LOGIC INVERTER
310 FOR I=1 TO 17: READ A: CHDFC A: NEXT I
320 CHBLD
400 CHINIT 202: REM DEFINE THE OR-NOT GATE
410 FOR I=1 TO 52: READ A: CHDFC A: NEXT I
420 CHBLD
500 CHINIT 203: REM DEFINE THE BUBBLE FOR THE OR-NOT GATE
510 FOR I=1 TO 7: READ A: CHDFC A: NEXT I
520 CHBLD
```

The following program segment will then use these definitions as part of a graphic image:

```
1000 CLEAR: VISMEM: GMODE 1: REM SETUP FOR DRAWING
1010 CHSCALE 1: REM DRAW LOGIC GATES TWICE NORMAL SIZE
1020 LINE 0,124,10,124: CHAR 201: DRAW 56,124: DRAW 56,154: DRAW 72,154
1030 LINE 56,199,56,174: DRAW 72,174
1040 LINE 0,164,10,164: CHAR 201: DRAW 74,164
1050 CHAR 202: DRAW 136,164: DRAW 136,140
1100 CHSCALE 0: REM DRAW LETTERS AT SMALLEST READABLE SIZE
1110 FOR I=156 TO 176 STEP 10
1120 MOVE 60,I: CHAR 200
1130 NEXT I
```

When debugging shape definitions it is important to realize that every time a definition is added to the shape table that an internal memory pointer is updated. Thus if a definition is added, checked, found to be in error, changed, and then added again, the old definition remains in memory taking up space. After enough iterations all of the memory available for shape table additions will be used up and error messages will begin to appear. To overcome this problem, the SYS 256*34, SYS 256*66, or SYS 256*98 used to initialize KGP should be done again to reset the internal pointer and thus write over all of the previous user specified definitions.

6.3.1 Adding Shape Table Entries In the User's Own Memory

For some applications, the 298 bytes available in KGP memory will not be sufficient to hold all of the needed extra shape table entries. In this case it is possible to "steal" additional memory from BASIC to hold the definitions. The procedure below can be used to reserve the needed extra memory and then add definitions in that memory.

1. First execute the following statement to find the current top of memory:
   TB=PEEK(52)+PEEK(53)*256.

2. Next, compute what the new top of memory should be to reserve N bytes of memory for shape definitions. To do this, execute the following statememt, replacing "N" with the number bytes you wish to reserve: NT=TB-N-3 The extra 3 bytes specified is due to the fact that the CHDFC function writes three zero bytes in the shape table after each execution. Since the internal memory pointer isn't updated when these zero bytes are written, they will be overwritten if future CHDFC and CHINIT commands are executed. This is done to guarantee termination of your shape definition should you forget to include necessary zero bytes in the definition. This only guarantees termination of the definition, the shape will still not draw correctly if zero bytes are missing. The 298 bytes available in the KGP memory includes a subtraction for these three zero bytes.

3. Next, execute the following command to reserve the desired area of memory:
   POKE 53,INT(NT/256):POKE 52,NT-PEEK(53):NEW

4. Now, you tell the KGP that you wish to store future shape definitions in this reserved memory area by executing the command: CHDLOC NT

5. You may now add shape definitions in the same manner as in the previous section.

6. If you remember the value for TB above, you may determine the amount of memory you have left for shape definitions by executing the following statement:
   RDDLOC CL :PRINT TB-CL-3

6.4 SAVING KGP WITH NEW SHAPE DEFINITIONS INCLUDED

If a set of shape definitions are to be used with several different application programs or if memory space is tight, it is desirable to define the shapes once, and then save KGP along with the new definitions on tape. Then whenever KGP is reloaded, the definitions are already in place and the application program will not have to define them.

To add shape definitions so that they become part of the standard set, execute a CSETUP command, which rebuilds the standard set of definitions. You may then add your definitions in the usual way. The reason the added definitions will become part of the standard set is due to the fact that the CSETUP and CHBLD commands will build a pointer in the shape pointer table for as many definitions as they find stored sequentially in memory. They will stop building when a zero byte is found as the shape ID and leave the internal memory pointer pointing to this byte. After executing CSETUP, adding definitions will overwrite the zero byte which initially terminates the CSETUP. Your added definitions then become part of the standard set. Thanks to the CHDFC command there will be three zero bytes in the shape table at the end of your added definitions (see step 4 in Section 6.3.1) to terminate future CSETUP commands. To save the KGP with your added definitions included, simply follow the same procedure you used to make a backup copy of the KGP distribution tape (see section 2.6). When you later use this copy, your added definitions will be automatically installed because the SYS command to initialize the KGP will execute a CSETUP itself. It is important to note that at this point, there is no simple way to remove the added definitions from the standard set.

To add some definitions to be kept seperate from the standard set you should execute CSETUP:CHINIT 0 before adding the definitions. The CHINIT 0 will preserve the zero byte which terminates the CSETUP's building process. The added definitions will then be separate from the standard set. The SYS command to initialize the KGP will also preserve this zero byte by incrementing the internal memory pointer after it executes a CSETUP. If the definitions were entered one right after the another, they can be considered to be a second set of definitions. To save this second set with the KGP, use the backup procedure as before. When this copy of the KGP is later used, executing the statememt CSETUP:CHINIT 0:CHBLD is required to install the second set.

In the above cases you are again limited to the 307 byte of unused space in the KGP memory. If you require more memory than this, you can store the set of definitions somewhere else in memory. To do this, execute steps 1,2, and 3 of the procedure given in Section 6.3.1. Next execute CHDLOC NT :CHINIT 0, and then add your definitions. Now save the KGP starting from the beginning of your added set to the end of KGP. When this copy of the KGP is later used, execute CHDLOC NT: CHINIT 0: CHBLD to build in the added set. The CHINIT 0 is needed in this case because the CHBLD will give an error if a CHINIT has not been executed since the last CSETUP or CHBLD.

If you have added a second set of definitions, it's possible to have only those definitions built into the shape pointer table. Simply execute the statement CSETUP: CHRESET: CHINIT 0: CHBLD, or CHDLOC NT: CHRESET: CHINIT 0: CHBLD depending on where the second set is located. The CHRESET command will clear the shape pointer table by storing zeros in all the pointers.

7. ADDITIONAL USAGE INFORMATION

In this section will be found additional hints, usage information, and obscure commands of use to advanced programmers.

7.1 HINTS FOR IMPROVING SPEED

The average user of the Keyword Graphics Package will quickly realize that the generally slow speed of interpreted BASIC really becomes noticable when hundreds or thousands of data points must be calculated to produce a single graph, family of curves, surface approximations of solid objects, etc. All of the techniques normally used for improving the speed of BASIC programs are equally applicable when KGP is being used.

However KGP itself slows down execution about 25% because each keyword in the program must be scanned to determine if it is one of the new graphics oriented commands. Nearly all of the speed may be regained by enabling the abbreviated command with the GRSHRT command. In the abbreviated mode all commands are preceeded by a] character which makes scanning for graphics commands very fast. Only the following commands are recognized in the short command mode:

Long form	Short form	Function
AUTEXT	]A	Plot text on multiple lines with formatting
CHAR	]C	Plot characters (shapes) in a single row
DRAW	]D	Draw a line from the drawing cursor to a point
CLEAR	]E	Clear the entire Visible Memory
XFFLG	]F	Set the coordinate transformation mode
GRACSR	]G	Display the graphics cursor (crosshairs)
LINE	]L	Draw an arbitrary line between two endpoints
MOVE	]M	Position the drawing cursor
PETMEM	]P	Set screen display to PET video only
TEXCSR	]T	Display the text cursor (underline)
WRPIX	]W	Plot a dot at the drawing cursor position
GRSHRT		Set abbreviated command mode
	]X	Restore full name mode

Note that the short form of the commands is always recognized thus they may be mixed with long form commands and the program will run correctly in long command mode.

If the execution of a particular non-graphic routine dominates the run time of a program (such as a Fourier analysis subroutine), one can simply code a GRSHRT command at the beginning of the routine and a]X command at the end to speed up that particular routine without having to sacrifice readability of other portions of the program. If one needs the last few percent of speed capability, KGP itself may be disabled with a GKILL command and then later re-enabled with the appropriate SYS command (see section 1).

If plotting time tends to dominate the program rather than computation (such as plotting from a list of precalculated values), plotting speed may be increased by turning boundary checking off with the NOCHK command. With boundary checking off, attempts to plot outside the window or screen boundaries may lead to strange-looking graphics on the screen. Enough residual checking is maintained however to prevent writing outside of the Visible Memory address range. Boundary checking may be restored with the BNDCHK command. See section 5.2 for additional information.

7.2 DEFAULTS

In order to simplify use by novices, the Keyword Graphics Package "comes up" in a usable state through the application of defaults. Thus every operating mode, parameter, etc. has an initial setting that is used in the absence of any commands to reset or change it. The theory behind defaults is that a novice user need not know or worry about a particular parameter until he has a need to use it. Below is listed the various KGP parameters and their default values:

1. Full name command mode - both full name and abbreviated commands are recognized.

2. Graphics scale factor is $2^0=1.0$ which means that the graphic screen is 320 coordinate units wide by 200 coordinate units high.

3. Graphics X and Y offset values are both zero. This means that the origin is at the lower left corner of the screen and all X and Y values must be positive.

4. XFFLG set to 0 - coordinate transformation not done regardless of the graphics scale factor and offset settings.

5. Normal display mode - black background with white (green) plotting.

6. Plotting mode set to 0 - Pixels are flipped when plotting, thus tiny gaps will appear where lines cross.

7. PET video display - only PET video is seen, a VISMEM command is required to see the graphics video.

8. Full boundary checking is enabled - plotting outside the boundaries will not show on the screen.

9. Solid line mode is selected - all lines are shown as solid lines.

10. Window 0 is selected for use.

11. The boundaries of all windows are set for left=0, right=319, bottom=0, top=199.

12. CHROT is set to zero - all characters and shapes are right-side-up.

13. VMPAGE is set to 64 in the 16K version, 96 in the 24K version, and 144 in the 32K version.

8. ERROR CODES AND MESSAGES

The KGP can detect when certain values are out of range, and in some cases, when parameters are missing. However, the majority of errors which occur during normal programming will be detected by the PET operating system. If only the Visible Memory is being displayed when an error is detected by the PET operating system, the PET error message will not be visible, nor any other direct indication that the program has stopped. In this case, the PET has not crashed and commands can still be entered even though they are not visible. The best procedure to follow when it is thought that an error has occured is to first hit the STOP key to make sure the program is stopped. Then enter the command "]P" followed by Carriage Return to bring back the PET display. If the last message is "BREAK IN LINE ...", then the program was still running when the STOP key was depressed. If an error had occurred, it will now be visible. If the KGP detects an error, it will switch back to the PET display before printing the error message.

The error checking in the KGP is limited to checking if all the required parameters are present and if certain parameters are within the required range. If a parameter is missing, the KGP will select PET video and then give a SYNTAX ERROR message. If a parameter is out of range, the KGP will select PET video and give an ILLEGAL QUANTITY ERROR message. Note that PET BASIC also uses these same messages for some of the errors _it_ detects and that PET BASIC will _not_ select PET video when printing the message.

X and Y plotting coordinates outside of the current window boundaries is itself not an error. If boundary checking is on, points outside the window boundaries are simply not plotted. If boundary checking is off, such points will be transformed such that they _may_ be plotted but in an unexpected location. If an X or Y coordinate is beyond the two byte integer range of -32768 to +32767, then an ILLEGAL QUANTITY ERROR will be generated as described above.

9. MEMORY MAP

The storage allocation utilized by the KGP is designed to minimize any possible interference with BASIC. Essentially three types of storage are needed for any 6502 program and the KGP is no exception. Memory required to hold the program is placed at the very top of available memory which is 2200 (hexadecimal) for 16K PET's, 4200 for 24K PET's, and 6200 for 32K PET's. Shape tables are also included in this memory area. Additional memory is required to hold the cursor positions, scale factors, and offsets for each of the 4 windows. This has been put into the second cassette buffer so that the addresses are version independent. Thus use of a second cassette recorder is incompatible with KGP. Finally, 4 bytes in page 0 are required to hold address pointers. The original contents of these bytes are saved by the KGP when it starts executing and are restored when it is finished executing a command. A detailed memory map of the Keyword Graphics Package is given below:

16K	24K	32K	Use
3FFF	5FFF	7FFF	
			Shape pointer table
3E00	5E00	7E00	
			Free space for user shape definitions
3CD1	5CD1	7CD1	
			Predefined shape definitions (ASCII character set)
3871	5871	7871	
			Graphics routines
2900	4900	6900	
			BASIC interface routines
2200	4200	6200	
			Available storage for user's program, Fast Wedge, etc.
0400	0400	0400	
			Used by the PET operating system
03B8	03B8	03B8	
			Variable storage used by the graphics routines
0340	0340	0340	
			Used by the PET operating system
006A	006A	006A	
			Variable storage used by the graphics routines
0066	0066	0066	
			Used by the PET operating system
0000	0000	0000	

10. LISTING OF KGPDEMO DEMONSTRATION PROGRAM

```
1 REM MTU GRAPHICS EXAMPLES
200 REM DEMONSTRATION OF POINT PLOT
205 REM PLOT A CIRCLE IN DEAD CENTER OF SCREEN USING 100 POINTS
210 CLEAR:REM CLEAR SCREEN
215 VISMEM:REM LOOK AT VISIBLE MEMORY
220 GMODE 1:REM PLOT ALL THE POINTS
230 FOR I=0 TO 100
240 A=6.28318*I/100
250 MOVE 159*COS(A)+160.5 , 99*SIN(A)+100.5
270 WRPIX:REM WRITE THE POINT
280 NEXT I
290 GOSUB9000
300 REM DEMONSTRATION OF VECTOR PLOT
310 REM SET MODES - ON, OFF, FLIP, FLIP
320 GM(1)=1:GM(2)=2:GM(3)=0:GM(4)=0
330 REM PLOT IN ALL 4 MODES
340 FOR MD=1 TO 4
345 REM CLEAR THE SCREEN BEFORE PLOTS 1 AND 3
350 IF(MD=1)OR(MD=3)THEN  E
360 TM=GM(MD):GMODE 1
365 AUTEXT 19;"GRAPHICS MODE";32;20;157;STR$(TM)
370 GMODE GM(MD)
380 FP=1:REM SET FIRST POINT FLAG
400 FOR I=0 TO 31
410 A=13*I*6.2831828/31
420 X=159*COS(A)+160.5
430 Y= 99*SIN(A)+100.5
440 IF FP=1 THEN MOVE X,Y:FP=0:GOTO470:REM MOVE FOR FIRST POINT
450 DRAW X,Y
470 NEXT I
480 GOSUB9000
490 NEXT MD
600 REM DEMONSTRATION OF AXIS PLOT AND LABEL
610 CLEAR:GMODE 1
620 REM INSERT Y AXIS LABELLING FIRST
630 DS=9:REM 9 DOTS BETWEEN LABELS
640 FOR Y=-10 TO 10 STEP 2
650 REM REPOSITION TEXT CURSOR
660 P=Y+10:REM P GOES FROM 0 TO 20
670 MOVE 0,P*DS+12
680 CHAR STR$(Y/10):REM PRINT THE LABEL
690 NEXT Y
700 REM PRINT X AXIS CAPTION
710 MOVE 49*6,90+12:CHAR "TIME"
730 REM PRINT X AXIS CAPTION AND FIGURE CAPTION
740 MOVE 0,0 :CHAR "AMPLITUDE     WAVEFORM OF GREAT DIASPON C4 16FT"
800 REM PLOT X AND Y AXES
820 LINE 20,105,284,105:REM HOR AXIS
840 LINE 20,11,20,199:REM VERT AXIS
900 REM PLOT TIC MARKS ON Y AXIS
910 FOR Y=-10 TO 10 STEP 2
920 P=Y+10:REM P GOES FROM 0 TO 20
930 Y1=P*DS+12+3
940 LINE 18,Y1,20,Y1
950 NEXT Y
1000 REM PLOT THE WAVEFORM USING VECTORS
1010 FP=1
```

```
1020 XF=270/(2*3.14159):REM X SCALE FACTOR
1030 YF=60:REM Y SCALE FACTOR
1040 FOR X=0 TO 2*3.14159 STEP 4*3.14159/270
1050 Y=SIN(X)+.49*SIN(2*X+3.9)+.3*SIN(3*X+5.81)
1060 Y=Y+.24*SIN(4*X+3.8)+.18*SIN(5*X+.97)
1070 Y=Y+.12*SIN(6*X+4.3)+.04*SIN(7*X+3.54)
1080 Y=Y+.07*SIN(8*X+.87)+.03*SIN(9*X+5.3)
1090 IF FP=1 THEN MOVE 20.5+XF*X,105.5+YF*Y:FP=0:GOTO 1130
1100 DRAW 20.5+XF*X,105.5+YF*Y
1130 NEXT X
1140 GOSUB9000
2000 REM EXAMPLE OF USING WINDOWS
2005 REM SETUP THE WINDOWS
2010 RVSDSP:CLEAR:NRMDSP
2020 W$="WINDOW":GMODE 1
2030 SETWIN 0,8,8,223,170
2035 WINDOW 0:WCLEAR:CHAR W$;STR$(0)
2040 SETWIN 1,240,9,311,90
2045 WINDOW 1:WCLEAR:CHAR W$;STR$(1)
2050 SETWIN 2,240,100,311,171
2055 WINDOW 2:WCLEAR:CHAR W$;STR$(2)
2060 SETWIN 3,24,180,295,195
2065 WINDOW 3:WCLEAR:CHAR W$;STR$(3)
2066 GOSUB 9000
2100 REM INITIALIZE ROUND-ROBBIN USE OF THE WINDOWS
2110 S0=1:S1=1:S2=1:S3=1:C1=0:C2=0:C3=0
2120 D3$="       THIS DISPLAY IS BROUGHT TO YOU BY"
2130 D3$=D3$+" THE MTU KEYWORD GRAPHICS PACKAGE "
2140 I3=6
2150 I2=65:WINDOW 2:WCLEAR:MOVE 309,108
2160 I1=32:WINDOW 1:WCLEAR
2170 C=6.28318/50:I0=0
2180 WINDOW 0:MOVE 115,89:DS=0
2200 REM BEGIN ROUND-ROBBIN USE OF THE WINDOWS
3000 IF S0=0 GOTO 3100
3010 WINDOW 0:REM DRAW LINES
3020 GMODE 1
3030 RDXY X,Y
3040 GRSHRT:DS=DS+1
3050 DR=DR+1:IF DR=4 THEN DR=0
3060 IF DR=0 THEN ]D X+DS,Y:GOTO 3080
3065 IF DR=1 THEN ]D X,Y+DS:GOTO 3080
3070 IF DR=2 THEN ]D X-DS,Y:GOTO 3080
3075 IF DR=3 THEN ]D X,Y-DS
3080 I0=I0+1:IF I0=100 THEN S0=0
3090 ]X
3100 IF S1=0 GOTO 3200
3110 WINDOW 1:REM PRINT CHARACTER SET
3120 AUTEXT I1:I1=I1+1
3130 IF I1=127 THEN I1=32:C1=C1+1
3140 IF C1=2 THEN S1=0
3200 IF S2=0 GOTO 3400
3210 WINDOW 2:REM PRINT UPSIDE-DOWN
3220 CHROT 2:REM SET FOR UPSIDE DOWN
3230 RDXY X,Y
3240 IF X<246 THEN GOSUB 3300
3250 CHAR I2:I2=I2+1
3260 IF I2=96 THEN I2=64:C2=C2+1
3270 IF C2=8 THEN S2=0
3280 CHROT 0:REM RESTORE
```

```
3290 GOTO 3400
3300 REM TEST FOR RETURN AND SCROLL
3310 IF Y>158 THEN Y=Y-10:SCROLL 240,100,240,110,311,171
3320 MOVE 308,Y+10
3330 RETURN
3400 IF S3=0 THEN GOTO 3480
3410 WINDOW 3:REM HORIZONTAL SCROLLING
3420 SCROLL 24,180,32,180,295,195
3430 MOVE 288,185
3440 CHAR MID$(D3$,I3,1)
3450 I3=I3+1
3460 IF I3=LEN(D3$)+1 THEN I3=1:C3=C3+1
3470 IF C3=2 THEN S3=0
3480 IF S0=1 OR S1=1 OR S2=1 OR S3=1 THEN GOTO 3000
3490 GOSUB 9000
4000 REM DO CHECKERBOARD
4010 NRMDSP:CLEAR
4020 GMODE 1
4030 FOR I=0 TO 319 STEP 40
4040 FOR J=0 TO 199 STEP 25
4050 FLPDSP
4060 SCLEAR I,J,I+39,J+24
4070 NEXT J
4080 FLPDSP
4090 NEXT I
4100 REM DO THE ROAMING "X"
4110 NRMDSP:GMODE 0
4120 SETWIN 0,0,0,319,199:WINDOW 0
4130 MOVE 160,100
4140 MD=ASC("3")+128:REM MOVE DIRECTION
4150 C=ASC("X"):BS=157:REM BACKSPACE
4160 CHAR C
4170 REM MAIN LOOP
4180 CHAR BS:RDPIX PX
4185 IF PX=255 THEN GOSUB 4300
4190 CHAR C:CHAR BS:REM ERASE
4200 CHAR MD;MD;MD;MD;MD;MD:REM MOVE
4210 CHAR C:REM DRAW AGAIN
4220 GET DR$
4230 IF DR$>"0" AND DR$<="9" THEN MD=ASC(DR$)+128
4240 IF DR$<>"0" THEN GOTO 4180
4250 PVMEM:GOSUB 9000: P:END
4260 REM END OF DEMO
4300 RDXY X,Y:REM AT BOARDER, CHANGE DIRECTION
4310 IF X>=0 GOTO 4340
4320 IF MD<=183 THEN MD=MD+2:RETURN
4330 MD=182:RETURN
4340 IF X<320 GOTO 4370
4350 IF MD>=179 THEN MD=MD-2:RETURN
4360 MD=180:RETURN
4370 IF Y>=0 GOTO 4400
4380 IF MD<=179 THEN MD=MD+6:RETURN
4390 MD=184:RETURN
4400 IF Y<200 THEN RETURN
4410 IF MD>=183 THEN MD=MD-6:RETURN
4420 MD=178:RETURN
9000 REM WAIT FOR A FEW SECONDS
9010 TJ=TI+100
9020 IF TI<TJ THEN 9020
9030 RETURN
9999 END
```

11.2 SPECIAL FUNCTION CHARACTERS

The following function characters are recognized in the character strings given to the CHAR command or the AUTEXT command:

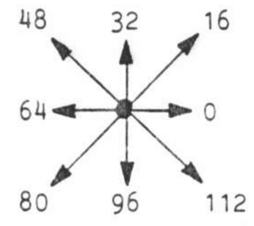
PET KEY	ASCII CODE (DECIMAL)	DEFINITION
DEL	20	Erase the character at the current character position.
Up-arrow	145	Move cursor up one line.
Down-arrow	17	Move cursor down one line.
Right-arrow	29	Move cursor right one character position.
Left-arrow	157	Move cursor left one character position.
Shift-1 - Shift-9	177-185	Move one dot position in direction the digit is from 5.
Shift-D	196	Add one to CHSCALE, i.e. double the current character size.
Shift-E	197	Subtract one from CHSCALE, i.e. halve the current character size.
Shift-F	198	Add one to CHROT, i.e. rotate orientation 45 degrees counterclockwise.
Shift-G	199	Subtract one from CHROT, i.e. rotate orientation 45 degrees clockwise.
Shift-$	164	Draw the contents of BASIC string variable GR$ as a definition, i.e. contains vector bytes, etc. The definition contained in GR$ MUST end in a zero byte!
CLR	147	AUTEXT only. Clears the region specified by the current boundaries, and then homes the drawing cursor coordinates.
HOME	19	AUTEXT only. Homes the drawing cursor coordinates.
RETURN	13	AUTEXT only. Performs a carriage return followed by a line feed.

11.3 CHARACTER DEFINITION INSTRUCTIONS

The following instructions are recognized inside of a shape definition:

SECTION	CODE	OPERANDS	DESCRIPTION
	239	A,B	Add A-128 to X cursor position and add B-128 to Y cursor position
	223	A,B	Draw a line from the current cursor position to A-128+Xcursor,B-128+Ycursor then update the cursor position.
	47	ID	Draw the shape specified by the ID at the current cursor position. The shape itself may use a 47 instruction.
	63	ID,ID,...,0	Draw the series of shapes specified by the ID's at the current cursor position. The shapes may themselves use a 63 or 47 instruction.
	95	val	Add val-128 to the present value of CHROT. Result should lie in the range of 0-3.
	175	val	Add val-128 to the present value of CHSCALE.
	0		End of shape definition
	Legal Vector Bytes		The vector byte is simply the sum of the three numbers calculated according to the chart below:

DIRECTION NUMBER



LENGTH NUMBER

Specifies length of move or draw. This number may be from 0 to 14.

MOVE/DRAW NUMBER

Is 0 for move.
Is 128 for draw.

11.4 COMMANDS WHICH LEAVE THE DRAWING CURSOR UNDEFINED

SCLEAR
SCFLIP
SETWIN
SCROLL
CSETUP
XFFLG
OFFSET if XFFLG is 1

12. APPENDIX B

This section describes appropriate corrective action when difficulty is experienced due to failure to follow instructions, etc.

12.1 WHAT TO DO IF THE KGP WON'T LINK IN

It is important to note that the first time the initializing SYS is executed, the KGP will link itself to BASIC and then set a status byte before performing the necessary initialization. If another initializing SYS command is executed, this status byte is checked, and if set, only the initialization is performed. This is necessary to allow the KGP to link in between another package such as the Programmer's Toolkit (tm). However, if the KGP link is destroyed after this status bit is set, the SYS command won't relink the KGP. This would happen if you executed the Toolkit's SYS command after the KGP was linked in rather than before as the instructions in section 2.6 indicate. You will also encounter this problem if you made your backup copy while the KGP was linked in rather than before as described in section 2.7. At this point the the quickest way to solve the problem is to reset the status byte with one of the following POKE commands:

```
16K Version:    POKE 256*34+18,0

24K Version:    POKE 256*66+18,0

32K Version:    POKE 256*98+18,0
```