PG user's manual

PG-640A

Professional Graphics Board
for the IBM XT and AT
July 1, 1987
277-MU-00
Rev. 5

matrox
electronic systems

[redacted]

PG-640A

Professional Graphics Board
for the IBM XT and AT
July 1, 1987
277-MU-00
Rev. 5

This Manual Is Valid For The Following Products		
Name	**Hardware I.D.**	**Firmware I.D.**
PG-640A CPU	REV.5	REV.2
PG-640A VGM	REV.1	

MATROX Electronic Systems Limited,
1055 St. Regis Boulevard,
Dorval, Quebec,
CANADA H9P 2T4
Telephone: (514)685-2630 Telex: 05-822798 FAX: (514)685-2853

FEATURES

- IBM Professional Graphics Controller (PGC) 100% compatible
- 10 times as fast as the IBM PGC
- 640 × 480 resolution
- 256 colours from a palette of more than 16 million colours
- 32/16 bit display processor
- VLSI drawing processor
- 40,000 vectors/second
- 5,000 characters/second
- 1,200,000 pixel/second raster operations
- Enhanced instruction set includes text windows, stroke text, and raster operations
- IBM XT or AT compatible
- VDI compatible
- Demonstration and diagnostics programmes included
- Low Cost

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

List of Tables

LIST OF TABLES

Chapter 1

Introduction

Thank you for purchasing the MATROX PG-640A. The PG-640A is a plug in card set that allows an IBM PC microcomputer to perform high level, high resolution graphics operations. This manual provides all of the information required to install, programme, and operate the PG-640A.

Details of the PG-640A's capabilities can be found in the functional description in Chapter 2. Chapter 3 is dedicated to programming the high level graphics engine, and provides information that the user must have in order to operate the graphics engine. Chapter 4 contains the command descriptions for the high level graphics engine. The PG-640A's colour graphics emulator is described in Chapter 5. Chapter 6 provides information on maintenance and warranty. Appendix A gives a brief installation and check out procedure, Appendix B lists the high level graphics engine's parameter default values, Appendix C lists the board's specifications and features in point form. Appendix D describes the monitor programme and Appendix E lists the lookup table data. The diagnostics programs, the self test program, and the PG-640A's LEDs are described in Appendix F. Appendix G outlines the utility programs provided with the PG-640A. Appendix H explains how to install the PG-

640A VDI device driver. Circuit board layout diagrams can be found in Appendix I, and Appendix J contains a summary of the commands for the high level graphics engine.

We believe this manual contains all the information needed to get your PG-640A operational; however, if you do have problems feel free to telephone anyone in our applications engineering department. They will be happy to answer any questions you may have.

Chapter 2

Functional Description

The PG-640A is an intelligent graphics controller for the IBM PC bus. It is 100% software compatible with the IBM Professional Graphics Controller and can execute software 10 times faster than the IBM PGC. Several new commands have been added to the PGC command set in order to improve the versatility of the PG-640A. The speed and power of the PG-640A make it an ideal choice for applications such as CAD/CAM, presentation graphics, and mapping systems.

The PG-640A has a colour graphics adaptor emulator section built-in that provides emulation of the alphanumeric and graphics modes of the IBM Colour Graphics Adaptor. The presence of the emulator allows the user to run software that requires a colour graphics adaptor without the need to purchase an additional monitor and adaptor card. The High Level Graphics Engine of the PG-640A occupies 1Kbyte of address space in the PC, and the emulator occupies 16Kbyte of address space and 16 bytes of I/O space.

The High Level Graphics Engine allows the user to create images with minimal use of the system micro-processor. The PG-640A provides the intelligence needed to draw, in two or three dimensions, geometric prim-

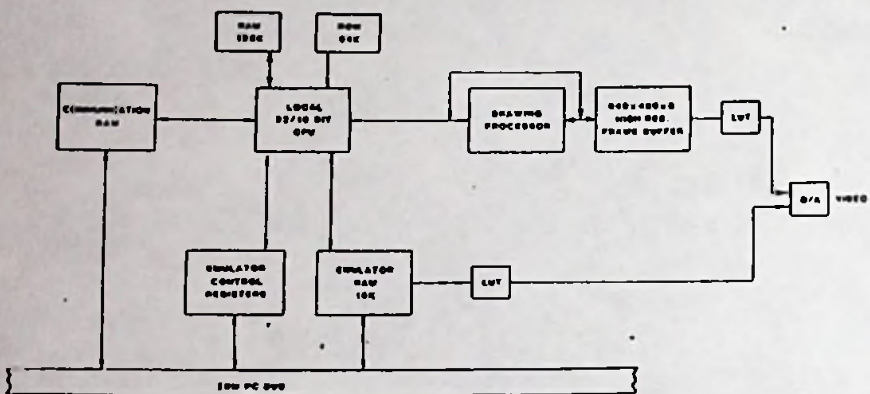


Figure 2.1: PG-640A Block Diagram

itives by specifying their size and type. High level graphics commands are sent to a 1Kbyte FIFO buffer and are executed by the PG-640A. Alternately several commands can be stored in a command list and then be executed at any time. This is different from the colour graphics adaptor, which allows the user to draw only single pixels and alphanumeric characters. As the on board micro-processor of the PG-640A provides the intelligence for the emulator and also controls the drawing processor, the user can display part of the emulator output in a window in the high level graphics display. The relationship between the two graphics drivers is illustrated in Figure 2.1.

2.1 The High Level Graphics Engine

2.1.1 Hardware

The PG-640A uses a micro-processor with a 32-bit internal architecture and a 16-bit bus. This processor acts as the command processor and provides the intelligence to process high level commands into instructions for the drawing processor. The on board CPU also has the processing power to provide virtual coordinate addressing and matrix transforms. This allows the user to choose the coordinate space to be in two or three dimensions with the PG-640A performing the necessary three dimensional to two dimensional transforms. The command processor uses a 1Kbyte FIFO queue to buffer commands from and responses to the system unit CPU. One hundred and twenty-eight kilobytes of ROM provide software to parse commands and to generate instructions for the drawing processor. There are 128Kbyte of RAM provided to store command lists, user fonts, and internal variables. The drawing processor draws primitive graphics forms directly into the 320Kbyte video display buffer.

The video display buffer provides output data which is passed through a lookup table. The user can load this LUT with any 256 colours from a palette of more than 16 million, permitting changes to any colour on the display with out altering the video display buffer.

2.1.2 Coordinate Space and Transforms

The PG-640A has firmware to enable it to draw in either the two or three dimensional virtual work spaces. In both work spaces the axes have 32-bit values and the user can define both the window and the view port. The window is the section of the virtual work space that the user wishes to be mapped to the view port. The view port is the physical area of the screen that can be modified. While the user can always modify the entire virtual work space, only the pixels that correspond to points in the window are affected by graphics commands. The results of drawing commands on areas inside the virtual work space, but outside of the window, will not appear on the screen or be saved – images that pass through the window will be clipped as they are mapped to the view port. Alternately, there is a set of direct screen commands that allow the user to draw directly to the screen, bypassing the transforms and increasing drawing speed.

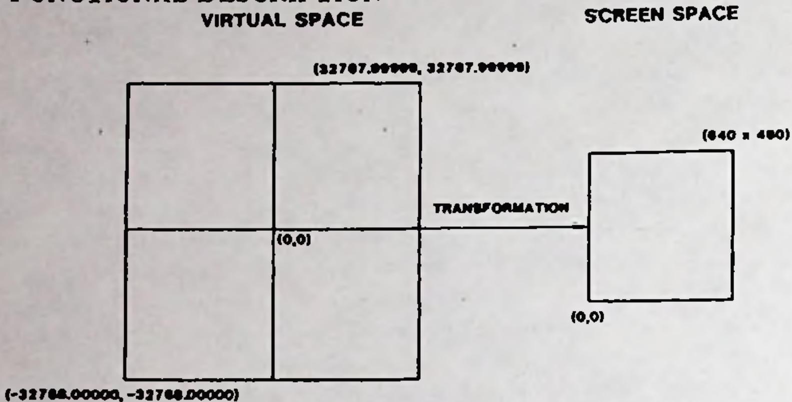


Figure 2.2: Two Dimensional Virtual Space to Pixel Mapping

When drawing in two dimensions, the user has at his disposal a set of two dimensional graphics commands. These commands draw the graphics primitives: points, lines, arcs, circles, ellipses and polygons. The user can set masks so that dashed lines and patterns in filled figures are produced. The virtual points are mapped to the real display coordinates (pixel locations) by the PG-640A (see Figure 2.2). For a more detailed discussion of two dimensional drawing, see Chapter 3.

In three dimensions, the user has access to the virtual coordinate system as well as full control over viewing angles and distances. The PG-640A uses a modelling matrix to rotate, scale, and translate the virtual coordinates of the three dimensional object. A viewer reference point matrix is used to translate a point to the centre of the currently defined view port. This view port matrix affects the angle of rotation by moving the eye about the object – leaving the object stationary, see Chapter 3.

The user can also set the angle and distance from the three dimensional origin to the two dimensional origin. This allows both two dimensional and three dimensional objects to be drawn in the same coordinate space.

2-D Command*	3-D Command	Effect	Move Current Point
ARC		draws arc	no
CIRCLE		draws circle	no
DRAW	DRAW3	draws line	yes
DRAWR	DRAWR3	draws line	yes
ELIPSE		draws ellipse	no
MOVE	MOVE3	moves current point	yes
MOVER	MOVER3	moves current point	yes
POINT	POINT3	colours current point	no
POLY	POLY3	draws polygon	no
POLYR	POLYR3	draws polygon	no
RECT		draws rectangle	no
RECTR		draws rectangle	no
SECTOR		draws pie slice	no

* Direct screen operations parallel the 2-D commands

Table 2.1: Drawing Command Summary

2.1.3 Graphics Attributes and Primitives

The PG-640A presents the user with a drawing model consisting of a pen and ink. The pen has two positions, the two dimensional and three dimensional current points. The ink has 256 colours, those that are stored in the output lookup table. Drawing operations use the current colour. The current points can be moved to any point in their respective coordinate spaces with a single command and the current colour can be selected from any of the LUT colours, again, with a single command. Primitives are drawn from the appropriate current point in the current colour – some relocate the current point, others do not. See Table 2.1.

The high level graphics commands provide the ability to draw geometric figures with single commands. These figures can be drawn with patterned lines, and filled in the case of closed figures. How the figure is drawn is dependent upon how the Area Pattern and Line Pattern Masks are set, and whether or not they are enabled. There are five drawing modes to allow for different types of pixel replacement. The PG-640A also has the ability to mask off entire bit planes in the display buffer from read and write operations. This allows the user to load different images into the buffer and to perform image overlays.

The two dimensional command set provides instructions to draw arcs, circles, ellipses, lines, points, polygons, and rectangles. In three dimensions, the user can draw lines, points, and polygons.

2.1.4 Text

Text is specified in two dimensional space. There are two pre-defined fonts and two user defined fonts. Characters can be drawn as thin stroke, vector based characters or fat, smooth characters that are constructed with lines whose thickness is proportional to the character size. The user can set the size, angle of rotation, and aspect ratio of the characters. The justification about the current point can also be set.

2.1.5 The Text Window

On the PG-640A there is a provision for a window, containing part or all of the emulator screen, to be overlayed on the high level graphics screen. This allows the user to concurrently display both high level graphics and emulator output. The user can set the size and position of the emulator window, and enable or disable it.

2.1.6 Direct Screen Operations

One of the major features of the PG-640A is the ability to perform block moves of pixel data. The user can copy a block from one part of the display buffer to another. Using a single command, the user defines the block to be transferred, its destination, and the major and minor directions in which it is to be read and written. It is by setting the transfer directions that the user has the ability to invert or rotate the pixel blocks. The inversion of a block of pixels is illustrated in Figure 2.3.

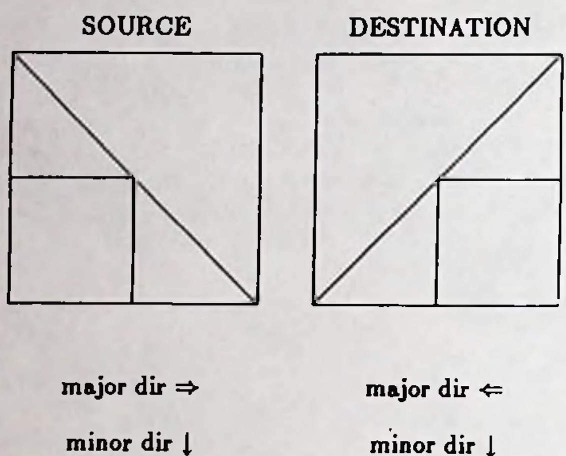


Figure 2.3: Raster Transfer of Pixels

Images can also be transferred to and from the system unit. Pixel values can be sent through the system unit and can also be transferred by DMA. This allows the rapid reading and writing of images making the PG-640A a useful tool for displaying images.

There are fourteen commands supported by the PG-640A that allow the user to plot pixels directly on to the display without going through the modelling mechanism. These commands have the advantage of having much faster drawing speeds and are specified directly in screen coordinates.

2.2 The CGA Emulator

The PG-640A has an on board colour graphics adaptor emulator. This emulator allows the user to run MS-DOS software in his PC without having to purchase a second monitor and adaptor. The emulator is fully compatible with the colour adaptor. See Chapter 5.

Chapter 3

The High Level Graphics Engine

This chapter explains how to program the HLGE. It does so by assembling related commands into groups and explaining how they are used together to accomplish various tasks. Although it gives the formats of many commands, it is not intended to be used for command reference–Chapter 4, which contains the command descriptions arranged in alphabetical order, is better suited for that purpose. Rather, it is intended to be an overview of the HLGE's various functions taken from a programmer's point of view.

3.1 Introduction

Most people using the HLGE will not have to program it. They will simply run applications programs that are compatible with it. In some cases, however, the user will want to program the HLGE.

In such a case the programmer's task with respect to the HLGE is to interface it to a CPU running another level of software. How this is done depends on the application. For example, if the HLGE is being used to display the output of an original assembly language application program, the programmer will have to write parts of that program to interface with the HLGE. If the programmer is adapting a graphics package to run the HLGE he will have to write drivers so that the package can display graphics on the HLGE. In another situation the programmer might be called upon to write driver routines that could be called from a program written in a high level language such as BASIC.

The programmer operates the HLGE by passing it commands. The form that those commands take depends on which of two command modes the programmer is using. In one command mode, called ASCII Mode, the commands are passed as ASCII strings forming keywords, ASCII decimal value parameters, and ASCII character parameters. The string 'CLEARS␣23', for example, causes the HLGE to clear the screen to the color corresponding to color index 23. Keywords in this mode have a short form which can be used for brevity. In this case, for example, 'CLEARS␣23' can also be sent as 'CLS␣23'. ASCII Mode provides ease of operation since the keywords are mnemonic in nature and the parameters are decimal values. Commands in this mode do, however, take more space than commands using the other command mode, referred to as Hex Mode.

Hex Mode allows the programmer to store and send his commands in a more compressed format. It uses binary opcodes instead of keywords and uses binary values instead of ASCII decimal values for parameters. For example, the Hex mode equivalent of 'CLEAR␣23' is `0F 17`. Hex Mode commands lack the mnemonic character of ASCII Mode commands and are more primitive; however, they can be stored in less space and sent to the HLGE in less time than ASCII Mode commands. See Section 3.2 for a more detailed explanation of the two command modes.

In this chapter, to keep things simple, we describe commands and give examples in ASCII Mode format only. Chapter 4, however, provides descriptions of both forms of each command.

The programmer communicates with the HLGE via a 1-Kbyte section of HLGE memory that is mapped into the system address space. This memory buffer is divided into 4 functional blocks referred to as the Command FIFO, the Read Back FIFO, the Error FIFO, and the Control Block. The user passes commands to the HLGE via the Command FIFO, reads status information from the Read Back FIFO, and reads error information from the Error FIFO. Both the HLGE and the system CPU use the Control Block to maintain pointers to the current read and write locations in each FIFO. See Section 3.3 for a detailed explanation of how to use FIFOs.

To make a 2D drawing, the user defines a window and a view port to map all or part of the 2D virtual coordinate space onto the screen; he selects graphics attributes such as color, line style, and drawing mode; then uses graphics primitives, text commands, and fill commands to draw the image. For example, putting the following string into the Command FIFO defines the window and view port shown in Figure 3.1 and draws a line in them. The ␣ characters represent any one of several delimiters. Valid delimiters are listed in Section 3.2, which explains the documentation conventions used to describe commands in this manual.

CLEARS␣0␣
WINDOW␣-10000␣10000␣-10000␣10000␣
VWPORT␣200␣500␣100␣400␣
MOVE␣0␣0␣
DRAW␣20000␣20000␣

Section 3.4 explains coordinate spaces, windows, and view ports; Section 3.5 explains graphics attributes; Section 3.6 explains graphics primitives; Section 3.7 explains text commands; and Section 3.8 explains fills.

3D drawing is a little more complicated than 2D drawing. The user makes the drawing in a 3D coordinate space which is mapped into the same window and view port used by the 2D coordinate space. How the image is mapped into the view port depends on a number of transforms that the user specifies before he does the drawing. These transforms define the following aspects of the image:

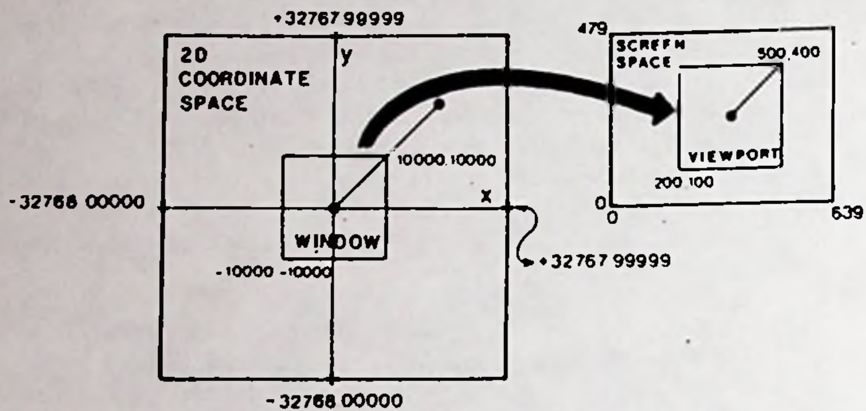


Figure 3.1: The 2D Drawing Environment

- the scale, rotation, and translation (position) of the image in the 3D coordinate space.
- the position and direction of view of the viewer with respect to the 3D coordinate space.
- the hither and yon clipping planes.
- the distance of the viewer from the viewing plane and his angle of view.

The 3D transforms and coordinate space are described in Section 3.4.

The following command string uses the default 3D transforms to draw the figure shown in Figure 3.2. The particular operations performed by this code will become clear as you read this chapter.

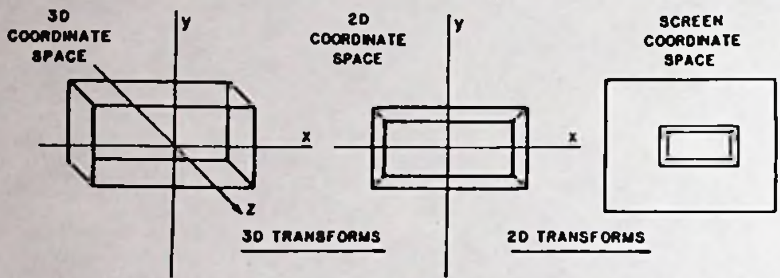


Figure 3.2: The 3D Drawing Environment

```
CLEARS␣0
MOVE3␣-100␣-50␣50␣
POLYR3␣4␣0␣0␣0 ␣200␣0␣0 ␣200␣100␣0 ␣0␣100␣0␣
DRAWR3␣0␣0␣-100␣
POLYR3␣4␣0␣0␣0 ␣200␣0␣0 ␣200␣100␣0 ␣0␣100␣0␣
MOVE3␣-100␣50␣-50␣
DRAWR3␣0␣0␣100␣
MOVE3␣100␣50␣-50␣
DRAWR3␣0␣0␣100␣
MOVE3␣100␣-50␣-50␣
DRAWR3␣0␣0␣100␣
```

The user can store drawings in the HLGE in the form of command lists that can be run (drawn) as required. For example, if a figure is in a command list and the user wants to move it to another part of the screen, he sets up a new translate transform, clears the screen, and runs the command list. The use of command lists is explained in Section 3.9.

The programmer can perform certain operations directly on the screen (the screen coordinate space), bypassing the coordinate spaces and transforms. He can use the 'S' series commands to draw fast graphics primitives in the screen coordinate space, he can use rasterops to copy one part of the screen to another and he can transfer all or part of the screen to or from system memory. These operations are described in Section 3.10.

3.2 Command Format

3.2.1 Documentation Conventions

Throughout this chapter and Chapter 4 we describe the different commands that the user can give to the HLGE. We use the following conventions to make these command descriptions easier to understand:

- We print parameter names in lower case block characters to identify them as such. For example, parameter.
- We print hexadecimal values in typewriter style characters. For example, `FFFE`.
- We print command keywords in upper case roman characters. For example, ARC.
- We use the ␣ character to indicate the position of a delimiter when it can be any one of several delimiters.

3.2.2 ASCII Command Format

When the HLGE is in ASCII Command Mode (the power-up default), the user passes commands to the HLGE as strings of ASCII characters. A command string consists of a keyword identifying the command, parameters (where required), and delimiter characters.

The keywords for most commands have a long form and a short form. For example the long form of the draw command is DRAW and the short form is D. The parameters are either text strings enclosed by quotes or ASCII decimal numbers. The allowed delimiters are:

- The space character.
- The tab character.

- The comma.
- The semicolon.
- The carriage return character.
- The line feed character.
- The hyphen acts as a delimiter at the same time that it identifies negative values.
- The plus sign acts as a delimiter at the same time that it identifies positive values.

For example, to draw a line from the current pen position to xy coordinate 100, 200 in the 2D coordinate space, the user would put the following ASCII string into the Command FIFO:

DRAW␣100␣200␣

where ␣ is any of the delimiters in the preceding list.

The ASCII Command Mode is particularly well suited for use with high level languages, since it takes advantage of their ability to easily manipulate strings.

Use the CA␣ command to enter ASCII Command Mode.

3.2.3 Hex Mode

When the HLGE is in Hex Mode, the commands that the user passes to the HLGE are binary byte values. A command consists of a single byte opcode followed by binary parameter values. In this manual we give these values as hexadecimal numbers.

Use the CX␣ command to enter ASCII Command Mode.

For example to draw a line from the current pen position to xy coordinate 100, 200 in the 2D coordinate space, the user would put the following values into the Command FIFO:

```
28  64  00  00  00  C8  00  00  00
 |  ______________  ______________
 |         |               |
 |         |               |_______200
 |         |_______________________100
 |_________________________________opcode
```

3.2.4 Parameter Types

The HLGE uses 3 different parameter types: Chars, Ints, and Reals. The way that these parameter types are represented depends on the current command mode.

The Char parameter type is a single ASCII character code in ASCII Mode. In Hex Mode it is a single byte value in the range 0-255.

An Int in ASCII Mode is an ASCII decimal value from -65535 to 65535 inclusive. A hyphen immediately preceding an ASCII Int indicates that it is a negative value. An unsigned Int is an ASCII decimal value from 0 to 65535. In Hex Mode an Int is a two byte binary value with the low byte first. Hex Mode negative Ints use two's complement form.

A Real is a value with a fractional part and a non-fractional part. In ASCII Command Mode, a Real is an ASCII decimal real number from -32768.00000 to +32767.99999 (the decimal is optional if the fractional part is 0). In Hex Command Mode, it is a real number represented by 4 bytes using the following format:

1	2	3	4	byte
xx	xx	xx	xx	

- 4: high byte of fractional part
- 3: low byte of fractional part
- 2: high byte of non-fractional part
- 1: low byte of non-fractional part

where the value of the bytes are derived by multiplying the decimal Real by 65536 and converting the result to hexadecimal form. For example 3.142 becomes:

$$3.142_{10} \times 65536_{10} = 205914_{10} = 0003245A_{16}$$

where 0003 is the non-fractional part, 245A is the fractional part, and the Real is sent as 03 00 5A 24.

This method is equally valid for calculating negative Reals. Thus -3.142 becomes FFFCDBA6 and is sent as FC FF A6 DB.

3.3 Communications

The user communicates with the HLGE via 3 FIFO's and a control block that are mapped into a 1K section of the system address space. On-board switches select one of two positions for this section. Each of the FIFO's occupies 256 bytes, the control block occupies 14 bytes, and 242 bytes are reserved. Table 3.1 gives the layout of the communications block and indicates how switch two of switch block SW1 selects its position. Subsection 3.3.1 explains how to access the FIFO's, Subsection 3.3.2 explains the use of the various flags in the control block, and Subsection 3.3.3 describes the commands to read the current status of certain system parameters. The last subsection is about the WAIT command.

3.3.1 FIFO Access

The user writes commands to the Command FIFO, reads read-back command data from the Read Back FIFO, and reads error and warning codes from the Error FIFO.

Each read pointer location and write pointer location contains an offset from the FIFO base address. The offset plus the base address give an address in the corresponding FIFO. In the case of a read pointer, this address is that of the next location to be read. In the case of a write pointer the address is that of the next location to be written to. Whenever the user or the HLGE's processor reads or writes a FIFO location, they adjust the corresponding pointer.

In a FIFO of this type there are two situations where the values of the pointers could be the same: (1) when the buffer is full of unread data and the write pointer is incremented to the value of the read counter or (2) when the FIFO is full of data that has been read and the read pointer is incremented to the value of the write counter. To avoid confusion and the possibility of overwriting unread data, our protocol only allows the latter of these two situations. That is to say you are not allowed to

ADDRESS		FUNCTION
2 SW1 OPEN	2 SW1 CLOSED	
C63FF	C67FF	Reserved
C63FE	C67FE	Reserved
C63FD	C67FD	Text Window Status
C63FC	C67FC	Turn Text Window On/Off
C63FB	C67FB	Reserved
C63FA	C67FA	Reserved
C63F9	C67F9	Board Type
C63F8	C67F8	Revision No.
C63F7	C67F7	
|	|	Reserved
C6314	C6714	
C6313	C6713	CMD List Offset 2
C6312	C6712	CMD List Offset 1
C6311	C6711	Self-test Flags
C6310	C6710	DMA Flag
C630F	C67BF	Expand Mode Status Flag
C630E	C67BE	CX/CA Status Flag
C630D	C670D	Emulator Status Flag
C630C	C670C	Emulator Control Flag
C630B	C670B	Emulator Strap Flag
C630A	C67BA	Reserved
C6309	C67B9	Reserved
C6308	C6708	Error Enable Flag
C6307	C6707	Warm Restart Flag
C6306	C6706	Cold Restart Flag
C6305	C6705	Error FIFO Read Pointer
C6304	C6704	Error FIFO Write Pointer
C6303	C6703	Input FIFO Read Pointer
C6302	C6702	Input FIFO Write Pointer
C6301	C6701	Output FIFO Read Pointer
C6300	C6700	Output FIFO Write Pointer
C62FF	C66FF	
|	|	Error FIFO
C6200	C6600	
C61FF	C65FF	
|	|	Read Back FIFO
C6100	C6500	
C60FF	C64FF	
|	|	Command FIFO
C6000	C6400	

Table 3.1: Communications Block Memory Map

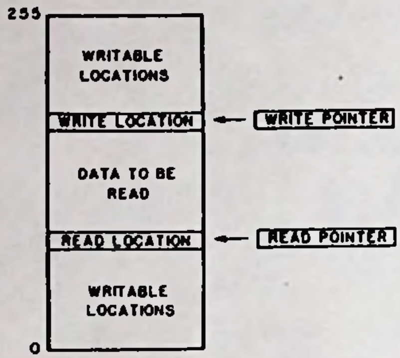


Figure 3.3: FIFO Pointer Protocol

write to the location immediately preceding the current read position. You may, however, read the location immediately preceding the current write position.

The preceding rules allow the user to use the values in the pointers to determine how full a particular FIFO is at any point in time. If the read and write pointers for a FIFO have the same value, the FIFO is empty. If the write pointer is one less than the read pointer (modulo 256) the FIFO is full. Figure 3.3 illustrates how the FIFO pointer protocol functions.

To access the FIFO's use the following procedures:

COMMAND FIFO WRITE

1. Read the values of the read and write pointers.

   (a) If $(writepointer + 1) MOD 256 = readpointer$ loop at step 1 (FIFO is full).

   (b) If $(writepointer + 1) MOD 256 \neq readpointer$ continue to step 2.

2. Write command byte to location pointed to by *writepointer*.
3. Increment *writepointer* (MOD 256).
4. Loop to step 1 until all command bytes are written to FIFO.

ERROR OR READ BACK FIFO READ

1. Read the values of the read and the write pointers.
   (a) If $writepointer = readpointer$ stop (FIFO is empty).
   (b) If $writepointer \neq readpointer$ continue to step 2.
2. Read byte at location pointed to by the *readpointer*.
3. Increment the *readpointer*.
4. Loop to step 1.

The HLGE uses complementary procedures when it reads the Command FIFO, writes to the Read Back FIFO, and writes to the Error FIFO.

3.3.2 The Control Block

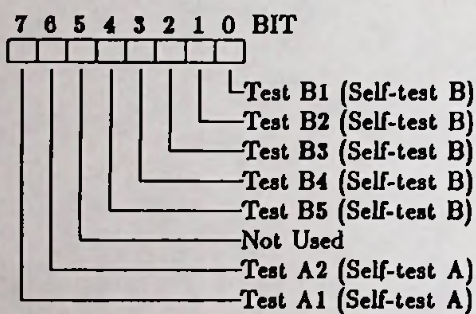
The control block consists of various locations in the communications area that are used to pass specific information between the board and the user.

Table 3.2 describes the various locations in the control block by giving the offset of each location from the base of the communications area, the user access type, and an explanation of how the location is used.

For the most part, the information transferred in these locations is either 1 or 0 and the explanations in Table 3.2 are all that you need; however, the data passed in the self-test status location is more complicated and requires further explanation.

The PG-640A has two self-tests: Self-test A and Self-test B. Self-test A is run at power up. Whether or not Self-test B is run depends on the state of the the self-test switch (switch 4). If the switch is on, the PG-640A runs Self-test B on power up and whenever a cold restart is issued. Self-test B reports in bits 0-4 of the self-test status location, and Self-test A reports in bits 6 and 7 of the self-test status location. All of the bits in the self-test status location are initially set to 0 and are set to 1 as the corresponding test starts, if a test fails the PG-640A clears the corresponding bit to 0.

The following diagram and text explain the functions of the individual bits in the self-test status location.



Bit 0: Test B1. This is the first test in the Self-test B sequence, and tests RAM from the start of the main buffer to the start of the communications area.

Bit 1: Test B2. This is the second test in the Self-test B sequence and tests that the PG-640A's CPU has access to the ACRTC.

Bit 2:	Test B3. This is the third test in the Self-test B sequence and tests that ACRTC can read and write the VRAM. Errors are indicated by pixels remaining visible on the screen.
Bit 3:	Test B4. This is the forth test in the Self-test B sequence and tests that the CPU can read and write the VRAM.
Bit 4:	Test B5. This is the fifth and final test in the Self-test B sequence and tests that the CPU can read and write the communications area FIFO's. This test assumes that the system CPU is reading and writing to the FIFO's. This test will not stop on its own; the user must write a non-zero value to the Warm Restart location (offset 307) to terminate the test.
Bit 6:	Test A2. This is the second and final test in the Self-test A series and tests the RAM stack area.
Bit 7:	Test A1. This is the first test in the Self-test A series and does a checksum test on the PG-640A's ROM.

Control Block Locations			
NAME	**OFFSET**	**ACCESS**	**DESCRIPTION**
Cold Restart	**306**	**R/W**	**Write a 1 to this bit to reset the board. The on-board CPU will write a 0 to this bit when the reset operation is complete.**
Warm Restart	**307**	**R/W**	**Write a 1 to this byte to halt command list execution, DMA transfers, and Self-test B, and to reset the FIFO pointers to 0. The on-board CPU writes a 0 to this byte when the halt operation is complete.**
Error Report	**308**	**R/W**	**Write a 1 to this byte to enable error reports to the error FIFO. Write a 0 to this byte to disable error reports. Read this byte to see whether error reports are enabled or disabled.**
Emulator Strap Status	**30B**	**R**	**The on-board CPU writes one to this byte if the emulator enable switch is enabled. It writes a zero to this byte if the emulator switch is not set.**
Turn Emulator On/Off	**30C**	**W**	**Write a 1 to this byte to turn on the CGE. Write a 0 to this byte to turn off the CGE. This bit does the same thing as the DISPLA command.**
Emulator On/Off Status	**30D**	**R**	**The on-board CPU writes a 1 to this byte when the CGE is on. It writes a 0 to this byte when the CGE is off.**
continued on next page			

continued from previous page

NAME	OFFSET	ACCESS	DESCRIPTION
DMA	310	R/W	The on-board CPU writes FF to this byte when a DMA operation is completed. It writes a 0 to this byte when a new DMA operation is in progress.
Self-test Status	311	R	The on-board CPU writes the status of the current self-test into this byte.
CMD List 1	312	R	The least significant byte of a word giving the offset of the most recently entered command in the command list currently being defined. The user may want to note this offset when entering commands that may have to be changed. When the time comes to change the commands, he can use the offset in the CLMOD command. The most significant byte of the offset word is in byte 313.
CMD List 2	313	R	The most significant byte of the command list command offset. The least significant byte is in byte 312.
Version	3F9	R	The version number of the board firmware. If you have to telephone our applications engineers for assistance please have this number and the revision number at hand.

continued on next page

continued from previous page			
NAME	OFFSET	ACCESS	DESCRIPTION
Revision	3F8	R	0 = PG-1280, PG-1280A, PG-1280A/8 1 = PG-640 2 = PG-640A 3 = not used 4 = SM-640
Window Switch	3FC	W	write a non-zero value to this byte to turn on the text window. Write 0 to this byte to turn off the text window.
Window Status	3FD	R	A non-zero value in this byte indicates that the text window is enabled. A zero value indicates that the text window is disabled.

Table 3.2: Control Block Locations

3.3.3 Setting System Flags

The user can read the current values of several system parameters using the FLAGRD command. This command has the following format:

FLAGRD␣flag

where flag selects one of the flags shown in Table 3.3. The current value of the flag is written to the read back buffer. Another command, the RESETF command, resets all flags to the default values listed in Appendix B. The system automatically resets flags to these values on power-up or after a reset of the board. The command format is the following:

RESETF␣

3.3.4 The WAIT Command

The WAIT command is provided as an easy way to suspend command execution for a specified length of time. The command format is as follows:

WAIT␣frames

where frames is the delay in $\frac{1}{60}$ seconds. You can have a delay of up to 18 minutes.

Flag	Name	Type of Value
1	AREAPT	16 Ints
2	CLIPH	1 Char
3	CLIPY	1 Char
4	COLOR	1 Char
5	DISPLA	1 Char
6	DISTAN	1 Real
7	DISTH	1 Real
8	DISTY	1 Real
9	FILMSK	1 Char
10	LINFUN	1 Char
11	LINPAT	1 Int
12	MASK	1 Char
13	MDORG	3 Reals
14	2-D current point	2 Reals
15	3-D current point	3 Reals
16	PRMFIL	1 Char
17	PROJCT	1 Int
18	TANGLE	1 Int
19	TJUST	2 Chars
20	TSIZE	1 Real
21	VWPORT	4 Ints
22	VWRPT	3 Reals
23	WINDOW	4 Reals
24	transformed 3- D current point	3 Reals
25	free memory	1 Int
26	current position of XHAIR	2 Ints
27	2-D position of XHAIR	2 Reals
28	Screen Current Point	2 Ints
29	free memory	1 Real*
30	TWVIS	1 Char
31	TWPOS	6 Ints
32	TSTYLE	1 Char
33	TASPCT	1 Real
34	TCHROT	1 Int
41	COLMOD	1 Char
42	BCOLOR	1 Char

* This value is treated as a double precision integer

Table 3.3: System Flags

3.4 Transforms

The HLGE displays images on a video screen using a physical coordinate space of 1280 pixels by 960 pixels, and this is the maximum resolution of the displayed image. The user, however, draws his images in one of two virtual coordinate spaces with a much higher resolution. The HLGE uses transforms to map images in the virtual coordinate space into real screen coordinate space in such a way that maximum resolution is always maintained. For example, a user could use the HLGE to draw a very detailed picture of a tree. When the whole tree was displayed the screen resolution would only allow larger details such as branches, the trunk, and the form of the tree to be seen. However, if the picture in the virtual coordinate space was detailed enough the user could zoom in on one leaf and see it in detail.

The two virtual coordinate spaces are a 2D coordinate space with two axes (x and y) and a 3D coordinate space with 3 axes. The resolution of both coordinate spaces is from -32768.0000 to +32767.9999 on each axis. Figure 3.4 shows the two virtual coordinate spaces and illustrates their relation to each other and the screen space.

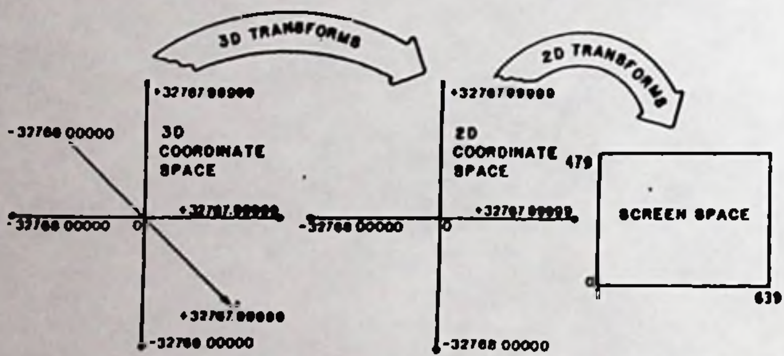


Figure 3.4: Coordinate Spaces

3.4.1 2D Transforms

The 2D coordinate space uses Cartesian coordinates with the origin in the centre and coordinates going from -32768.0000 to +32767.9999 on each axis. The user utilises the WINDOW and VWPORT commands to map a rectangular section of this coordinate space onto the display. The WINDOW command takes the following format:

WINDOW␣x_1␣x_2␣y_1␣y_2

where the parameters x_1 and y_1 form one coordinate pair, and x_2 and y_2 form another. These coordinate pairs specify two opposing corners of a rectangular section of the 2D coordinate space. This rectangular section is referred to as a window and any image drawn in it is mapped into the current view port–a rectangular section of the screen space. If the user does not specify a window, the HLGE defaults to a 640 by 480 window centred on the the coordinate space origin.

The VWPORT command defines the view port, and has the following format:

VWPORT␣x_1␣x_2␣y_1␣y_2

where coordinate pairs x_1, y_1 and x_2, y_2 specify the opposing corners of a rectangular section. In this case, however, the coordinates must be given in screen coordinates rather than 2D coordinate space coordinates. As indicated in Figures 3.4 and 3.5, the screen coordinate space has its origin in the lower left corner, has 640 (0-639) points on the x axis, and 480 (0-479) points on the y axis. If the user does not specify a view port the HLGE defaults to a view port that includes the whole screen.

The command string that defines the window and view port in Figure 3.1 of Section 3.1 illustrates how the user can define different windows and view ports.

3.4.2 3D Transforms

The user draws 3D pictures in the 3D coordinate space. When he draws them, their position, size, and how they are viewed are determined by the current state of a number of transforms. Modeling transforms determine the scale (size), rotation, and position (translation) of the picture within the coordinate space. Viewing transforms determine the position of the viewer and his direction of view with respect to the coordinate. The clipping function's hither and yon clipping planes slice off the front and the back of the picture if that is required. 3D to 2D transforms map the 3D image into the 2D coordinate space, establishing the distance of the viewer from the image and his angle of view (perspective). Once the image is in the 2D coordinate space it is mapped onto the screen by the window and view port transforms that we have already described during the description of 2D drawing.

The 3D transforms allow the user to manipulate the graphic object and the viewer. For example, let us assume that the user has a routine to draw a house. If he wants 2 houses in different parts of the 3D coordinate space, he sets up the translation transform for one position then runs the routine to draw the first house. Then he sets up the translation transform for another position and runs the same program again to draw the second house.

The diskette that you received with the PG-640A contains a file named house.pga. It contains a list of commands that draw a house. Figure 3.5 shows how that house is displayed when the HLGE uses its default parameters for the 3D transforms. In this section we use several examples to show how different transform settings affect this house. You can easily use the PG-640A monitor program to input the example code to the HLGE so that you can see the results on the screen. If you wish to do so, execute the following procedure. It loads the monitor, and puts the house description into a command list (command lists are described in Section 3.9).

1. Put 82960A–12001 diskette into drive A.

2. Type "A:PG-MON(carriage return)" to load monitor program.

3. Press the F6 key then the F8 key to enable the HLGE display with a text window at the bottom.

4. Press the F1 key then type "A:house.pga". Instructions in the file will put the house routine into command list number 100.

5. Now to send command strings to the Command FIFO you just type them as they are shown in the examples, using a delimiter such as space, comma, or carriage return in place of the ␣ characters.

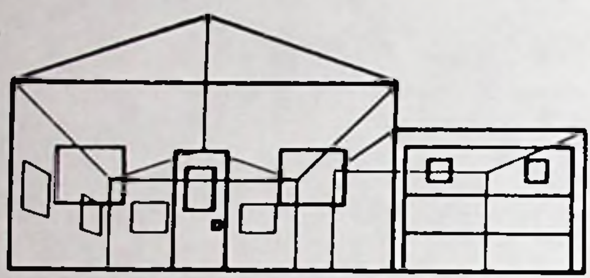


Figure 3.5: Default House

Modeling Transforms

The modeling transforms are the first transforms to affect the house when it is being drawn. There are 3 different modeling transforms–the translation transform, which moves objects in the coordinate space by offsetting their coordinates as they are drawn; the rotation transform which rotates the object around each of the three axes; and the scaling transform, which determines the size of the object.

The HLGE performs the modeling transforms by multiplying each x,y,z coordinate set in the graphic object's description by a modeling matrix (M). The user can load the modeling matrix directly by using the MDMATX command, or he can modify various aspects of it by using 5 modeling commands (MDTRAN, MDSCAL, MDROTX, MDROTY, and MDROTZ). When the HLGE receives a modeling command it temporarily creates a submatrix corresponding to the command function, multiplies it by the modeling matrix then discards it, leaving a modified modeling matrix. The submatrices created by the modeling commands are: the translation matrix (T), the scaling matrix (S), and the 3 rotation matrices (R_x, R_y, R_z).

The submatrices are multiplied by the master in the order that their corresponding commands are received. Since matrix multiplication is not commutative this means that the order that you send your modeling commands in affects the form of the master matrix.

At reset the modeling matrix is a unity matrix. You can return it to unity at any time by issuing the MDIDEN command. You can read the current modeling matrix by issuing a MATXRD command with a parameter of 1.

The rotation and scaling transforms require an origin. In rotation operations the origin is the point around which the graphic object turns. In scaling operations it is the point at the centre of the expansion or contraction. The MDORG command is used to specify the modeling origin; its format is as follows:

MDORG␣ox␣oy␣oz

The parameters are an x,y,z coordinate set that specifies the modeling origin with respect to the graphic object's original coordinates. For example, our house is centred on the coordinates 0, 50, 0. To specify this point as the modeling origin we would pass the following ASCII string to the HLGE:

MDORG␣0␣50␣0␣

Use the MDROTX, MROTY, and MDROTZ commands to rotate graphic objects. The command formats are as follows:

MDROTX␣deg
MDROTY␣deg
MDROTZ␣deg

where deg is the number of degrees of rotation to be performed. The HLGE calculates the sin and cos of these angles and enters them into the rotation matrices as shown below:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & sin\theta & 0 \\ 0 & -sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y = \begin{pmatrix} cos\theta & 0 & -sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z = \begin{pmatrix} cos\theta & sin\theta & 0 & 0 \\ -sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The HLGE uses the right-hand rule for rotation. This rule defines the x, y, and z axes to be in the directions that the first finger, second finger, and thumb of a right hand will point in if they are held at right angles to each other (see Figure 3.6). The origin of these axes is at the modeling origin, and the object rotates around the axes as illustrated in Figure 3.6.

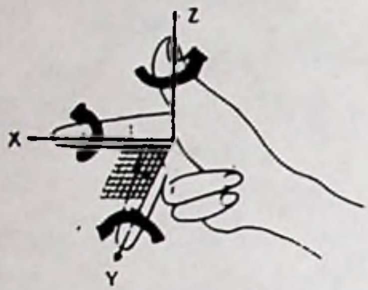


Figure 3.6: Rotation Direction

The default modeling transforms use identity matrices that do not affect the graphic object. There will be situations where the user will want to get back to this identity state–i.e. to reset the transforms. The HLGE provides the MDIDEN command for this purpose. In our examples of modeling transforms we use this command to reset the transforms so that you can see the effect of one transform without interference from others.

The following command string resets the modeling transforms, sets the modeling origin, sets up the rotation transforms, then runs command list number 100. If command list 100 has the house routine from house.pga (see page 3-18) then the result will be as shown in Figure 3.7.

MDIDEN␣
MDORG␣0␣50␣0␣
MDROTX␣45␣
MDROTY␣45␣
MDROTZ␣45␣
CLRUN␣100␣

The MDSCAL command is used to scale graphics objects. Its format is as follows:

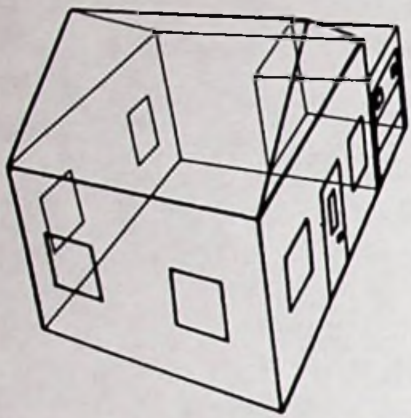


Figure 3.7: Rotation Example

MDSCAL␣sx␣sy␣sz

where sx, sy, and sz are entries in the scaling transform as follows:

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The result of this is that when it is drawn, the size of the graphic object along each axis is multiplied by the corresponding parameter. For example, if sx is 2 the graphic object is expanded by 2 times along its x axis. If sy is .5, the graphic object's size along the axis is halved.

The MDTRAN command is used to offset a graphic object from its as sent coordinates to a different position. The command format is as follows:

MDTRAN␣tx␣ty␣tz

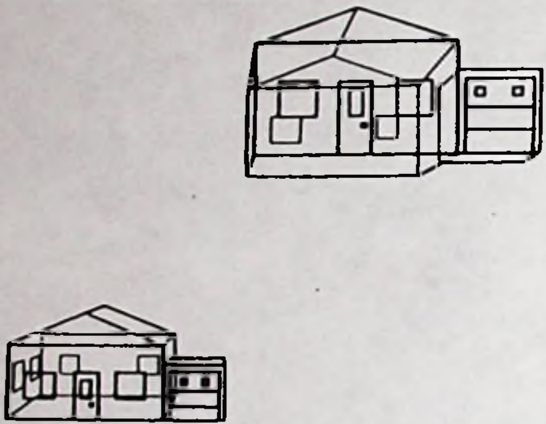


Figure 3.8: Translation Example

where the parameters are values to be added to the x, y, and z as sent coordinates. The HLGE enters these values into its translation matrix as follows:

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_y & 1 \end{pmatrix}$$

The following command string makes 2 half-size copies of our house in different positions as shown in Figure 3.8.

```
CLEARS␣0␣
MDIDEN␣
MDSCAL␣.5␣.5␣.5␣
MDTRAN␣50␣40␣50
CLRUN␣100
MDTRAN␣-150␣-150␣-150␣
CLRUN␣100␣
```

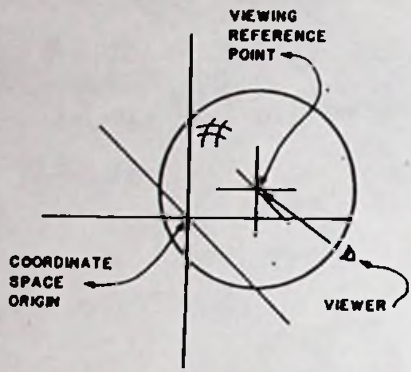


Figure 3.9: Viewing Reference Point

Viewing Transforms

The HLGE uses a viewing transformation to position the viewer with respect to the coordinate space. It establishes a viewing reference point, which is mapped into the centre of the view port, and it positions the viewer somewhere on the surface of a sphere that has its centre at the viewing reference point, as illustrated in Figure 3.9. The radius of the sphere and the amount of the coordinate space that is mapped into the view port are determined by the 3D to 2D transformation, which is described further along. Our examples up to this point have used the default viewing reference point and viewer position–the viewer reference point is in the centre of the coordinate space and the viewer is looking down the positive Z axis.

As is the case with the modeling transform, the viewing transform uses a master matrix (the viewing matrix). The user can load the viewing matrix directly with the VWMATX command, or he can alter various aspects of it with the viewing commands (VWRPT, VWROTX, VWROTY, VWROTZ). The viewing commands function like the modeling commands in the respect that they set up submatrices that are multiplied by the viewing matrix then discarded; and like the modeling commands, the order that they are issued in has an effect on the final view. The user can read the current viewing matrix at any time by issuing the MATXRD command with a parameter of 2.

The VWIDEN command is similar to the MDIDEN command, and we use it in our examples to reset the viewing matrices so that other matrices don't affect the matrix that we are using in the example.

The VWRPT command is used to specify the viewing reference point. The command format is as follows:

VWRPT␣x␣y␣z

where x, y, and z are a coordinate set specifying the 3D coordinate space point that the user wants in the centre of the viewer's field of view (i.e. the centre of the view port).

The VWROTX, VWROTY, and VWROTZ commands determine the position of the viewer on the viewing sphere. The command formats are as follows:

VWROTX␣deg
VWROTY␣deg
VWROTZ␣deg

where deg is the number of degrees the viewer is to move around the corresponding axis in the direction indicated in Figure 3.6. Note that the axes used by these commands are parallel to the coordinate system axes but that their origin is at the viewing reference point. The HLGE takes the sin and cos of the angle and enters them into the viewing rotation matrices as follows:

$$VWR_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$VWR_y = \begin{pmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$VWR_z = \begin{pmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The following string clears the display, resets modeling and viewing transforms, sets the viewing reference point to {0,50,0} (the default value), moves the viewer's position to 90 degrees up from the xz plane in the yz plane, then runs command list number 100 to draw our house. Figure 3.10 shows the result.

```
CLEARS␣0
MDIDEN␣
VWIDEN␣
VWRPT␣0␣50␣0␣
VWROTY␣90␣
CLRUN␣100␣
```

Hither and Yon Clipping

The WINDOW command, which we have already examined, clips the sides of the picture to frame the part of the coordinate space that we want

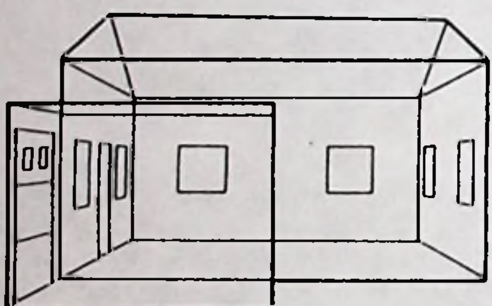


Figure 3.10: Viewing Transform Example

to look at. The HLGE also has commands to clip everything in front of a given point and every thing behind a given point. The operation is referred to as hither and yon clipping, and to do it you must specify clipping planes, then set clipping enable flags. The clipping planes are set with the following commands:

DISTH␣dist
DISTY␣dist

where dist in the DISTH command is the distance from the viewing reference point to the hither (foreground) clipping plane, and dist in the DISTY command is the distance from the viewing reference point to the yon (background) clipping plane. The polarity of the parameter values are the opposite of what the user might think. That is to say negative value are closer to the viewer than positive values.

The commands that actually enable or disable clipping have the following format:

CLIPH␣flag
CLIPY␣flag

where flag is 0 or 1. A 1 enables clipping; a 0 disables clipping. As the last letter in the command keywords suggest, CLIPH controls hither clipping and CLIPY controls yon clipping.

The following string clears the screen, sets the clipping planes and flags, then runs command list 100. The result is a house with the front and back clipped off as shown in Figure 3.11.

```
CLEARS␣0␣
VWRPT␣0␣0␣0␣ VWIDEN␣
DISTH␣-90␣
DISTY␣90␣
CLIPH␣1␣
CLIPY␣1␣
CLRUN␣100␣
```

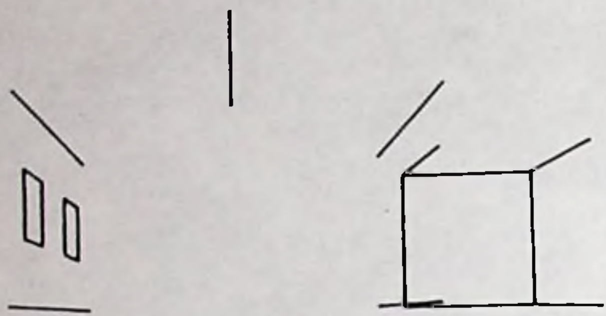


Figure 3.11: Clipping Example

To appreciate applications of the hither and yon clipping function imagine that our graphic object is not a simple line drawing house but a complex gear box. If such was the case we would now be able to examine its inner workings.

Clipping should be disabled when it is not required since it requires extra calculations on the part of the HLGE, with the result that performance is decreased.

3D to 2D Projection

In addition to the VWROT commands and the hither and yon clipping parameters there are 3 other factors that affect the appearance of a 3D object on the screen: the distance of the viewer from the object, the projection angle, and the current window position.

The HLGE projects the area around the viewing reference point onto the 2D coordinate space. The size of this area depends on 2 parameters: the viewing angle and the viewing distance as illustrated in Figure 3.12. The viewing angle specifies the number of degrees on the horizontal axis and the vertical axis of the viewer's field of view (default is 60°), centred

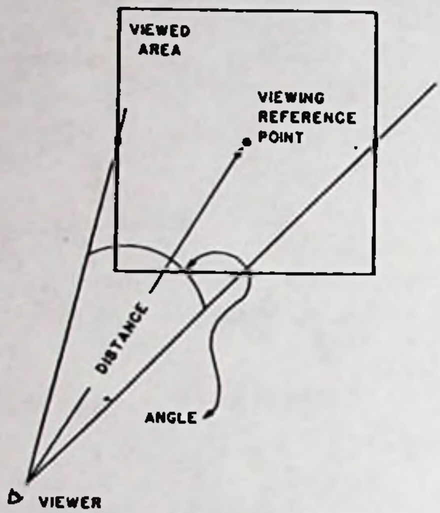


Figure 3.12: Viewing Angle and Viewing Distance

on the viewing reference point, and the viewing distance is the distance that the user is from the viewing reference point (default is 500). In an analogy with a camera, the viewing angle would be determined by the type of lens (wide angle, narrow angle, etc.) and the viewing distance would be determined by the distance of the camera from the subject. If the viewing angle is larger, more of the 3D coordinate space is projected into the window. Likewise, if the viewer moves farther away from the viewing reference point more of the 3D coordinate space is projected into the window.

The DISTAN command is used to specify the viewing distance. Its format is as follows:

DISTAN␣dist

where dist is the distance (specified in 3D coordinate point units) of the viewer from the viewing reference point.

The PROJCT command is used to set the viewing angle and the type of perspective that is to be used for the projection. Its format is as follows:

PROJCT␣angle

where angle is the number of degrees (horizontal and vertical) in a field of view with the viewing reference point at its centre. An angle of 0° is a special case. It specifies a orthographic parallel (non-oblique) projection. When this type of projection is used the viewing distance has no effect on the size of the picture.

The HLGE uses the following formulas to convert 3D coordinates to 2D coordinates:

$$x_{2D} = \frac{1}{\text{dist} - z_{vw}} \times x_{vw} \times \frac{windowdiagonal}{2 \times tan\frac{\text{angle}}{2}}$$

$$y_{2D} = \frac{1}{\text{dist} - z_{vw}} \times y_{vw} \times \frac{windowdiagonal}{2 \times tan\frac{\text{angle}}{2}}$$

The HLGE does not automatically map the view into the current window; however, the transformations used do guarantee that the viewing reference point is mapped to the origin of the 2D virtual space. So if your window includes the 0,0 coordinate, you will see your viewing reference point on the screen, and you can adjust the window position as required to see any part of the object that is not in the window.

Window size, however, makes no difference to all projections except the 2D and 3D orthographic cases. That is to say, the window size is ineffective in displays with PROJECT angles greater than 0°.

This is because the 2D virtual coordinates from the equations above are next passed through another transform to bring them to screen coordinates. This final transform has the following form:

$$x_{scrn} = (x_{2d} - x_{windowleft}) \times \frac{(viewportsize)}{windowsize} + x_{viewportleftedge}$$

Substituting for x_{2d} and separating out the constant terms leaves:

$$x_{scrn} = \frac{1}{dist - z_{vw}} \times \frac{windowdiagonal}{2 \times \tan\frac{angle}{2}} \times \frac{viewportsize}{windowsize} + K$$

If the current window is close to being square, the *windowdiagonal* is close enough to the *windowsize* in both the x coordinate and y coordinate transforms so they will cancel out for all practical purposes.

Also note that since *dist* is in the denominator, larger distances give smaller screen images. Similarly, since the tangent of half the projection angle is in the denominator, when the angle is bigger, the screen image is smaller (especially for large angles).

The following command string uses the 3D to 2D transform to zoom in on the house as shown in Figure 3.13. The 3D to 2D transform converts the 3D coordinates to 2D coordinates then the window to view port mapping converts the 2D coordinates to screen coordinates.

```
CLEARS␣0␣
MDIDEN␣
VWIDEN␣
CLIPH␣0␣
CLIPY␣0␣
DISTAN␣300␣
CLRUN␣100␣
```

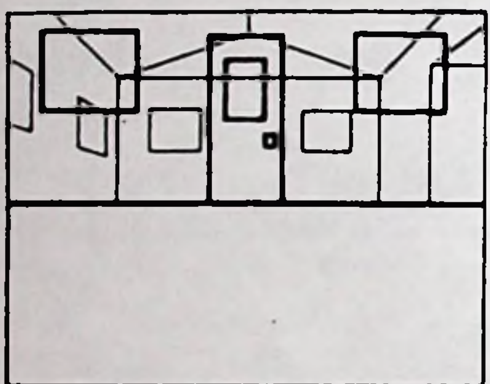


Figure 3.13: 3D To 2D Projection Example

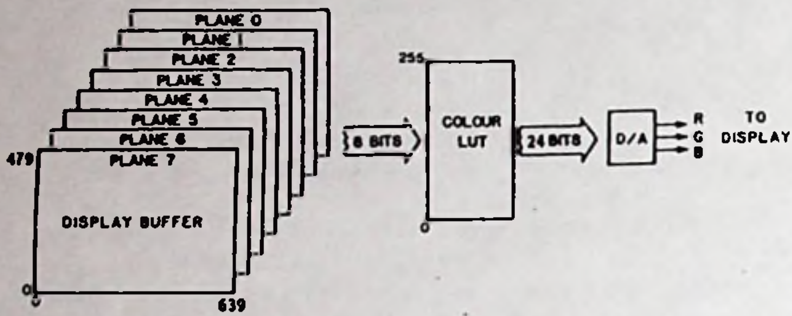


Figure 3.14: The Output Stage

3.5 Graphic Attributes

After the HLGE has performed all of the transforms described in the preceding section, the resulting image is drawn by loading 8-bit color indices into pixel locations in the display buffer. The display buffer is a 640 by 480 array of pixel locations that is mapped onto the display screen through a color lookup table. This lookup table determines the color that corresponds to each index. Figure 3.14 illustrates the relation of the display buffer to the screen.

When drawing an image in the display buffer, the color indices used depend on several graphics attributes. These attributes are: the current index, the current line style, the current drawing mode, and the current mask.

3.5.1 Drawing Mode

The current drawing mode affects all the other modes. There are five drawing modes: Replace, Complement, OR, AND, and XOR.

The user selects the mode with the following command:

LINFUN␣mode

where mode is a Char from 0 through 4.

When Replace Drawing Mode is active, lines and fills are drawn by replacing the contents of pixel locations with the current index.

When Complement Drawing Mode is active, the PG-640A draws lines and fills by complementing the current contents of pixel locations. For example, the default contents of the display buffers is index 0 in all pixel locations; in Complement Drawing Mode the PG-640A would draw a line on this background by changing the index of every pixel in the line to 255, since 255 (FF) is the complement of 0 (00). The advantage of this mode is that it allows individual graphic objects to be erased easily without affecting underlying graphic objects or the background. For example, to erase a line that was just drawn, we would merely redraw it, and it would be complemented back to what it was before. The disadvantage of Complement Drawing Mode is that the color displayed is affected by the underlying color.

The XOR Drawing Mode is a more general form of the Complement Drawing Mode and can be used for similar applications. It, however, allows more flexibility, since it XORs the current index with the current values of underlying pixels to obtain the new pixel values as a line is drawn. Drawing the same line twice in this mode results in no line, since the second line reverses the first.

The OR Drawing Mode ORs the current index with the current values in underlying pixels, and the AND Drawing Mode ANDs the current index with the current values in underlying pixels. The uses for these two drawing modes are less common; however, the experienced programmer should be able to put them to use in certain applications.

Note that all of the drawing modes interact with the PRMFIL command (refer to Section 3.7).

3.5.2 Color

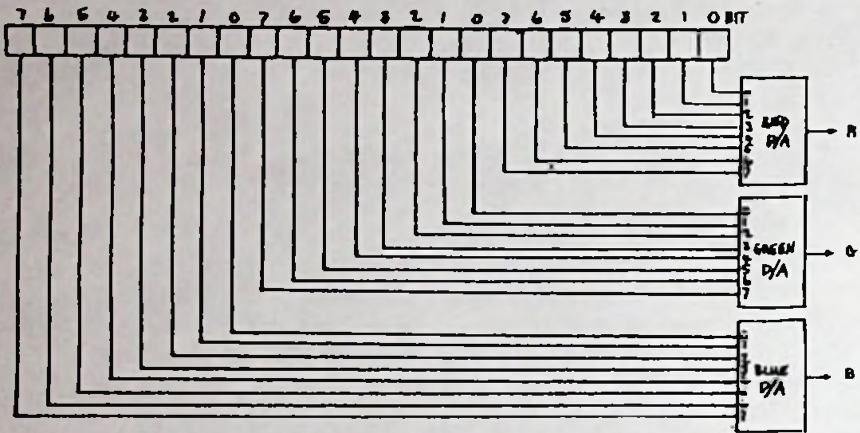


Figure 3.15: Lookup Table Bit Map

The user selects the current index by issuing the COLOR command, which has the following format:

COLOR␣index

where index is a value from 0 to 255. A color index is not a color in itself; it is the address of a location in the lookup table. As the display buffer is scanned, the value in each pixel location is sent to the lookup table. The lookup table provides three values to the digital to analog converter. These values are used to generate the three analog signals to drive the red, green, and blue guns of the color display. Each lookup table location has 24 bits that are mapped into the digital to analog converter (D/A) inputs as indicated in Figure 3.15.

Referring to Figure 3.15, you will see that there are 256 intensity values for each of the three primary colors. The color that appears on the screen depends on the combination of these values. For example, a lookup table value of FF FF 00 generates bright blue-green, 00 FF FF generates bright yellow, and 00 00 00 generates black.

The LUTX, LUT and LUTINT commands allow the user to load various color values into the lookup table. The LUTX and LUT commands write values into single lookup table locations, and the LUTINT command initializes the whole lookup table to one of several sets of predetermined values. The format of the LUTX command follows:

LUTX index r g b

where index is the index of a lookup table location, and r, g, and b are values from 0 to 255 specifying the intensity of the red, green, and blue elements respectively for that location. The LUT command is similar to the LUTX command except that only the four low bits are loaded into the four high bits of the lookup table entry. LUT is provided in order to maintain software compatibility with other MATROX products. For example, the following LUTX command string sets lookup table location 4 to bright yellow:

LUTX 4 255 255 0

The following LUT command string will put bright yellow into the lookup table location 4:

LUT 4 15 15 0

The LUTINT command has the following format:

LUTINT set

where set is a number specifying one of several sets of values to be loaded into the lookup table. Table 3.4 lists these sets and Appendix E gives their contents.

Set 0 has values that generate colors in the standard color cone used by graphic artists. The relationship between the color index and the color

SET	DESCRIPTION
0	Color-cone
1	2 surface
2	rrgggbbb
3	rrrggbbb
4	rrrgggbb
5	6-level rgb
253	Alternate saved LUT
254	Saved LUT 1
255	Saved LUT 2

Table 3.4: List Of Lookup Table Value Sets

that is generated by it is arbitrary. The values of the predefined lookup table can be found in Appendix E.

Sets 2 to 5 are arranged in such a way that there is a relationship between the format of the color index and the color that it generates. When Set 2, 3, or 4 is in the lookup table, the color index is divided into three binary numbers: a red number, a green number, and a blue number. The number of bits in each number depends on the lookup table set as shown below:

		76543210	bit
Set 2 index	=	rrgggbbb	
Set 3 index	=	rrrggbbb	
Set 4 index	=	rrrgggbb	

The value of these numbers determines the intensity of the red, green, and blue components of the color. The two-bit intensity values are related to the three-bit intensity values as shown in Table 3.5.

For example, if Set 2 is in the lookup table, index 63 (00111111) selects bright cyan. When Set 5 is in the lookup table, the relationship of the

VALUE		
2-BIT	3-BIT	INTENSITY
0	0	0
-	1	3
1	2	5
-	3	7
-	4	9
2	-	10
-	5	11
-	6	13
3	7	15

The high nibble of each color component contains the selected entry from the INTENSITY column; the low nibble of each color component is set to zero.

Table 3.5: 2-Bit/3-Bit Correspondence

index to the color selected is a follows:

$$index = (r \times 36) + (g \times 6) + b$$

where r, g, and b are intensity values from 0 through 5 for the color components of the selected color.

Set 1 has a special set of color values designed to provide two superimposed display surfaces. When Set 1 is in the lookup table, the index is divided into two subindices: ones in the low four bits select the underlying color and ones in the high four bits select the overlying color. Zeroes in the high four bits makes the foreground surface transparent, allowing the underlying surface to show through. Further on we explain how to use the MASK command to write to one surface or the other.

Sets 253, 254 and 255 load the lookup table with sets of lookup table values that the user has previously saved using the LUTSAV and LUTSTO commands. Note, however, that Set 253 alternately loads the lookup table with the specified lookup table values. The LUTSAV command, which has no parameters, saves the current contents of the lookup table to a special on-board memory buffer reserved for Set 255. The LUTSTO is similar to the LUTSAV command except that it allows two sets of lookup table contents to be stored. It has a parameter which specifies that the current lookup table be saved to Set 255 or to a second buffer reserved for Set 254. Subsequent LUTSAV and LUTSTO commands overwrite any lookup table sets that may have already been saved in the lookup table buffers.

The user can read the contents of a lookup table location by issuing the LUTRD command or the LUTXRD command. These commands have the following formats:

LUTRD␣index or LUTXRD␣index

where index is a value from 0 to 255 specifying the lookup table location to be read. The HLGE will copy the contents of the specified lookup table location into the Read Back FIFO.

3.5.3 Line Texture And Blinking Pixels

Lines can have texture as well as color. The texture is determined by the current line pattern, which the user sets with the LINPAT command. LINPAT has the following format:

LINPAT␣pattern

where pattern is a word with the line bit pattern. For example, the decimal value 61680 is equivalent to the binary value 1111000011110000. Issuing the following command:

LINPAT␣61680␣

causes lines to be drawn with four pixels in the current index alternating with four transparent pixels that allow the underlying index to show through (1 = current index, 0 = transparent).

Under certain conditions, primitives may generate both a background and a foreground. When a patterned line is drawn, for example, the pattern is made up of a foreground and a background, a character cell has a foreground and a background, and any of the commands that produce filled areas produce a foreground and a background if the fill is in the form of a pattern. In such a case, using the COLMOD command specifies the color mode that determines whether the background is transparent or is the color last specified by the background color index. The background color is specified by the BCOLOR command. Note that the color mode affects the LINPAT command.

The COLMOD command has the following format:

COLMOD␣mode

where mode is a Char equal to 0 or 1. When parameter mode is 0, the background is set to the color specified by the BCOLOR command; when mode is 1, the background is transparent.

The BCOLOR command has the following format:

BCOLOR␣index

where index is a Char from 0 to 255 specifying the background color index.

Color indices can also be given a blink attribute to make them blink with the BLINKX command. It has the following format:

BLINKX␣index␣red␣green␣blue␣ontime␣offtime

where index specifies the lookup table index to blink. The parameters red, green, and blue are values from 0 through 255 that compose the color that the index is to blink to. The time that the affected pixels will be the blink color is specified by ontime in $\frac{1}{60}$ seconds. The time that the pixels are their normal color is set by offtime in $\frac{1}{60}$ seconds. A similar command, BLINK, is provided for software compatibility with other MATROX products.

If you want to stop all blinking set by BLINK and BLINKX commands simply use the SBLINK command. It has the following format:

SBLINK␣

All pixels will be assigned their original color.

3.5.4 Masking Bit Planes

If you refer to Figure 3.14 again you will note that the display buffer is composed of eight bit planes – one for each of the eight bits in the color index. The MASK command can mask off specified bit planes so that they cannot be overwritten when the HLGE draws in the display buffer. The MASK command has the following format:

MASK␣planemask

where planemask is an eight bit value (0-255). Zeroes will prevent access to their corresponding bit planes and ones will permit access. For example, the value 240 (11110000) masks access to the four least-significant bit planes.

The mask allows the display buffer to be divided into different display surfaces. This is particularly useful when used in conjunction with the Set 1 lookup table values. For example, to superimpose the layers of artwork for a multilayer printed circuit board, the user could draw one layer with the four lower bit planes masked off, and then mask off the high four bits and draw the second layer. The image already on the lower bit planes would not be affected.

3.6 Primitives

The HLGE maintains 2 current points: a 2D current point and a 3D current point. These points are analogous to the position of a pen on a piece of paper. Just as you would move a pen over paper to draw an image, you move the 2D current point to draw an image in the 2D coordinate space and you move the 3D current point to draw an image in the 3D coordinate space. The commands that allow you to move the current point are called graphic primitives, and are explained in this section.

There are 2 main categories of graphics primitives: those that are used in the 2D coordinate space and those that are used in the 3D coordinate space. The keywords for commands in the 2 groups are similar. The 3D keywords are distinguished from their 2D counterparts by having a 3 as the last character of their keywords. Note, however, that not all the 2D primitives have 3D counterparts.

In this section we describe all of the 2D primitives then describe the 3D primitives. In both cases we use a running example to illustrate how the commands work. The reader is invited to use the PG-640A monitor program to input the commands in these examples to the HLGE (see Subsection 2.5.2 or Appendix D for instructions on how to use the PG-640A monitor program).

3.6.1 2D Primitives

When you draw on a piece of paper your pen is not always on the paper. You need to lift it and move it from time to time to start new lines. The same is true for drawing with the HLGE. The MOVE and MOVER commands are provided to move the pen in the 2D coordinate space without drawing. The format of the MOVE command is as follows:

MOVE␣x␣y

where x and y are reals specifying a coordinate pair in the 2D coordinate space. When it receives this command, the HLGE moves the current point to the indicated point without drawing.

The format of the MOVER command is as follows:

```
MOVER␣dx␣dy
```

where dx and dy are reals specifying the distance that the current point is to be moved on the x and y axes respectively. Note that the 'R' termination on this and other command keywords identify the command as using relative coordinates.

If you want to draw a dot at the current point, you issue a POINT command. It draws a point in the current index or complemented index depending on the current drawing mode, as is the case with all graphics primitives. The POINT command has no argument.

To draw a single straight line (also called a vector) you issue either a DRAW or DRAWR command. The parameters for these commands are the same as those for the MOVE and MOVER commands and the effect is the same with the difference that the DRAW and DRAWR commands draw lines from the old current point to the new current point.

The following example clears the screen then moves the current point to the centre of the coordinate space and draws a point. Then moves the current point again (using relative coordinates this time) and draws two lines–one using relative coordinates and one using absolute coordinates. The result is illustrated in Figure 3.16.

```
COLOR␣24␣
CLEARS␣0␣
MOVE␣0␣0␣
POINT␣
MOVER␣0 -10␣
DRAWR␣-20␣-5␣
DRAW␣0␣60␣
```

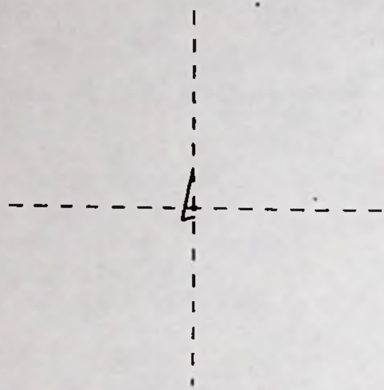


Figure 3.16: Example: Moves, Lines, And Points

The HLGE has several graphic primitives that use a sequence of straight lines to draw polygons. These include the RECT, RECTR, POLY, and POLYR commands. RECT and RECTR draw rectangular polygons. RECT uses absolute coordinates, and RECTR uses relative coordinates. The format for the RECT command is as follows:

RECT␣x␣y

where x and y are reals specifying a coordinate pair at one corner of the rectangle to be drawn. The HLGE assumes that the opposite corner on the diagonal is the current point and draws a rectangle based on the two corners. The current point does not move.

The format of the RECTR command is as follows:

RECTR␣dx␣dy

where dx and dy are reals indicating the distance along the x and y axes respectively from the current point to the corner opposite on the diagonal of the rectangle to be drawn.

The POLY and POLYR commands draw general polygons. The format of the POLY command is as follows:

```
POLY␣npts␣x1␣y1␣x2␣y2 ... xn␣yn
```

where npts is a value of 0-255 giving the number of corners in the polygon to be drawn, and the rest of the argument is a series for coordinate pairs specifying the positions of the corners in the order that they are to be drawn. When the HLGE receives this command it draws the polygon specified and leaves the current point at its original position.

The POLYR command is similar except that instead of absolute coordinates (relative to the origin of the coordinate space) it uses coordinates relative to the current point in effect when the command is issued.

The following command string draws a rectangle using absolute coordinates, a rectangle using relative coordinates, a 6-sided polygon using relative coordinates, and a 6-sided polygon using absolute coordinates, in that order. The result is shown in Figure 3.17 combined with the result of the previous example.

```
MOVE␣20␣-50␣
RECT␣-20␣-60␣
MOVE␣60␣180␣
RECTR␣-120␣40␣
MOVE␣50␣180␣
POLYR␣6␣0␣0 ␣60␣-160 ␣-30␣-280 ␣-70␣-280 ␣-160␣-160 ␣-100␣0␣
POLY␣6 ␣30␣-55␣20␣-65␣ -20␣-65␣-30␣-55␣-20␣-45 ␣20␣-45␣
```

The HLGE can draw curved lines as well as straight lines, and has 3 commands that do so–CIRCLE, which draws a circle; ARC, which draws an arc of a circle, and ELIPSE, which draws an ellipse. The format of the CIRCLE command is as follows:

```
CIRCLE␣radius
```

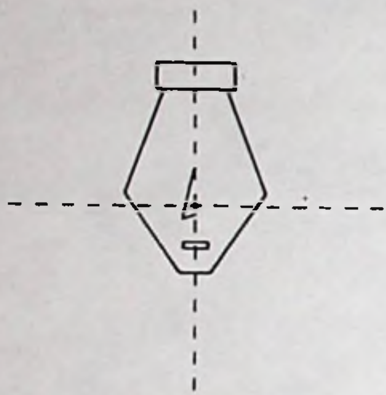


Figure 3.17: Example: Polygons

where radius is a Real specifying the radius of the circle to be drawn and the circle's centre is at the current point.

The format of the ARC command is as follows:

ARC␣radius␣deg0␣deg1

where radius is a Real specifying the size of the circle on which the arc is drawn, deg0 is an Int giving the starting angle, and deg1 is an Int giving the ending angle. The starting angle and ending angle are measured in degrees counterclockwise from the positive x axis of the circle on which the arc is drawn.

The ELIPSE command has the following format:

ELIPSE␣xradius␣yradius

where xradius is the distance from the centre of the ellipse to the circumference along the x axis and yradius is the distance from the centre to the circumference along the y axis. The centre of the ellipse is the current point.

The HLGE has one primitive that combines curved and straight lines. It is the SECTOR command and draws sections of circles shaped like pieces of pie. Its parameters are exactly the same as those used by the ARC command. The SECTOR command, however, draws lines from the ends of the arc to the centre of the circle on which the arc is drawn.

The following command string draws 2 circles, 2 ellipses, 2 arcs, and 2 circle segments. Figure 3.18 shows these elements combined with the results of the 2 preceding examples.

```
MOVE␣50␣70␣
CIRCLE␣10␣
ELIPSE␣30␣20␣
ARC␣30␣45␣135␣
MOVE␣-50␣70␣
CIRCLE␣10␣
ELIPSE␣30␣20␣
ARC␣30␣45␣135␣
MOVE␣110␣10␣
SECTOR␣60␣265␣275␣
MOVE␣-110␣10␣
SECTOR␣60␣265␣275␣
```

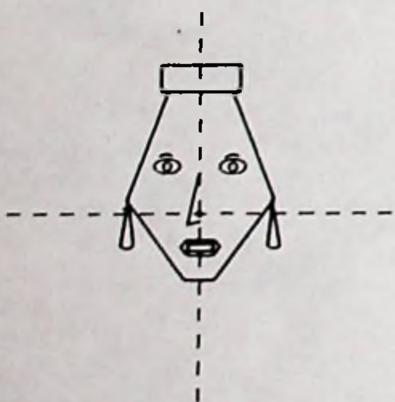


Figure 3.18: Example: Circles, Ellipses, Arcs, And Sectors

3.6.2 3D Primitives

The HLGE has the following 3D primitives:

MOVE3
MOVER3
POINT3
DRAW3
DRAWR3
POLY3
POLYR3

These commands function in the same way that their 2D counterparts do, except that they require an extra coordinate parameter–a coordinate on the z axis.

The following command string uses all 3D primitives to draw the house shown in Figure 3.19. The 3 dots on the end of the roof are there to illustrate the use of the POINT command; they have no architectural significance.

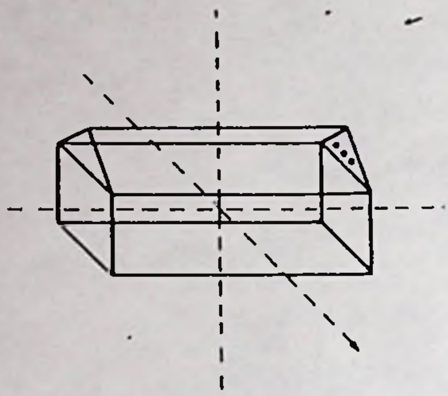


Figure 3.19: 3D Example

```
CLEARS␣0␣
MOVE3␣-100␣-30␣50␣
POLYR3␣4␣0␣0␣0␣200␣0␣0 ␣200␣60␣0␣0␣60␣0 ␣
DRAWR3␣0␣0␣-100␣
POLYR3␣4␣0␣0␣0␣200␣0␣0 ␣200␣60␣0␣0␣60␣0 ␣
MOVE3␣-100␣30␣50␣
DRAWR3␣0␣0␣-100␣
MOVE3␣100␣-30␣50␣
DRAWR3␣0␣0␣-100␣
MOVE3␣100␣30␣50␣
DRAWR3␣0␣0␣-100␣
POLY3␣4␣100␣30␣-50␣100 ␣60␣0␣-100␣60␣0␣ -100␣30␣-50␣
MOVE3␣-100␣30␣50␣
DRAW3␣-100␣60␣0␣
MOVE3␣100␣30␣50␣
DRAW3␣100␣60␣0␣
MOVE3␣100␣40␣-20␣
POINT3␣
MOVER3␣0␣0␣20␣
POINT3␣
MOVER3␣0␣0␣20␣
POINT3␣
```

3.6.3 Converting the Current Point

As we explained at the start of this section, there are two current points: the 3D current point, used to draw 3D primitives; and the 2D current point, used to draw 2D primitives and text. In many cases you will want to combine 2D primitives or text with 3D primitives in the same picture, and you will want to position the 2D images in relation to the 3D image. The CONVRT command will help you to do this. It moves the 2D current point to the position that the current 3D current point would occupy if it was mapped into the 2D coordinate space by the current 2D to 3D transforms. This saves you the trouble of calculating the position of 2D points with respects to 3D points.

So, for example, after you have drawn a 3D image, you can move the 3D current point to the place where you want explanatory text, then issue the CONVRT command, followed by a TEXT command.

3.7 Fills

There are three methods to fill areas of the screen with solid colors and patterns: primitive fills, area fills, and screen fills.

PRMFIL, the primitive fill command, allows the user to fill closed primitives (polygons, ellipses, sectors, etc.) as he draws them. The command has the following format:

PRMFIL␣flag

where flag is 0, or 1, and becomes the current primitive fill flag. If the flag is 0, closed primitives are left unfilled when they are drawn. If the flag is 1, closed primitives are filled with the current color when they are drawn. The primitive fill function works with both 2D and 3D filled primitives.

The following command string draws a box and uses the PRMFIL command to fill one side as shown in Figure 3.20:

```
CLEARS␣0
MOVE3␣-100␣-50␣50␣
POLYR3␣4␣0␣0␣0 ␣200␣0␣0 ␣200␣100␣0 ␣0␣100␣0
DRAWR3␣0␣0␣-100␣
PRMFIL␣1␣
POLYR3␣4␣0␣0␣0 ␣200␣0␣0 ␣200␣100␣0 ␣0␣100␣0
MOVE3␣␣-100␣50␣-50␣
DRAWR3␣0␣0␣100␣
MOVE3␣100␣50␣-50␣
DRAWR3␣0␣0␣100␣
MOVE3␣100␣-50␣-50␣
DRAWR3␣0␣0␣100␣
```

The primitive fill function is powerful and easy to use; however, it does have the disadvantage that it can be used only to draw filled closed primitives. When areas not in this category need to be filled, as is often the case, the user can use either of two more general area fill commands:

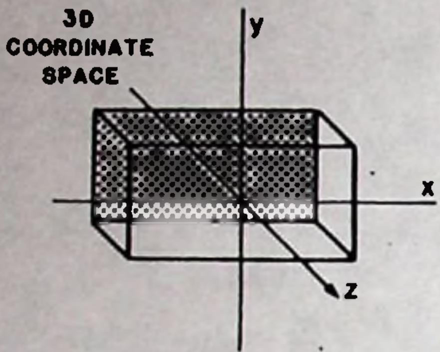


Figure 3.20: Primitive Fill Example

AREA and AREABC. These commands, which function only in the 2D coordinate space, fill outward from the current point until they reach a specified boundary. The difference between them is the way in which the boundary is defined. The AREA command has no parameters and fills with the current index outward from the current point until it encounters indices that are neither the current index nor the index of the current point (see Figure 3.21).

The AREABC command allows you to specify the boundary of the filled area. Its format is as follows:

AREABC␣bindex

where bindex is the color index the HLGE uses to contain the fill.

When the HLGE executes an AREA or AREABC command it reads pixels and compares them with the current index and the index at the current point or the boundary index to know whether it should continue filling. What the HLGE reads depends on both the mask set by the MASK command and a special mask called the fill mask. The fill mask affects read data only and is only active during area fill operations. It is set with the FILMSK command, which has the following format:

Figure 3.21: AREA Fill

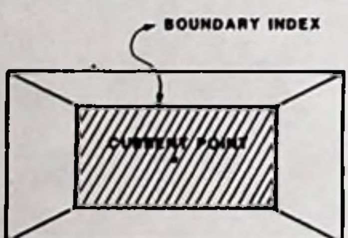


Figure 3.22: AREABC Fill

FILMSK␣mask

where mask is an 8-bit value (0-255) that is logically ANDed with plane-mask (set by the MASK command) and indices read during an area fill. The AND operation takes place before the indices are compared with the boundary index and the current index (AREABC), or the current index and the index in the current point (AREA).

The mask and the fill mask give you more flexibility in boundary specification. When the AREA command is used, they allow you to ignore certain boundary colors by masking them to look like the current index or the index at the current point. When the AREABC command is used, the masks allow you to use more than one index in the boundary by making them to look like the specified boundary index.

The most general commands, which fill the entire viewport, are the FLOOD command and the CLEARS command. The FLOOD command has the following format:

FLOOD␣index

where index is the color used to fill the viewport; the current color is not changed. The final color written to the display depends on the current mask, as selected by the MASK command.

The CLEARS command sets all pixels in the display buffer(both visible and hidden) to particular color. The format of the command is:

CLEARS␣index

where index is the color used to fill the display buffer; the current color is not changed. The current mask is ignored.

A fill does not have to be done with a solid index. The AREAPT command is provided so that the user can specify a pattern composed of the filling index and the underlying index. The command format is as follows:

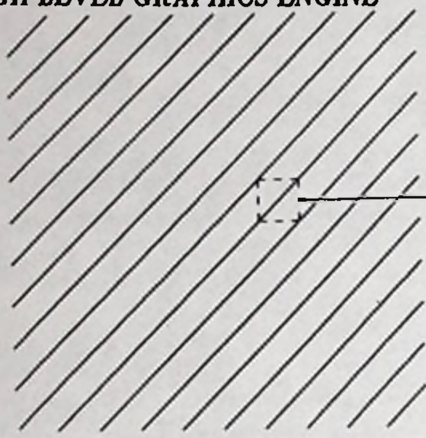


Figure 3.23: AREA Pattern Example

AREAPT␣pattern

where pattern is a 16-word array that functions as a 16-pixel by 16-pixel bit mapped pattern. Zeros in the bit map inhibit fill and allow the underlying index to show through. The HLGE applies the pattern to the filled area. The following command string defines the pattern shown in Figure 3.23.

```
AREAPT ␣1     ␣2     ␣4      ␣8
       ␣16    ␣32    ␣64     ␣128
       ␣256   ␣512   ␣1024   ␣2048
       ␣4096  ␣8192  ␣16384  ␣32768␣
```

The AREA and AREABC commands can be used to fill 3D drawings by working on them after they have been projected onto the 2D coordinate space. The CONVRT command is useful here. It converts the 3D current point into the 2D current point. Thus it can be used to position the 2D current point in the area that you wish to fill.

The following command string draws a tetrahedron illustrated in Figure 3.24, and fills one side.

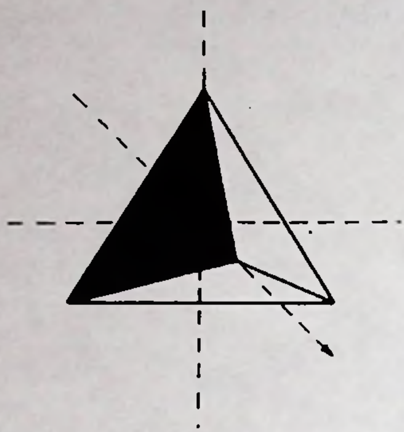


Figure 3.24: AREABC Fill Example

CLEARS␣0
COLOR␣24
MOVE3␣0␣100␣0␣
DRAW3␣100␣-60␣0␣
DRAW3␣-100␣-60␣0␣
DRAW3␣0␣100␣0␣
DRAW3␣0␣0␣170␣
DRAW3␣-100␣-60␣0␣
MOVE3␣0␣0␣170␣
DRAW3␣100␣-60␣0␣
MOVE3␣-10␣0␣0␣
CONVRT␣
COLOR␣70␣
AREABC␣24

3.8 Text

The HLGE provides the following commands to draw text:

TEXT	Draws text using standard font.
TEXTP	Draws text using user font.
TSTYLE	Selects fat or thin text for standard font.
TDEFIN	Defines raster text characters for user font.
GTDEF	Defines vector text characters for user font.
TJUST	Sets text position relative to current point.
TSIZE	Sets text size.
TASPCT	Sets text aspect ratio.
TANGLE	Sets drawing angle.
TCHROT	Sets character rotation.
RDEFIN	defines raster text charaters for user fonts 1 to 15
RFONT	selects active user raster font

The HLGE has 2 character fonts, the standard font and the user font, and each of these fonts uses two different kinds of text. The standard font uses thin text or fat text, and the user font uses raster text or vector text.

To display text the user issues a TEXT or a TEXTP command followed by the text to be displayed. The TEXT command has the following format:

TEXT␣string

where string is a string of characters delimited by either single or double quotes. The HLGE uses the standard character font (Figure 3.25) to draw the characters in the string at a position relative to the 2D current point as specified by the most recent TJUST command. The TEXTP command is identical except that it uses the user font defined by the RDEFIN, TDEFIN and GTDEF commands.

The TJUST command allows you to position text to the left of the current point, to the right of the current point, or centred on the current point in the horizontal direction; and it allows you to position the text above, below, or centred on the current point in the vertical direction (see Figure 3.26). The command format is as follows:

TJUST␣horiz␣vert

where horiz and vert specify the position of text with respect to the current point as follows:

horiz

1	start of text line is at the current point
2	centre of text line is a current point
3	end of text line is a current point

vert

1	bottom of text is at current point
2	centre (vertically) of text is at current point
3	top of text is a current point

If you use the standard font, you can use either fat text or thin text. Use the TSTYLE command to select one or the other. Slim text characters are always one pixel wide irrespective of their size. The lines that make up fat characters, on the other hand, become wider as the characters become larger. Fat style characters are the same as slim characters when the default text size is in effect. The 'fat' effect is noticeable only as text sizes become larger. Each character exists in both forms.

If you use the user font you can use either vector text or raster text, provided that you have created the characters that you want to use. Use the GTDEF command to create vector text characters and use the

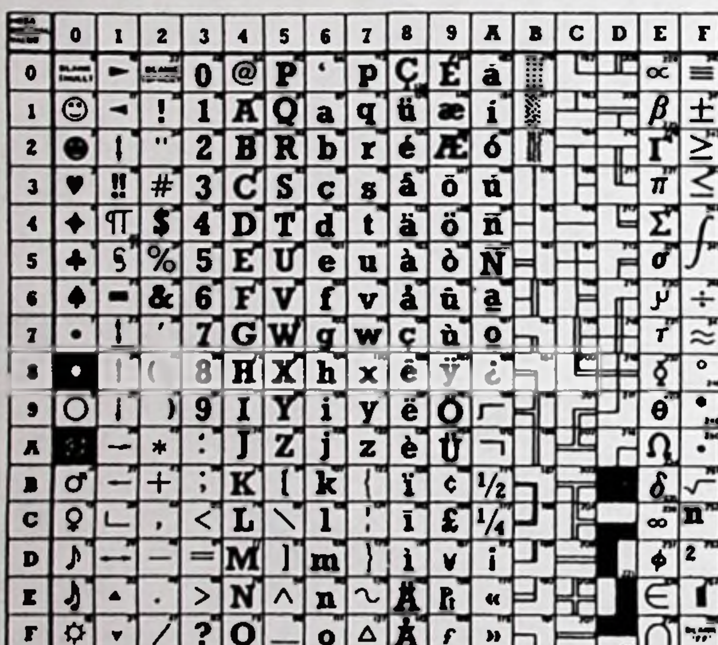


Figure 3.25: The Standard Font

TDEFIN or RDEFIN command to create raster text characters. Note that whereas fat and thin characters with the same code coexist, raster text characters and vector text characters with the same code do not. That is to say that you can not create both a vector text character and a raster text character for the same font position. If you attempt to display a character that you have not defined, the HLGE will use the corresponding standard font thin character.

Subsection 3.8.2 explains how to define characters for the user font.

3.8.1 Character Attributes

HLGE text may have the following attributes:

ATTRIBUTE	COMMAND
color	COLOR
angle	TANGLE
rotation	TCHROT
size	TSIZE
aspect ratio	TASPCT

The color attribute applies to all text types in both fonts and is simply the current color set with the COLOR command; however, the other attributes do not apply to all text types. Table 3.6 indicates what the restrictions are.

The TSIZE and TASPCT commands allow you to set the size and aspect ratio of the text characters. The format of the TSIZE command is as follows:

TSIZE␣size

where size specifies the number of coordinate space points between the start of one character and the start of the next in the horizontal direction.

	STANDARD FONT		USER FONT	
	THIN TEXT	FAT TEXT	VECTOR TEXT	RASTER TEXT
TANGLE	*	*	*	
TCHROT	*		*	
TSIZE	*	*	*	
TASPCT	*		*	
* = applicable				

Table 3.6: Character Attribute Use Restrictions

The height of the characters is determined by the aspect ratio command, which has the following format:

TASPCT␣ratio

where ratio is character cell height divided by character cell width. Thus, if you set width to 10 and aspect ratio to 1, you will draw character cells 10 points by 10 points in size. The as-viewed aspect ratio also depends on the current window to viewport map and how the characters are defined in the character font.

The TANGLE and TCHROT commands allow you to draw slanted text in various ways. TANGLE allows you to draw text at an angle and TCHROT allows you to rotate characters on their lower left corner. Thus you can have both types of slanted text shown in Figure 3.27 and variations in between. In both cases the command argument is an angle from the x axis in counterclockwise direction.

The following command string draws large (50 pixels wide) thin characters rotated, on an angle, and centred on the current point. It uses the standard character set and an aspect ratio of 1.5. The result is illustrated in Figure 3.28.

```
CLEARS␣0␣
TJUST␣2␣2␣
TSIZE␣50␣
TSTYLE␣1␣
TANGLE␣45␣
TCHROT␣45␣
TASPCT␣1␣
MOVE␣0␣0␣
TEXT␣'PG-640A'␣
```

3.8.2 Defining Characters For The User Font

At reset the user font is empty, but characters can be defined in it by the RDEFIN command, TDEFIN command or the GTDEF command.

Characters created with the GTDEF command and the characters in the standard font are formed from small character command lists similar to the command lists used to save graphics commands, and as such they can be rotated, scaled, and translated.

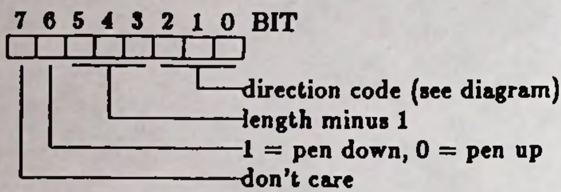
The format for the GTDEF command is as follows:

GTDEF␣ch␣n␣width␣array

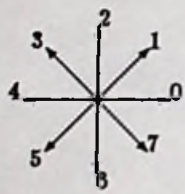
where ch is a number from 0 to 255 identifying a character position in the standard ASCII character map, n is the number of bytes in the array parameter, width is the width of the character in character vectors, and array is an array of vector parameters. The height of the character cell is fixed at 12 vectors. A vector parameter gives a direction, a distance, and a draw/move flag. In ASCII Command Mode, ch. n, and width are Chars and each vector parameter in array is composed of 3 Chars. In Hex Command Mode, ch. n, and width are byte values and each vector parameter in array is composed of a single value. The format of the vector parameters is shown in the following 3 diagrams:

```
Char Char Char
 |    |    |
 |    |    └──direction code (see diagram)
 |    └───────length
 └────────────1 = pen down, 0 = pen up
```

ASCII Command Mode Vector Parameter Format



Hex Command Mode Vector Parameter Format



Vector Parameter Direction Codes

For example, the following code defines an 'A':

ASCII

```
GTDEF 65 7 8 4
           1 7 2
           1 2 1
           1 3 0
           1 2 7
           1 7 6
           0 4 2
           1 7 4
```

Hex

89 41 07 08 72 49 50 4F 76 1A 74

The PG-640 allows the user to define up to 16 raster fonts in memory, labeled from 0 to 15. The raster characters are bit maps and can not be transformed, so you must define them as you want to see them.

User Raster Font 0

User raster font 0 characters are defined using the TDEFIN command. For this font, each character must be defined separately. The maximum cell size of these characters is 255×255 pixels. This font is the PGC compatible user definable raster font.

The TDEFIN characters are bit maps and cannot be transformed, so you must define them as you want to see them. The command format is as follows:

TDEFIN␣n␣x␣y␣array

where n is a number from 0 to 255 identifying a character position in the font, x is the character cell width in screen coordinates, y is the character cell height in screen coordinates, and array is an array of bytes that forms the bit map of the character being defined.

User Raster Fonts 0 to 15

User raster fonts 1 to 15 are special fonts optimized for drawing speed. Each font must be defined "a complete font at a time ", using the RDEFIN command. All characters in a given font in this range must have the same cell dimension; the maximum size is 16×16 pixels.

User Raster Font Selection

Only one of the 16 user raster fonts can be active at any one time. The raster font used to draw characters (0 to 15), with the TEXTP and TEXTPC commands, is selected using the RFONT command. This command also specifies the aspect ratio of the characters drawn, with a choice of any combination of single/double height and single/double width.

The following command string creates the character shown in Figure 3.29 and assigns it to character 'A' (code 65).

```
TDEFIN 65 8 41 1 1 1 0 0 0 0
               1 0 0 1 0 0 0 0
               1 0 0 1 0 0 0 0
               1 1 1 1 1  1 1 0
```

PG-640A　PG-*640A　PG-640A
.PG-640A　PG-.640A　PG-640A.　* current point
'PG-640A　PG-'640A　PG-640A'

Figure 3.26: Justification Options

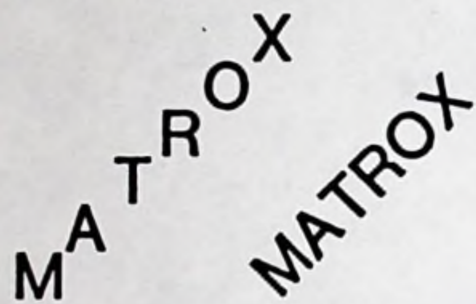


Figure 3.27: Slanted Text

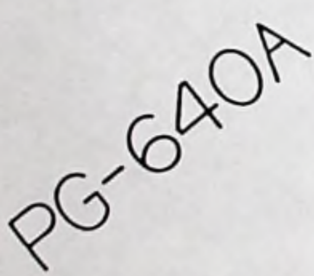


Figure 3.28: Text Example

Figure 3.29: TDEFIN Example

3.9 Command Lists

A command list is a list of commands that draws an object or performs a sequence of other functions. Most graphics software creates command lists which are stored and used repeatedly as required.

If you have complex objects and the command lists are stored in system RAM, loading them into the command FIFO can take a relatively long time, time that the system CPU could better use for other purposes. Fortunately the HLGE allows you to store command lists in the PG-640A itself. The system CPU then needs only to pass one command to the HLGE to call the command list and draw the graphic object that is contains.

The user defines a command list by sending the HLGE a CLBEG command followed by the command list terminated with a CLEND command. The format of the CLBEG command is as follows:

CLBEG␣clist

where clist is a number from 0-255 identifying the command list. The CLEND command has no argument.

Once it is defined, the user can run a command list by issuing either a CLRUN command or a CLOOP command. The CLRUN runs a specified command list one time; the CLOOP command runs a specified command list a specified number of times. The format of the CLRUN command is as follows:

CLRUN␣clist

where clist is a number from 0-255 identifying the command list that is to be run.

The format of the CLOOP command is as follows:

CLOOP␣clist␣count

where clist is a number from 0-255 identifying the command list to be run, and count is a number from 0-65535 specifying the number of times the command list is to be run.

The following command string defines a command list, then runs it by looping it twice. Because the last two commands in the command list change the modeling transform, the loop gives two different views of the same object, as shown in Figure 3.30. Note that you don't see anything on the screen until you issue CLRUN.

```
CLEARS␣0␣
CLBEG␣1␣
MOVE3-100␣50␣0␣
POLYR3␣4␣0␣0␣0 ␣200␣0␣0␣ 200␣50␣0␣ 0␣50␣0␣
DRAWR3␣0␣0␣100␣
POLYR3␣4␣0␣0␣0 ␣200␣0␣0 ␣200␣50␣0 ␣0␣50␣0␣
MOVE3-100␣100␣0
DRAWR3␣0␣0␣100␣
MOVE3␣100␣100␣0␣
DRAWR3␣0␣0␣100␣
MOVE3␣100␣50␣0
DRAWR3␣0␣0␣100␣
MDROTY␣45␣
MDTRAN␣0␣-125␣0␣
CLEND␣
MDIDEN␣
CLOOP␣1␣2␣
```

Once a command list has been defined, it can be read and modified by the user. The CLRD command allows the user to read a specified command list. The CLMOD command allows the user to modify a command list.

The CLRD function has the following format:

CLRD␣clist

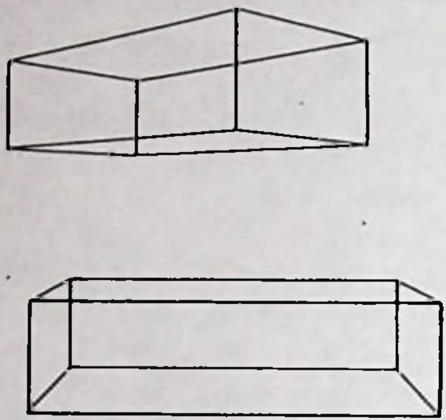


Figure 3.30: Command List Example

Where clist is the name of the command list to be read. When it receives this command, the HLGE sends the command list, in hexadecimal, to the read back buffer. The data consists of one word giving the number of the bytes in the command list, followed by those bytes.

The CLMOD command specifies a section of a command list and replaces that section with an array of bytes provided in the command argument. The command has the following format:

CLMOD␣list␣offset␣nbytes␣bytes␣

Where list is the number of the command list to be modified, offset is the offset in bytes from the start of the command list to the start of the section that is to be replaced, nbytes is the number of bytes to be replaced, and bytes is an array of replacement bytes. The number of bytes in the replacement array (bytes) must be exactly the same as the number of bytes in nbytes. Because of this restriction, if you need to remove a command without replacing it, you will have to put a NOOP command in its place.

When using the CLMOD command, keep in mind that real coordinates

(32 bits) are not stored in memory in the same order as they are received from the Host. When you specify a real number it is in the form of:

[low integer][high integer][low fraction][high fraction]

This number is received by the Host and stored in memory in the following form:

[low fraction][high fraction][low integer][high integer]

When a coordinate is stored in a command list, the firmware exchanges the bytes so that the fractional part is stored first. When a CLRD command is processed, a reverse exchange is made so that coordinates appear just as they were sent.

Using the CLMOD command on a section of a real coordinate, stored in a command list, performs no exchange. Therefore:

- Modifying the first byte of a coordinate modifies its [low fraction].
- Modifying the second byte of a coordinate modifies its [high fraction].
- Modifying the third byte of a coordinate modifies its [low integer].
- Modifying the fourth byte of a coordinate modifies its [high integer].

For example:

```
CLBEG␣1␣
MOVE␣10␣20␣
CLEND␣
CLRD␣1␣
```

The read back buffer contains:

```
09 00  10  0A 00  00 00  14 00  00 00
                                      y fraction
                                      y integer
                                      x fraction
                                      x integer
                                      opcode
                                      length of command
                                      list
```

CLMOD␣1␣3␣1␣30␣
CLRD␣1␣

Note that the previous CLMOD command modified the third byte in clist, which is the low byte of the integer part of x.

The read back buffer contains:

```
09 00  10  1E 00  00 00  14 00  00 00
                                      y fraction
                                      y integer
                                      x fraction
                                      x integer
                                      opcode
                                      length of commmand
                                      list
```

The modified byte seems to be the second byte in the command list, but in fact it is the third byte because the CLRD command exchanges real coordinates.

3.10 Direct Screen Operations

The HLGE has a number of commands which allow the user to bypass the normal coordinate space/transform sequence and work directly in the display buffer.

The 'S' series commands, listed below, are graphics primitives that draw directly on the screen. They are the same as the 2D primitives except that they use screen coordinates instead of 2D coordinates. They are faster than the 2D primitives but have the disadvantage that they limit images to the resolution of the screen–you can not focus on different parts of the image with the window function and you can not zoom in on details. Pictures created with the 'S' series commands are clipped to the current viewport, and the end points of lines are not drawn. For **the 'S' series primitives to function properly the window and viewport must have exactly the same coordinates which must be equal to the maximum screen resolution.** That is to say, the viewport must be equal to the full screen, and the window must have corners at 0,0 and 639,479. negative values are not allowed.

SARC
SCIRC
SDRAW
SDRAWR
SELIPS
SMOVE
SMOVER
SPOLY
SPOLYR
SRECT
SRECTR
SSECT

For those users who require something even faster than the 'S' series commands we have provided the PDRAW command. It has a more primitive coordinate specification format that allows it to execute moves and draws faster than the 'S' series commands. The command format is as follows:

$$\mathrm{PDRAW}\sqcup x_1\sqcup y_1\sqcup x_2\sqcup y_2\sqcup \ldots x_n\sqcup y_n\sqcup$$

where x and y are Int screen coordinates with the most significant bit of the y coordinate used to specify a move or a draw and the most significant bit of the x coordinate used to specify continue or stop.

The IMAGER and IMAGEW commands allow you to transfer lines or parts of lines of pixel values between the system memory and the display buffer, the RASTRD and RASTWR commands allow you to move rectangular sections of the display buffer to and from the system memory, and the RASTEROP command allows you to move rectangular sections of the display memory from one part of the display memory to another.

The IMAGER command has the following format:

IMAGER␣line␣x1␣x2

where line is a y coordinate indicating a horizontal line of pixels in the screen coordinate space, x1 is an x coordinate indicating a starting point on the line, and x2 is an x coordinate indicating an ending point (inclusive) on the line. The HLGE copies the pixel values in the specified section of the specified line to the Read Back FIFO. The data format depends on whether the HLGE is in ASCII Mode or Hex Mode.

In ASCII Mode a line is passed in the following format:

IR,x,x,x...(CR)

where 'IR' is a header identifying the string as the result of an IMAGER command, where the x's represent ASCII decimal color indices separated by commas, and where the carriage return character ends the string.

In Hex Mode the data is run-length encoded. When two or more contiguous pixels have the same index, two bytes are sent: the first one with the number of bytes minus one and the second byte with the index.

When 2 or more contiguous pixels have different indices, the number of pixels minus one is sent in a byte with the most significant bit set, then binary values of the indices for each pixel in the series are sent in separate bytes (1 byte per pixel). Since the most significant bit in the initial byte is used to differentiate the 2 types of code, only 7 bits remain to give the number of pixels in the series, limiting the number of pixels in each series to 128.

For example, a series of pixels with the values 1 1 1 1 2 3 4 5 5 5 5 6 7 would be encoded as: 03 01 82 02 03 04 03 05 81 06 07.

The IMAGEW command has the following format:

IMAGEW␣line␣x1␣x2␣data␣

where line, x1, and x2 specify a line segment in the same way as in the IMAGER command and data is data that is to be written into that segment. The data format is the same as for the IMAGER command except that the first two characters in the ASCII string are 'IW' instead of 'IR'.

Although the RASTRD and RASTWR commands also transfer data directly between the display memory and the system memory they differ from IMAGER and IMAGEW in that they do a raster scan of a specified rectangular area, incorporate certain logical functions, do not use the FIFO for data transfer, and do not provide run-length encoding. The data transfer is made over one of the PC's DMA channels–usually channel 1, although channel 2 or channel 3 can be used if necessary (see Appendix A).

The format of the RASTWR command is a follows:

RASTWR␣oper␣dir␣x0␣x1 ␣y0␣y1

where oper specifies a logical operation (see Table 3.7), dir specifies major and minor scan directions (see Table 3.8), x0,y0 specify, in screen coordinates, one corner of the rectangle to be scanned, and x1,y1 specify the opposite corner on the diagonal. The HLGE scans by reading and writing a line of pixels along the major scan direction, then moving one scan line in the minor direction and repeating the process. As it passes over each pixel in the scan, it performs the specified logical operation between the data coming from the DMA interface and the current data in the pixel location and writes the result into the location. Figure 3.31 shows a typical scan, and Table 3.8 indicates the scan directions that can be used with this command (note that the use of some scan directions depends on the logical operation selected). The raster referred to here is the same as the video raster used to refresh the screen but has a different scan.

The RASTRD command is the same as the RASTWR command except that it transfers data from the scanned area to the DMA interface, and does not do any logical operations on the data. In both cases each index is passed in a single byte, and until the transfer is complete, no other commands are interpreted by the PG-640A. The number of bytes transferred is $(x_1 - x_0 + 1) \times (y_1 - y_0 + 1)$.

The following command string XORs data from the DMA interface with data in the specified rectangle and writes the results into the rectangle. Figure 3.31 shows the scan directions. Before sending such a command string to the command FIFO, the user must program the DMA channel for a memory to I/O transfer of 60000 bytes, and the area of system memory specified for the transfer must contain the data that he wants to write into the rectangle.

RASTWR␣3␣1␣100␣400␣100␣300␣

The HLGE has a third raster command which uses the same general format to copy rectangular areas from one part of the screen to another. It is the RASTOP command and has the following format:

RASTOP␣oper␣srcdir␣destdir␣x_0␣x_1␣y_0␣ y_1␣x'_0␣y'_0

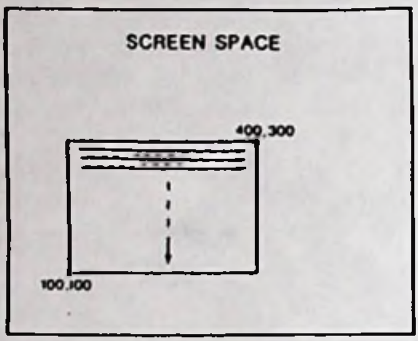


Figure 3.31: Raster Scan

RASTER FUNCTIONS	
Function Code	Operation
0	Copy
1	OR
2	AND
3	XOR

Table 3.7: Logic Operations

Scanning Direction			Used By		
Code	Major	Minor	RASTWR	RASTRD	RASTOP
0	⇒	↑	√	√	√
1	⇒	↓	√*	√	√
2	⇐	↑	√*	√	√
3	⇐	↓	√*	√	√
4	⇑	→			√
5	⇓	→			√
6	⇑	←			√
7	⇓	←			√
* only when logic operation 0 is selected					

Table 3.8: Scan Directions

where oper specifies a logical operation, srcdir is the scan direction in the source rectangle, destdir is the scan direction in the destination rectangle, x_0, y_0, x_1, and y_1 specify the source rectangle, and x'_0, y'_0 specify the lower left corner of the destination rectangle.

The following command string copies the contents of rectangle A in Figure 3.32 to rectangle B. Note that by using different source and destination scan directions we are able to draw a mirror image.

RASTOP␣0␣1␣3␣100␣300 ␣100␣:300␣400␣100␣

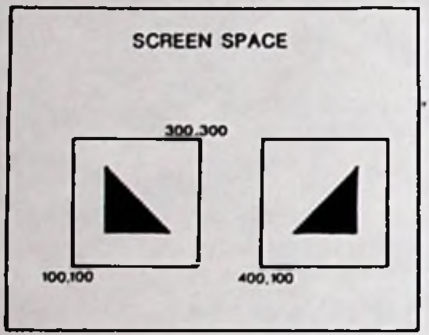


Figure 3.32: RASTOP Example

3.11 The Text Window

The HLGE has a special feature which allows you to have a window into the CGA Emulator screen while looking at the HLGE screen. The window can be moved to various positions on both screens and does not affect underlying graphics on the HLGE screen. It allows the user to use the CGA Emulator and the HLGE at the same time without adding a second monitor.

The text window has some restrictions. (1)The CGA Emulator must be in an alphanumeric video mode. (2)The text window is not an exact copy of the CGA emulator screen; color and intensity attributes are ignored. One foreground color is used throughout the text window; the background is transparent. The cursor is still visible in the text window, as is the blink attribute.

The text window is controlled by three HLGE commands: TWPOS, which positions the window, TWVIS, which controls visibility, and TWCOL, which sets a foreground color.

The format of the TWPOS command is as follows:

$$\text{TWPOS}_\sqcup x_{0\sqcup} x_{1\sqcup} y_{0\sqcup} y_{1\sqcup} e_{0\sqcup} e_1$$

where x_0, y_0 and x_1, y_1 specify, in screen coordinates, opposite corners of a rectangular area of the HLGE screen; and e_0 and e_1 specify the upper left corner of a corresponding rectangle on the CGA screen. Values of x_0 and $x_1 + 1$ must be on 16-pixel boundaries(ie: divisible by 16).

The x_0, x_1, y_0, and y_1 parameters are specified in pixels; the point of origin of the HLGE screen is the lower left corner. The e_0 and e_1 parameters are specified in character cells based on the CGA 80×25 video mode; the point of origin of the CGA screen is the upper left corner. The relationship of e_0 and e_1 to the x and y coordinates of the upper left corner of the CGA rectangle depends on the current CGA video mode as follows:

VIDEO MODE	RELATION OF X AND Y TO e_0 AND e_1
80 × 25	$x = e_0$, $y = e_1$
40 × 25	$x = e_0$, $y = 2e_1$

The reason this relationship is true is because a line of text in the 40 × 25 CGA mode occupies half as many bytes of memory as a line of text in the 80 × 25 CGA mode. If you specify $e_1 = 4$ when the CGA mode is 40 × 25 then the CGA rectangle would actually start at line y = 8.

The TWVIS command sets a flag which determines whether the HLGE displays the window defined by the TWPOS command. Specify a foreground color using the TWCOL command; this command has the format: TWCOL␣r␣g␣b

The following command string defines the window shown in Figure 3.33.

TWPOS␣512␣527␣256␣271␣20␣10␣
TWVIS␣1␣

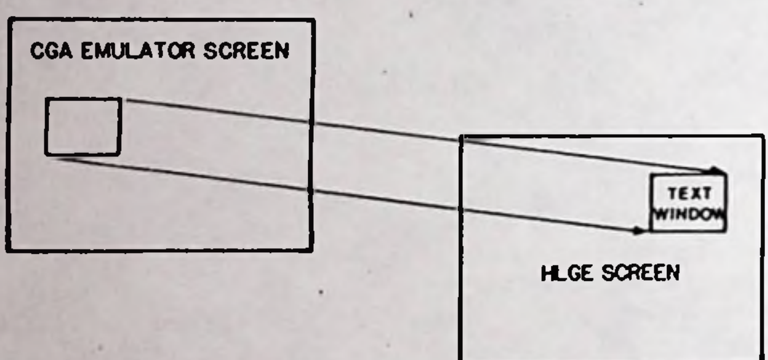


Figure 3.33: The Emulator Window

You also have the option of full-screen CGA emulation using the DISPLA command: DISPLA␣flag

Depending on the value of flag either the HLGE screen or the CGA

screen is displayed on your monitor. Full-screen emulation is the only way to see the CGA screen exactly as is.

3.12 Read Back Commands

The PG-640A supports a number of read back commands that will allow the user to determine the exact values of the High Level Graphics Engine's parameters. The read back commands are: Command List Read (CLRD), Flag Read (FLAGRD), Image Read (IMAGER), LUT Read (LUTRD), and Matrix Read (MATXRD). These commands are detailed in the command summary chapter.

When a read back command is executed, the HLGE puts the requested information in the Read Back Buffer. When in ASCII mode, the data is returned as ASCII decimal numbers terminated by a carriage return. Some commands return multiple values; the individual command descriptions give the data formats in both ASCII and Hex communication modes.

Note that if a read back is requested and the read back buffer is full, the HLGE will halt and wait for you to empty the buffer.

3.13 Error Handling

If the user has set the Error Enable Flag in the communications area, the PG-640A will return error messages or codes in the current communication mode. In ASCII mode the PG-640A will return ASCII strings containing an error message, in Hex mode a single byte is returned containing an error code. The return messages and codes are summarized in Table 3.9.

The HLGE writes error messages into the Error FIFO. If the FIFO becomes full before the message is complete the HLGE waits until there is room in the FIFO. While it is waiting, the HLGE will not accept any commands.

Hex Code	ASCII String	Means
1	Range	parameter out of range
2*	Integer	wrong data type–need integer
3	Memory	ran out of memory
4	Overflow	arithmetic overflow
5*	Digit	wrong data type– need digit
6	Opcode	opcode not recognized
7	Running	recursion in command list
8	Stack	commands lists nested more than 16 deep
9	Too long	string or command list too long
A	Area	area fill error
B*	Missing	missing parameter

* These errors do not occur in Hex Mode

Table 3.9: Summary of Error Codes and Messages

3.14 Graphics Input Support

Many applications will require the use of a graphics input device such as a mouse, joystick, or trackball. The graphics input device will be interfaced to the user's software, which will use it to move a cursor, to frame areas of text, to draw lines, or to implement some other application dependent function. For example, in a computer aided design application the operator might use a mouse to move a cursor to specify points that need to be interconnected on a design.

The HLGE provides the following 3 commands to help the applications software implement graphics input functions:

XHAIR
XMOVE
RBAND

XHAIR displays a cross hair cursor, XMOVE moves the cursor, and RBAND performs two kinds of rubberbanding. All 3 commands operate in screen space only.

The format of XHAIR is as follows:

XHAIR␣flag␣xsize␣ysize

where flag enables the cross hair display at the current cross hair position, and xsize and ysize determine its size. The flag parameter can be Chars 0 through 4.

- *0:* Cross hair display disable.
- *1:* Cross hair display enabled, clipped to screen space.
- *2:* Cross hair display with dimensions of 100 by 100 is enabled, clipped to screen space.
- *3:* Cross hair display enabled, clipped to current view port.

4: Cross hair display with dimensions of 100 by 100 is enabled, clipped to current view port.

The xsize and ysize parameters must be given in screen coordinates and determine the x and y dimensions of the cross hair respectively. When flag is 0, 2, or 4, x size and y size must not be sent.

The HLGE draws the cross hair in complement drawing mode, so its color is affected only by what is already on the screen and not by the current index.

The XMOVE command has the following format:

XMOVE␣x␣y

where x and y are the screen coordinates of a new cross hair position. The XHAIR command has no effect on this command. That is to say XMOVE, moves the cross hair whether or not it is currently displayed.

The RBAND command has the following format:

RBAND␣flag

where flag is a Char from 0 through 2 that affects rubberbanding as follows:

0: Disables rubberbanding.

1: Enables vector rubberbanding. The current cross hair position, at the time when rubberbanding is enabled, becomes the anchor point. The HLGE draws a line between the anchor point and each new cross hair position. Each time that it draws a line from the anchor point to a new cross hair position it erases the line that it drew from the anchor point to the previous cross hair position. When rubberbanding is disabled, the HLGE erases the most recent rubber band vector and the cross hair is left at the coordinate pair most recently entered.

2: Enable rectangle rubberbanding. This rubberbanding mode is the same as vector rubberbanding except that instead of drawing a line between the anchor point and the cross hair position, the HLGE draws a rectangle with one corner at the anchor point and the diagonally opposite corner at the current cross hair position. Note, however, that since the rectangle is drawn in complement mode, you will lose the part of the rectangle that overlaps the cross hair, if the cross hair display is enabled. For this reason it is best to disable cross hair display when using rectangle rubberbanding. This mode is useful for framing parts of the display that the application program treats in some special way.

The following sequence of commands illustrates the use of the graphics input commands. The first 2 commands enable the cross hair display and move the cross hair to screen coordinates 100, 200. The next two commands enable vector rubberbanding, establishing the anchor point, and move the cross hair to 500, 400. The rubberbanding function draws a line from the anchor point to the new cross hair position. The last command moves the cross hair to 500,50, and the rubberbanding function erases the first line and draws a new line to the new cross hair position. Figure 3.34 shows the process.

```
XHAIR␣1␣100␣100␣
XMOVE␣200␣250␣
RBAND␣1␣
XMOVE␣500␣400␣
XMOVE␣500␣50␣
```

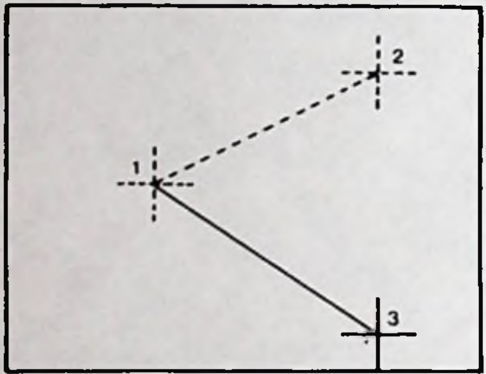


Figure 3.34: Graphics Input Example

Chapter 4

Command Descriptions

The following pages contain descriptions of the commands used by the high level graphics engine. These commands are arranged in alphabetical order by command name and use the conventions set out in Chapter 3 to distinguish hexadecimal numbers, command names, and parameters from regular text. The parameter types use the definitions that are laid out in Section 3.2.

COMMAND :

LONG FORM : ARC radius angle1 angle2

SHORT FORM : AR radius angle1 angle2

HEX FORM : 3C radius angle1 angle2

PARAMETER TYPE : radius = Real
angle1 = Int
angle2 = Int

DESCRIPTION : ARC draws a circular arc using the currently selected color. The center is at the 2D current point. The start and finish angles are specified in the command. The angle can be any Int value (angles greater than 360° and less than -360° are handled as modulo 360). Negative radii will result in 180° being added to both angles. This command does not affect the 2D current point.

EXAMPLE :

CODE :

ASCII : AR 100.00 0 180

HEX : 3C 64 00 00 00 00 00 B4 00

RESULT : An arc with radius 100 from 0° to 180° (a semi-circle) is drawn about the 2D current point.

ERRORS : Overflow

RELATED MATERIALS : CIRCLE, COLOR, LINFUN, LINPAT, Section 3.6

AREA
(Area Fill)

COMMAND :

LONG FORM : **AREA**

SHORT FORM : **A**

HEX FORM : **C0**

PARAMETER TYPE : None

DESCRIPTION : AREA sets all the pixels in a closed area to the current color. The closed area starts from the 2D current point and continues outward in all directions until a boundary with a color different from that of the starting pixel's original color is reached. The data tested is ANDed with the fill mask (FILMSK) and the bit plane mask (MASK) before comparing colors. The start pixel's original color should not be the current color.

EXAMPLE :

CODE :

ASCII : **A**

HEX : **C0**

RESULT : The bounded area that contains the 2D current point is changed to the current color.

ERRORS : None

RELATED MATERIALS : AREAPT, FILMSK, MASK, Section 3.7

AREABC
(Area Fill to Boundary Color)

COMMAND :

LONG FORM : AREABC index

SHORT FORM : AB index

HEX FORM : C1 index

PARAMETER TYPE : index = Char

DESCRIPTION : AREABC fills a closed area bounded by color index with the current color. The closed area starts from the 2D current point and continues outward in all directions until reaching a boundary of pixels of color index. All pixel data read is ANDed with the fill mask (FILMSK) and the bit plane mask (MASK) before testing for the boundary.

EXAMPLE :

CODE :

ASCII : AB 100

HEX : C1 64

RESULT : The color of the area containing the 2D current point and bounded by color index is changed to the current color.

ERRORS : Boundary = current color

RELATED MATERIALS : AREAPT, COLOR, FILMSK, MASK, Section 3.7

AREAPT
(Area Pattern)

COMMAND :

LONG FORM : AREAPT pattern

SHORT FORM : AP pattern

HEX FORM : E7 pattern

PARAMETER TYPE : pattern = 16 Unsigned Ints

DESCRIPTION : AREAPT sets the area pattern mask. The pattern mask defines a 16 by 16 pixel array which is repeated horizontally and vertically when drawing filled figures. Each value in pattern is 16 bits long and sets one row of the pattern mask. Since there are 16 words in pattern, the user is able to define the value of each pixel in the pattern mask. Pixels that are where the mask is set to 1 are changed to the current color; where the mask is 0, the pattern is transparent. Setting all the bits in the mask (sending 16 words of 65535) causes areas to be filled solidly; this is the default after a reset. The area pattern is used by the following commands when drawing a filled primitive:

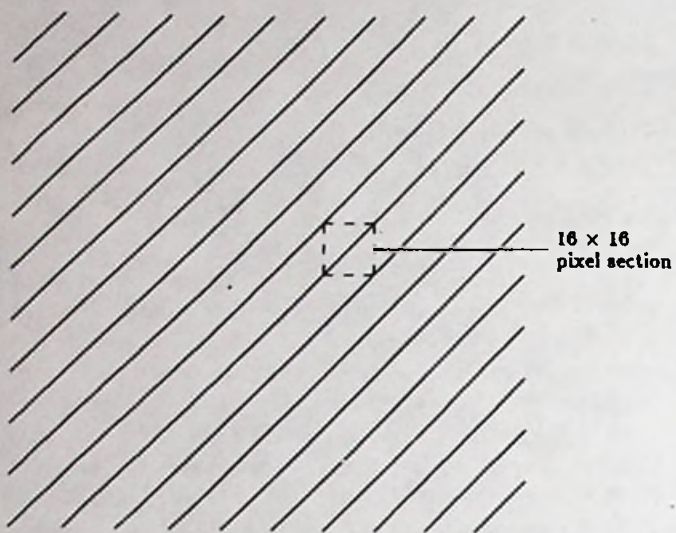
CIRCLE, ELIPSE, POLY, POLYR, POLY3, POLYR3, RECT, RECTR, SECTOR.

EXAMPLE :

CODE :

ASCII : AP	1	2	4	8
	16	32	64	128
	256	512	1024	2048
	4096	8192	16384	32768
HEX : E7	00 01	00 02	00 04	00 08
	00 10	00 20	00 40	00 80
	01 00	02 00	04 00	08 00
	10 00	20 00	40 00	80 00

RESULT :



ERRORS : None

RELATED MATERIALS : AREA, AREABC, BCOLOR, COLMOD, Section 3.7

BCOLOR
(Set Background Color)

COMMAND :

LONG FORM : BCOLOR index

SHORT FORM : BC index

HEX FORM : CB index

PARAMETER TYPE : index = Char [0..255]

DESCRIPTION : This command sets the index of the background index to be used when COLMOD is set to 0.

EXAMPLE :

CODE :

ASCII : BCOLOR 24

HEX : CB 18

RESULT : The background index is changed to 24.

ERRORS : None

RELATED MATERIALS : COLMOD, AREAPT, LINPAT, TEXT, Section 3.5, Section 3.8

COMMAND :

LONG FORM : BLINK index red green blue ontime offtime

SHORT FORM : BL index red green blue ontime offtime

HEX FORM : C8 index red green blue ontime offtime

PARAMETER TYPE : index = Char
red = Char [0..15]
green = Char [0..15]
blue = Char [0..15]
ontime = Char
offtime = Char

DESCRIPTION : BLINK causes all the pixels having the color in the currently selected lookup table specified by index to blink on and off. The periods, in $\frac{1}{60}$ seconds, are specified by ontime and offtime. During the on time, the pixel will have the original color – during the off time the color will be the one defined by red, green and blue. This command only specifies the high nibble of red, blue, and green values; BLINKX is the preferred form of the command. The low nibbles are set to zero.

Up to four indices can be set to blink at any one time. A blink is cancelled by issuing a second BLINK command for an index with the other parameters equal to zero.

Warning: Do not perform LUT changes on indices that are currently blinking.

EXAMPLE :

CODE :

ASCII : BL 15 0 0 0 30 30

HEX : C8 0F 00 00 00 1E 1E

RESULT : White (index 15) blinks to black once a second.

ERRORS : Too many blinks specified, Color already blinking

RELATED MATERIALS : LUT, LUTX, LUTINT, Subsection 3.5.3

BLINKX
(Blink – 8 Bit)

COMMAND :

LONG FORM : BLINKX index red green blue ontime offtime

SHORT FORM : BLX index red green blue ontime offtime

HEX FORM : E6 index red green blue ontime offtime

PARAMETER TYPE : index = Char
red = Char
green = Char
blue = Char
ontime = Char
offtime = Char

DESCRIPTION : BLINKX causes all the pixels having the color in the currently selected lookup table specified by index to blink on and off. The periods, in $\frac{1}{60}$ seconds, are specified by ontime and offtime. During the on time, the pixel will have the original color – during the off time the color will be the one defined by red, green and blue. Up to four indices can be set to blink at any one time. A blink is cancelled by issuing a second BLINKX command for an index with the other parameters equal to zero.

Warning: Do not perform LUT changes on indices that are currently blinking.

EXAMPLE :

CODE :

ASCII : BLX 15 0 0 0 30 30

HEX : E6 0F 00 00 00 1E 1E

RESULT : White (index 15) blinks to black once a second.

ERRORS : Too many blinks specified, Color already blinking

RELATED MATERIALS : LUT, LUTINT, LUTX, VDISP, Subsection 3.5.3

COMMAND :

LONG FORM : CA␣

SHORT FORM : CA␣

HEX FORM : 43 41 20

PARAMETER TYPE : None

DESCRIPTION : CA sets the communication mode to ASCII. This command may be given when in either ASCII mode or Hex mode. Note that the Hex and ASCII forms of this command are identical.

EXAMPLE :

CODE :

ASCII : CA␣

HEX : 43 41 20 or D2

Note: You can user either of the 2 hex formats given above to issue this command; however, the PG-640Aalways uses D2 in command lists that it creates.

RESULT : The communications mode is set to ASCII.

ERRORS : None

RELATED MATERIALS : CX, Section 3.2

COMMAND :

LONG FORM : CIRCLE radius

SHORT FORM : CI radius

HEX FORM : 38 radius

PARAMETER TYPE : radius = Real

DESCRIPTION : CI draws a circle with radius radius centered on the 2D current point. The circle is filled if the PRMFIL flag is set. This command does not affect the 2D current point.

EXAMPLE :

CODE :

ASCII : CI 100

HEX : 38 64 00 00 00

RESULT : A circle with radius 100 is drawn from the 2D current point.

ERRORS : Overflow

RELATED MATERIALS : AREAPT, ARC, ELIPSE, LINFUN, LINPAT, PRMFIL, SECTOR, Section 3.6

COMMAND :

LONG FORM : CLBEG clist

SHORT FORM : CB clist

HEX FORM : 70 clist

PARAMETER TYPE : clist = Char

DESCRIPTION : CLBEG begins the definition of the command list number clist. Commands are saved, without being executed, in the command list definition area. Defining a list using an already existing clist will overwrite the old command list. A command list may be up to 64Kbytes long.

EXAMPLE :

CODE :

ASCII : CB 2

HEX : 70 02

RESULT : Command list 2 is started.

ERRORS : Not enough memory, command list running, clist > 64K in size

RELATED MATERIALS : CLEND, CLOOP, CLDEL, CLMOD, CLRD, CLRUN, Section 3.9

CLDEL
(Command List Delete)

COMMAND :

LONG FORM : CLDEL clist

SHORT FORM : CD clist

HEX FORM : 74 clist

PARAMETER TYPE : clist = Char

DESCRIPTION : CLDEL deletes the definition of the command list specified by clist.

EXAMPLE :

CODE :

ASCII : CD 2
HEX : 74 02

RESULT : Command list 2 is deleted.

ERRORS : Command list running

RELATED MATERIALS : CLBEG, CLEND, Section 3.9

COMMAND :

LONG FORM : CLEARS index

SHORT FORM : CLS index

HEX FORM : 0F index

PARAMETER TYPE : index = Char

DESCRIPTION : CLEARS sets all the pixels in the display buffer to the color designated by index regardless of the value of MASK. The current color is not changed.

Note: This command does not affect only the visible VRAM, but also the hidden space. If you want to clear only the visible buffer, use the FLOOD command.

EXAMPLE :

CODE :

ASCII : CLS 17

HEX : 0F 11

RESULT : Screen is filled with color 17.

ERRORS : None

RELATED MATERIALS : FLOOD, Section 3.1, Section 3.7

CLEND
(Command List End)

COMMAND :

LONG FORM : CLEND

SHORT FORM : CE

HEX FORM : 71

PARAMETER TYPE : = None

DESCRIPTION : CLEND ends the command list currently being defined. After a CLEND, the controller resumes executing commands as they are received.

EXAMPLE :

CODE :

ASCII : CE
HEX : 71

RESULT : The command list being defined is ended.

ERRORS : None

RELATED MATERIALS : CLBEG, CLDEL, Section 3.9

COMMAND :

LONG FORM : **CLIPH flag**

SHORT FORM : **CH flag**

HEX FORM : **AA flag**

PARAMETER TYPE : **flag = Char [0..1]**

DESCRIPTION : **CLIPH enables or disables hither plane clipping. Setting flag to 0 disables hither plane clipping; setting flag to 1 enables it.**

EXAMPLE :

CODE :

ASII : **CH 1**

HEX : **AA 01**

RESULT : **Hither clipping is enabled.**

ERRORS : **None**

RELATED MATERIALS : **DISTH, Subsection 3.4.2**

CLIPY
(Clip Yon)

COMMAND :

LONG FORM : CLIPY flag

SHORT FORM : CY flag

HEX FORM : AB flag

PARAMETER TYPE : flag = Char [0..1]

DESCRIPTION : CLIPY enables or disables yon plane clipping. Setting flag to 0 disables yon plane clipping; setting flag to 1 enables it.

EXAMPLE :

CODE :

ASCII : CY 1
HEX : AB 01

RESULT : Yon clipping is enabled.

ERRORS : None

RELATED MATERIALS : DISTY, Subsection 3.4.2

CLOOP
(Command List Loop)

COMMAND :

LONG FORM : CLOOP clist count

SHORT FORM : CL clist count

HEX FORM : 73 clist count

PARAMETER TYPE : clist = Char
count = Unsigned Int

DESCRIPTION : CLOOP executes the command list clist, count times.

EXAMPLE :

CODE :

ASCII : CL 4 300

HEX : 73 04 2C 01

RESULT : Command list 4 is executed 300 times.

ERRORS : Command list running, stack full

RELATED MATERIALS : CLRUN, Section 3.9

CLMOD
(Command List Modify)

COMMAND :

> ***LONG FORM :*** **CLMOD clist, offset, nbytes, bytes**
>
> ***SHORT FORM :*** **CM clist, offset, nbytes, bytes**
>
> ***HEX FORM :*** **78 clist, offset, nbytes, bytes**

PARAMETER TYPE : **clist = Char**
offset = Unsigned Int
nbytes = Unsigned Int
bytes = nbyte's of Char

DESCRIPTION : **CLMOD replaces nbytes bytes in command list clist, starting at byte number offset from the start of the command list, with the bytes contained in bytes. Note that bytes cannot be added or deleted, only replaced.**

EXAMPLE :

CODE :

ASCII : **CM 3 7 2 175 8**

HEX : **78 03 07 00 02 00 AF 08**

RESULT : **The two bits in command list 3 with offsets 7 and 8 are replaced with CONVRT and POINT commands.**

ERRORS : **None**

RELATED MATERIALS : **CLRD, NOOP, Section 3.9**

COMMAND :

LONG FORM : **CLRD clist**

SHORT FORM : **CRD clist**

HEX FORM : **76 clist**

PARAMETER TYPE : **clist = Char**

DESCRIPTION : **CLRD sends the information stored in command list clist (hex form of the command) to the output channel. The first word in the data stream represents the number of bytes in the command list. It is followed by the bytes as they are stored.**

EXAMPLE :

CODE :

ASCII : **CRD 7**

HEX : **76 07**

RESULT : **Command list 7 is listed to the read back buffer in hex.**

ERRORS : **None**

RELATED MATERIALS : **CLBEG, CLEND, CLDEL, Section 3.9**

CLRUN
(Execute Command List)

COMMAND :

LONG FORM : CLRUN clist

SHORT FORM : CR clist

HEX FORM : 72 clist

PARAMETER TYPE : clist = Char

DESCRIPTION : CLRUN executes the commands in command list clist.

EXAMPLE :

CODE :

ASCII : CR 3

HEX : 72 03

RESULT : Command list 3 is executed.

ERRORS : Command list running, stack full

RELATED MATERIALS : CLBEG, CLEND, Section 3.9

COMMAND :

LONG FORM : COLMOD mode

SHORT FORM : CLM mode

HEX FORM : CA mode

PARAMETER TYPE : mode = Char [0 or 1]

DESCRIPTION : Under certain conditions primitives may generate both a background and a foreground. When we draw a patterned line, for example, the pattern is made up of a foreground and a background, a character cell has a foreground and a background, and any of the commands that produce filled areas produce a foreground and a background if the fill is in the form of a pattern. In such a case, the COLMOD command determines whether the background is transparent or is the color last specified by the BCOLOR command.

When mode is 0, this command sets the board to Replace Color Mode, with the result that backgrounds are given the background color set by the most recent BCOLOR command.

When mode is 1, this command sets the board to Foreground Color Mode, with the result that backgrounds are drawn to be transparent.

Note that no background is drawn if the character type is graphic(vector text) and the cell rotation (TCHROT) is not a multiple of 90o.

Default is Foreground Color Mode.

EXAMPLE :

CODE :

ASCII : COLMOD 0

HEX : CA 00

RESULT : The board enters Replace Color Mode.

ERRORS : **Range**

RELATED MATERIALS : **BCOLOR, AREAPT, LINPAT, TEXT, Section 3.5, Section 3.8**

COLOR
(Color)

COMMAND :

LONG FORM : COLOR index

SHORT FORM : C index

HEX FORM : 06 index

PARAMETER TYPE : index = Char

DESCRIPTION : COLOR sets the current color to index.

EXAMPLE :

CODE :

ASCII : C 12

HEX : 06 0C

RESULT : The current color is set to color 12.

ERRORS : Value out of range(ASCII only)

RELATED MATERIALS : Section 3.5

CONVRT
(Convert)

COMMAND :

LONG FORM : CONVRT

SHORT FORM : CV

HEX FORM : AF

PARAMETER TYPE : None

DESCRIPTION : CONVRT maps the 3D current point to the 2D current point.

EXAMPLE :

CODE :

ASCII : CV

HEX : AF

RESULT : The 3D current point is mapped to 2D and placed in the 2D current point.

ERRORS : Overflow

RELATED MATERIALS : Section 3.6.3, Section 3.7

CX
(Communications Hexadecimal)

COMMAND :

LONG FORM : CX␣

SHORT FORM : CX␣

HEX FORM : 43 58 20

PARAMETER TYPE : None

DESCRIPTION : CX sets the communication mode to hexadecimal. This command may be given when in either ASCII mode or Hex mode. Note that the Hex and ASCII forms of this command are identical.

EXAMPLE :

CODE :

ASCII : CX␣

HEX : 43 58 20 OR D1

Note: You can user either of the 2 hex formats given above to issue this command; however, the PG-640Aalways uses D1 in command lists that it creates.

RESULT : The communication mode is set to hexadecimal.

ERRORS : None

RELATED MATERIALS : CA, Subsection 3.2.3

COMMAND :

LONG FORM : DISPLA flag

SHORT FORM : DI flag

HEX FORM : D0 flag

PARAMETER TYPE : flag = Char [0..1]

DESCRIPTION : DISPLA displays either the high level graphics screen (flag = 0) or the emulator graphics screen (flag = 1). In either case, high level graphics commands are accepted and executed. If the emulator enable dip switch is off, high level graphic will always be displayed

EXAMPLE :

CODE :

ASCII : DI 1

HEX : D0 01

RESULT : Emulator screen is displayed.

ERRORS : None

RELATED MATERIALS : Section 3.3, Appendix A.

COMMAND :

LONG FORM : DISTAN dist

SHORT FORM : DS dist

HEX FORM : B1 dist

PARAMETER TYPE : dist = Real

DESCRIPTION : DISTAN sets the distance from the eye to the viewing reference point. This only affects 3D drawing. The default distance is 500.

EXAMPLE :

CODE :

ASCII : DS 1200

HEX : B1 B0 04 00 00

RESULT : Distance to viewing reference point is set to 1200.

ERRORS : None

RELATED MATERIALS : PROJCT, Subsection 3.4.2

DISTH
(Distance Hither)

COMMAND :

> ***LONG FORM :*** **DISTH dist**
>
> ***SHORT FORM :*** **DH dist**
>
> ***HEX FORM :*** **A8 dist**

PARAMETER TYPE : **dist = Real**

DESCRIPTION : **DISTH sets the distance from the viewing reference point to the hither clip plane. When hither clipping is enabled, no points farther from the viewer than the hither plane are displayed. The hither plane is parallel to the viewplane. Hither clipping affects only 3D drawing.**

EXAMPLE :

> ***CODE :***
>
> > ***ASCII :*** **DH -12.00**
> >
> > ***HEX :*** **A8 F4 FF 00 00**
>
> ***RESULT :*** **The hither plane is defined to be 12.00 units in front of the viewplane.**

ERRORS : **None**

RELATED MATERIALS : **CLIPH, Subsection 3.4.2**

DISTY
(Distance Yon)

COMMAND :

LONG FORM : DISTY dist

SHORT FORM : DY dist

HEX FORM : A9 dist

PARAMETER TYPE : dist = Real

DESCRIPTION : DISTY sets the distance from the viewing reference point to the yon clip plane. When yon clipping is enabled, no points closer to the viewer than the yon plane are displayed. The yon plane is parallel to the viewplane. Yon clipping affects only 3D drawing.

EXAMPLE :

CODE :

ASCII : DY 12.00

HEX : A9 0C 00 00 00

RESULT : The yon plane is defined to be 12.00 units behind the viewplane.

ERRORS : None

RELATED MATERIALS : CLIPY, Subsection 3.4.2

DRAW
(Draw)

COMMAND :

LONG FORM : DRAW x y

SHORT FORM : D x y

HEX FORM : 28 x y

PARAMETER TYPE : x = Real
y = Real

DESCRIPTION : DRAW draws a line from the 2D current point to {x,y} and positions the 2D current point at {x,y}. Both the first and the last pixels of the line are drawn.

EXAMPLE :

CODE :

ASCII : D 10.0 12.0

HEX : 28 0A 00 00 00 0C 00 00 00

RESULT : A line is drawn from the 2D current point to {10.0,12.0}.

ERRORS : Arithmetic overflow

RELATED MATERIALS : DRAWR, LINFUN, LINPAT, MOVE, MOVER
Section 3.6

COMMAND :

LONG FORM : DRAWR Δx Δy

SHORT FORM : DR Δx Δy

HEX FORM : 29 Δx Δy

PARAMETER TYPE : Δx = Real
Δy = Real

DESCRIPTION : DRAWR draws a line from the 2D current point to ($\{\Delta x, \Delta y\}$ + 2D current point). The 2D current point is moved to the end of the line. Both the first and the last pixels of the line are drawn.

EXAMPLE :

CODE :

ASCII : DR.100.00 200.00

HEX : 29 64 00 00 00 C8 00 00 00

RESULT : A line is drawn from the 2D current point to (the 2D current point + {100.00,200.00}).

ERRORS : Arithmetic overflow

RELATED MATERIALS : DRAW, LINFUN, LINPAT, MOVE, MOVER, Section 3.6

DRAW3
(Draw in 3D)

COMMAND :

LONG FORM : DRAW3 x y z

SHORT FORM : D3 x y z

HEX FORM : 2A x y z

PARAMETER TYPE : x = Real
y = Real
z = Real

DESCRIPTION : DRAW3 draws a line from the 3D current point to {x,y,z} and moves the current point to {x,y,z}.

EXAMPLE :

CODE :

ASCII : D3 5.0 10.0 12.0

HEX : 2A 05 00 00 00 0A 00 00 00 0C 00 00 00

RESULT : A line is drawn from the 3D current point to {5.0,10.0,12.(

ERRORS : Arithmetic overflow

RELATED MATERIALS : DRAWR3, LINFUN, LINPAT, MOVE3, MOVER3, Section 3.6

COMMAND :

LONG FORM : DRAWR3 Δx Δy Δz

SHORT FORM : DR3 Δx Δy Δz

HEX FORM : 2B Δx Δy Δz

PARAMETER TYPE : Δx = Real
Δy = Real
Δz = Real

DESCRIPTION : DRAWR3 draws a line from the 3D current point to ($\{\Delta x, \Delta y, \Delta z\}$ + the current point) and moves the current point to the end of the line.

EXAMPLE :

CODE :

ASCII : DR3 5.0 10.0 12.0

HEX : 2B 05 00 00 00 0A 00 00 00 0C 00 00 00

RESULT : A line is drawn from the 3D current point to ({5.0,10.0,12.0} + 3D current point).

ERRORS : Arithmetic overflow

RELATED MATERIALS : DRAW3, LINFUN, LINPAT, MOVE3, MOVER3, Section 3.6

COMMAND :

LONG FORM : ELIPSE xradius yradius

SHORT FORM : EL xradius yradius

HEX FORM : 39 xradius yradius

PARAMETER TYPE : xradius = Real
yradius = Real

DESCRIPTION : ELIPSE draws a 2D ellipse centered on the 2D current point. Its x and y radii, which are parallel to the screen's x and y axes, are given by xradius and yradius. The ellipse will be filled if drawn while the PRMFIL flag is set. This command does not affect the 2D current point.

EXAMPLE :

CODE :

ASCII : EL 32.00 128.00

HEX : 39 20 00 00 00 80 00 00 00

RESULT : An ellipse is drawn with x radius 32 and y radius 128.

ERRORS : Overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, PRMFIL, Section 3.6

COMMAND :

LONG FORM : **FILMSK mask**

SHORT FORM : **FM mask**

HEX FORM : **EF mask**

PARAMETER TYPE : **mask = Char**

DESCRIPTION : **FILMSK defines the area fill mask to be the value mask. When an area fill command tests for a boundary index, pixel data is ANDed against this mask as well as MASK. Default value is no mask.**

EXAMPLE :

CODE :

ASCII : **FM 126**

HEX : **EF 7E**

RESULT : **Area fill mask is set to 126.**

ERRORS : **None**

RELATED MATERIALS : **AREA, AREABC, MASK, Section 3.7**

FLAGRD
(Flag Read)

COMMAND :

LONG FORM : FLAGRD flag

SHORT FORM : FRD flag

HEX FORM : 51 flag

PARAMETER TYPE : flag = Char [1..30]

DESCRIPTION : FLAGRD places the current value of the flag specified by flag into the read back buffer. Data are read back using the current communications mode using the same format as the instructions that wrote them. The values of flag are specified in the table on the following page.

EXAMPLE :

CODE :

ASCII : FRD 25

HEX : 51 19

RESULT : The amount of free memory is placed in the read back buffer.

ERRORS : No such flag

RELATED MATERIALS : RESETF, Section 3.12

Flag	Name	Type of Value
1	AREAPT	16 Ints
2	CLIPH	1 Char
3	CLIPY	1 Char
4	COLOR	1 Char
5	DISPLA	1 Char
6	DISTAN	1 Real
7	DISTH	1 Real
8	DISTY	1 Real
9	FILMSK	1 Char
10	LINFUN	1 Char
11	LINPAT	1 Int
12	MASK	1 Char
13	MDORG	3 Reals
14	2-D current point	2 Reals
15	3-D current point	3 Reals
16	PRMFIL	1 Char
17	PROJCT	1 Int
18	TANGLE	1 Int
19	TJUST	2 Chars
20	TSIZE	1 Real
21	VWPORT	4 Ints
22	VWRPT	3 Reals
23	WINDOW	4 Reals
24	transformed 3- D current point	3 Reals
25	free memory	1 Int
26	current position of XHAIR	2 Ints
27	2-D position of XHAIR	2 Reals
28	Screen Current Point	2 Ints
29	free memory	1 Real*
30	TWVIS	1 Char
31	TWPOS	6 Ints
32	TSTYLE	1 Char
33	TASPCT	1 Real
34	TCHROT	1 Int
41	COLMOD	1 Char
42	BCOLOR	1 Char

* This value is treated as a double precision integer

FLOOD
(Flood)

COMMAND :

LONG FORM : FLOOD index

SHORT FORM : F index

HEX FORM : 07 index

PARAMETER TYPE : index = Char

DESCRIPTION : FLOOD sets all the pixels in the defined viewport to the color specified by index. The current color is not changed and the command is subject to MASK.

EXAMPLE :

CODE :

ASCII : F 12
HEX : 07 0C

RESULT : The rectangular area defined by the viewport is filled with color 12.

ERRORS : None

RELATED MATERIALS : CLEARS, MASK, Section 3.7

GTDEF
(Graphics Text Font Define)

COMMAND :

LONG FORM : GTDEF ch n width array

SHORT FORM : GTD ch n width array

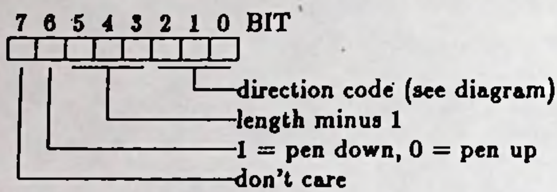
HEX FORM : 89 ch n width array

PARAMETER TYPE : ch = Char
n = Char
width = Char [1..12]
array = n values

DESCRIPTION : GTDEF defines the character given by ch in the user font as a series of vector plots stored in the n values of array. The width of the character cell is given by width and the height is fixed at 12. The starting point for the definition is at {0,3} of the character cell. Each value in the array consists of three parts: the pen action, the length, and the direction. The pen action may be move (pen action = 0) or write (pen action = 1). The length may take a value from one to eight. The direction can be from 0 to 7 and is summarized in the diagram below.

3		2		1
	↖	↑	↗	
4	←		→	0
	↙	↓	↘	
5		6		7

Each of these values is specified by a separate number when in ASCII mode. In Hex mode, the values are packed into a single byte with the three low bits containing the direction, the next three bits containing the length less one and the seventh bit containing the pen action. The format of the vector parameter is shown in the following diagram:



Notes :

- Any previous definition is lost. To reset a character to its default value specify n as 00.
- Specifying characters using this command (rather than TDEFIN) will allow the characters to be enlarged and rotated.
- If you plan to define an entire font, it is faster to reset all previous characters starting by the last character (255, 254, 253, ..., 0) and then define the character font starting at 0, 1, 2, ..., 255.

EXAMPLE :

CODE :

ASCII : GTD 65 7 8 1 7 2
1 2 1
1 3 0
1 2 7
1 7 6
0 4 2
1 7 4

HEX : 89 41 07 08 72 49 50 4F 76 1A 74

RESULT : The letter "A" is defined.

ERRORS : Not enough memory, Bad definition

RELATED MATERIALS : Section 3.8

COMMAND :

LONG FORM : IMAGER line x1 x2

SHORT FORM : IR line x1 x2

HEX FORM : D8 line x1 x2

PARAMETER TYPE : line = Unsigned Int [0..479]
x1 = Unsigned Int [0..639]
x2 = Unsigned Int [0..639]

DESCRIPTION : IMAGER reads pixel values from the image currently being displayed and sends these values to the read back buffer. Parameters line, x1 and x2 are measured in pixels from the lower left corner of the screen. When the communication mode is set to ASCII, the values of the pixels are sent as ASCII numbers separated by commas. When the communication mode is set to hex, then the output is sent in run-length encoded format (see Section 3.10).

EXAMPLE :

CODE :

ASCII : IR 50 0 256

HEX : D8 32 00 00 00 00 01

RESULT : The values of pixels 0 through 256 from line 50 are sent to the read back buffer.

ERRORS : Value out of range

RELATED MATERIALS : CA, CX, IMAGEW, Section 3.10

COMMAND :

LONG FORM : IMAGEW line x1 x2 data

SHORT FORM : IW line x1 x2 data

HEX FORM : D9 line x1 x2 data

PARAMETER TYPE : line = Unsigned Int [0..479]
x1 = Unsigned Int [0..639]
x2 = Unsigned Int [0..639]
data = ASCII: string of Chars
Hex: run length encoded string

DESCRIPTION : IMAGEW writes pixel values to the image currently being displayed. Parameters line, x1 and x2 are measured in pixels from the lower left corner of the screen. When the communication mode is set to ASCII, the values of the pixels are expected to be ASCII numbers separated by blanks. When the communication mode is set to hex, the input is expected be in run-length encoded format. In run length encoded form the user sends either byte pairs or a count and a string of bytes. When the high bit of the first byte is not set, a byte pair is expected: the first byte represents the count less one, the second byte the pixel value to be repeated count times. If the high bit is set, then the first byte is the length less one of the byte string which follows. In both cases the count and the length only use the low seven bits for the value. See Section 3.10 for more information on run-length encoding.

EXAMPLE :

CODE :

ASCII : IW 50 0 10 0 0 0 0 0 0 0 0 0 0 0

HEX : D9 32 00 00 00 0A 00 0B 00

RESULT : The values of pixels 0 through 10 of line 50 are set to 0

ERRORS : **Value out of range**

RELATED MATERIALS : **CA, CX, IMAGER, Section 3.10, Section 3.14.**

LINFUN
(Line Function)

COMMAND :

LONG FORM : LINFUN function

SHORT FORM : LF function

HEX FORM : EB function

PARAMETER TYPE : function = Char [0..4]

DESCRIPTION : LINFUN sets the drawing function to the function specified by the following table.

function	Mode
0	Replace Mode
1	Complement Mode
2	XOR
3	OR
4	AND

When Replace Mode is selected, drawing is done in the current color. Choosing Complement Mode will complement each pixel as it is drawn – the current color will be ignored. The remaining modes perform the specified logic operation between the pixel and the current color. Drawing is subject to MASK.

EXAMPLE :

CODE :

ASCII : LF 0

HEX : EB 00

RESULT : Drawing is performed in the current color.

ERRORS : None

RELATED MATERIALS : MASK, Subsection 3.5.1

COMMAND :

LONG FORM : LINPAT pattern

SHORT FORM : LP pattern

HEX FORM : EA pattern

PARAMETER TYPE : pattern = Unsigned Int

DESCRIPTION : LINPAT sets the line drawing pattern mask to pattern. Each of the 16 bits in pattern represents a pixel in subsequently drawn lines. The pattern is repeated along the entire length of the line drawn when using one of the following commands (and PRMFIL is not set, in the case of closed figures):

ARC, CIRCLE, DRAW, DRAWR, DRAW3, DRAWR3, ELIPSE, POLY, POLYR, POLY3, POLYR3, RECT, RECTR, SECTOR.

EXAMPLE :

CODE :

ASCII : LP 255

HEX : EA FF 00

RESULT : Dashed lines are drawn when the above commands are used.

ERRORS : None

RELATED MATERIALS : BCOLOR, COLMOD, LINFUN, PRMFIL, Subsection 3.5.3

LUT
(Lookup Table)

COMMAND :

LONG FORM : LUT index r g b

SHORT FORM : L index r g b

HEX FORM : EE index r g b

PARAMETER TYPE : index = Char
r = Char [0..15]
g = Char [0..15]
b = Char [0..15]

DESCRIPTION : LUT loads the three RGB intensity values into the LUT entry specified by index. The values sent by this command are loaded into the high order nibbles of the lookup table entry; the low order nibbles are set to zero. The LUTX is the preferred form of the command.

EXAMPLE :

CODE :

ASCII : L 15 2 4 8

HEX : EE 0F 02 04 08

RESULT : LUT entry 15 is set to r = 32, g = 64 and b = 128.

ERRORS : Out of range

RELATED MATERIALS : LUTINT, LUTRD, LUTSAV, LUTSTO, LUTX LUTXRD, Subsection 3.5.2

COMMAND :

LONG FORM : LUTINT state

SHORT FORM : LI state

HEX FORM : EC state

PARAMETER TYPE : state = Char

DESCRIPTION : LUTINT sets the LUT to the state specified by the following table.

state	Description
0	Color cone distribution
1	Foreground/background colors in the low 4 bits of a value code will be visible only if the high 4 bits are 0 (or invisible)
2	Value codes interpreted as R R G G G B B B
3	Value codes interpreted as R R R G G B B B
4	Value codes interpreted as R R R G G G B B
5	6 level RGB
253	Alternately load from LUT storage area 0 and 1
254	Load LUT from LUT storage area 1
255	Load LUT from LUT storage area 0

EXAMPLE :

CODE :

ASCII : LI 255

HEX : EC FF

RESULT : LUT is loaded from LUT storage area.

ERRORS : Value out of range

RELATED MATERIALS : LUT, LUTRD, LUTSAV, LUTSTO, LUTX, LUTXRD, Subsection 3.5.3

LUTRD
(Lookup Table Read)

COMMAND :

LONG FORM : LUTRD index

SHORT FORM : LRD index

HEX FORM : 50 index

PARAMETER TYPE : index = Char

DESCRIPTION : LUTRD reads high order nibbles of the red, green and blue values of the LUT entry specified by index and sends them to the output buffer. In ASCII mode, the three values are ASCII numbers separated by commas and terminated by a carriage return. In Hex mode, the high order nibbles of LUT values are sent in the low order nibbles of three bytes, one byte for each color. LUTXRD is the preferred form of the command.

EXAMPLE :

CODE :

ASCII : LRD 25

HEX : 50 19

RESULT : Values of the high nibbles of the LUT entry 19 are returned.

ERRORS : None

RELATED MATERIALS : CA, CX, LUT, LUTINT, LUTSAV, LUTSTO, LUTX, LUTXRD, Subsection 3.5.3

COMMAND :

LONG FORM : **LUTSAV**

SHORT FORM : **LS**

HEX FORM : **ED**

PARAMETER TYPE : **None**

DESCRIPTION : **LUTSAV writes all 256 LUT entries to LUT storage area 0. These values may be reloaded into the LUT using a LUTINT 255 command. Each LUTSAV command overwrites any LUT data previously saved.**

EXAMPLE :

CODE :

ASCII : **LS**

HEX : **ED**

RESULT : **LUT data is stored in the LUT storage area.**

ERRORS : **None**

RELATED MATERIALS : **LUT, LUTINT, LUTRD, LUTSTO, LUTX, LUTXRD, Subsection 3.5.3**

LUTSTO
(LUT Store)

COMMAND :

LONG FORM : LUTSTO flag

SHORT FORM : LST flag

HEX FORM : C9 flag

PARAMETER TYPE : flag = Char [0..1]

DESCRIPTION : LUTSTO saves the current lookup table in one of two user areas. Note that LUTSAV and LUTSTO 0 are identical. Table 0 can be recalled by LUTINT 255 and Table 1 by LUTINT 254. Each LUTSTO command overwrites any LUT data previously saved in the specified user area.

EXAMPLE :

CODE :

ASCII : LST 1

HEX : C9 01

RESULT : The current LUT values are stored in Table 1.

ERRORS : None

RELATED MATERIALS : LUT, LUTINT, LUTSAV, LUTSTO, LUTX, LUTXRD, Subsection 3.5.3

COMMAND :

LONG FORM : LUTX index r g b

SHORT FORM : LX index r g b

HEX FORM : E6 index r g b

PARAMETER TYPE : index = Char
r = Char [0..255]
g = Char [0..255]
b = Char [0..255]

DESCRIPTION : LUTX loads the three eight-bit RGB intensity values into the lookup table entry specified by index.

EXAMPLE :

CODE :

ASCII : LX 15 2 4 8

HEX : E6 0F 02 04 08

RESULT : Lookup table entry 15 is set to r = 2, g = 4 and b = 8.

ERRORS : None

RELATED MATERIALS : LUTINT, LUTRD, LUTSAV, LUTSTO, LUTXRD, Subsection 3.5.3

LUTXRD
(Lookup Table Read – 8 Bit)

COMMAND :

LONG FORM : LUTXRD index

SHORT FORM : LXR index

HEX FORM : 53 index

PARAMETER TYPE : index = Char

DESCRIPTION : LUTXRD reads the red, green and blue values of the LUT entry specified by index and sends them to the output buffer. In ASCII mode, the three values are ASCII numbers separated by commas and terminated by a carriage return. In Hex mode, the LUT values are sent in three bytes, one byte for each color. Each LUT value is in the range 0 to 255.

EXAMPLE :

CODE :

ASCII : LXR 25

HEX : 53 19

RESULT : Values of LUT entry 19 are returned.

ERRORS : None

RELATED MATERIALS : CA, CX, LUTX, LUTINT, LUTSAV, LUTSTO, Subsection 3.5.3

COMMAND :

LONG FORM : MASK planemask

SHORT FORM : MK planemask

HEX FORM : E8 planemask

PARAMETER TYPE : planemask = Char

DESCRIPTION : MASK sets the 8-bit read/write pixel data bit plane mask to the value contained in planemask. Each bit in planemask will enable the corresponding bit plane in the video buffer to be read or written. Zeroes written to all 8 bits will prevent data from being written to any pixel data bit plane and will cause 0's to be returned when pixel data are read.

EXAMPLE :

CODE :

ASCII : MK 255

HEX : E8 FF

RESULT : All bit planes can be read or written.

ERRORS : None

\ *RELATED MATERIALS :* Subsection 3.5.4

MATXRD
(Matrix Read)

COMMAND :

LONG FORM : MATXRD matrix

SHORT FORM : MRD matrix

HEX FORM : 52 matrix

PARAMETER TYPE : matrix = Char [1..2]

DESCRIPTION : MATXRD copies the contents of the matrix specified by matrix to the read back buffer. When matrix is 1, the contents of the 3D modelling transformation matrix are copied, when matrix is 2 the contents of the 3D viewing transformation matrix are copied. In ASCII mode, the matrix elements are written in four lines, each of which has four entries separated by commas and terminated by a carriage return. In Hex mode, each matrix element is written as four bytes with the following reading order.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

EXAMPLE :

CODE :

ASCII : MRD 2

HEX : 52 02

RESULT : The contents of the viewing transformation matrix are copied to the Data Out Register.

ERRORS : Value out of range

RELATED MATERIALS : CA, CX, Section 3.4.2

COMMAND :

LONG FORM : MDIDEN

SHORT FORM : MDI

HEX FORM : 90

PARAMETER TYPE : None

DESCRIPTION : MDIDEN sets the modelling transformation matrix to the identity matrix.

EXAMPLE :

CODE :

ASCII : MDI

HEX : 90

RESULT : The modelling transformation matrix is set to the identity matrix.

ERRORS : None

RELATED MATERIALS : DRAW3, DRAWR3, MDMATX, MOVE3, MOVER3, POINT3, POLY3, POLYR3, Subsection 3.4.2

MDMATX
(Modelling Matrix)

COMMAND :

LONG FORM : MDMATX array

SHORT FORM : MDM array

HEX FORM : 97 array

PARAMETER TYPE : array = 16 Reals

DESCRIPTION : MDMATX loads the modelling matrix directly from the data in array.

EXAMPLE :

CODE :

ASCII : MDM 36.25 12.00 128 2 0 36.75 100 0
72.5 0 2.5 0 100.25 0 0 0

HEX : 97 24 00 00 40
0C 00 00 00
80 00 00 00
02 00 00 00
00 00 00 00
24 00 00 C0
64 00 00 00
00 00 00 00
52 00 00 80
00 00 00 00
02 00 00 80
00 00 00 00
64 00 00 40
00 00 00 00
00 00 00 00
00 00 00 00

RESULT : The modelling matrix is set to the above data.

ERRORS : Arithmetic overflow

RELATED MATERIALS : MDORG, MDROTX, MDROTY, MDROTZ, MATXRD, Subsection 3.4.2

COMMAND :

LONG FORM : MDORG ox oy oz

SHORT FORM : MDO ox oy oz

HEX FORM : 91 ox oy oz

PARAMETER TYPE : ox = Real
oy = Real
oz = Real

DESCRIPTION : MDORG defines the origin section of the modelling transformation matrix used in modelling transformation scaling and rotating.

EXAMPLE :

CODE :

ASCII : MDO 0.0 12.5 1.0

HEX : 91 00 00 00 00 0C 00 00 80 01 00 00 00

RESULT : Origin is defined as x = 0, y = 12.5 and z = 1.

ERRORS : None

RELATED MATERIALS : MDROTX, MDROTY, MDROTZ, MATXRD, Subsection 3.4.2

MDROTX
(Modelling Rotate X Axis)

COMMAND :

LONG FORM : MDROTX angle

SHORT FORM : MDX angle

HEX FORM : 93 angle

PARAMETER TYPE : angle = Int

DESCRIPTION : MDROTX rotates the object about the x axis by angle.

EXAMPLE :

CODE :

ASCII : MDX 45

HEX : 93 2D 00

RESULT : The object is rotated by 45° about the x axis.

ERRORS : Arithmetic overflow

RELATED MATERIALS : MDMATX, MDORG, MDROTY, MDROTZ, Subsection 3.4.2

COMMAND :

LONG FORM : MDROTY angle

SHORT FORM : MDY angle

HEX FORM : 94 angle

PARAMETER TYPE : angle = Int

DESCRIPTION : MDROTY rotates the object about the y axis by angle.

EXAMPLE :

CODE :

ASCII : MDY 45

HEX : 94 2D 00

RESULT : The object is rotated by 45° about the y axis.

ERRORS : Arithmetic overflow

RELATED MATERIALS : MDMATX, MDORG, MDROTX, MDROTZ, Subsection 3.4.2

MDROTZ
(Modelling Rotate Z Axis)

COMMAND :

LONG FORM : MDROTZ angle

SHORT FORM : MDZ angle

HEX FORM : 95 angle

PARAMETER TYPE : angle = Int

DESCRIPTION : MDROTZ rotates the object about the z axis by angle.

EXAMPLE :

CODE :

ASCII : MDZ 45

HEX : 95 2D 00

RESULT : The object is rotated by 45° about the z axis.

ERRORS : Arithmetic overflow

RELATED MATERIALS : MDMATX, MDORG, MDROTX, MDROTY, Subsection 3.4.2

MDSCAL
(Modelling Scale)

COMMAND :

LONG FORM : **MDSCAL sx sy sz**

SHORT FORM : **MDS sx sy sz**

HEX FORM : **92 sx sy sz**

PARAMETER TYPE : **sx = Real**
sy = Real
sz = Real

DESCRIPTION : **MDSCAL changes the scaling component of the modelling matrix for 3D drawing.**

EXAMPLE :

CODE :

ASCII : **MDS 2 4 8**

HEX : **92 02 00 00 00 04 00 00 00 08 00 00 00**

RESULT : **Scaling component is set to {2,4,8}.**

ERRORS : **Arithmetic overflow**

RELATED MATERIALS : **MDMATX, Subsection 3.4.2**

MDTRAN
(Modelling Translation)

COMMAND :

LONG FORM : MDTRAN tx ty tz

SHORT FORM : MDT tx ty tz

HEX FORM : 96 tx ty tz

PARAMETER TYPE : tx = Real
ty = Real
tz = Real

DESCRIPTION : MDTRAN moves the translation component of the modelling matrix for 3D drawing by {tx,ty,tz}.

EXAMPLE :

CODE :

ASCII : MDT 2 4 8
HEX : 96 02 00 00 00 04 00 00 00 08 00 00 00

RESULT : Translation component is set to {2,4,8}.

ERRORS : Arithmetic overflow

RELATED MATERIALS : MDMATX, Subsection 3.4.2

COMMAND :

LONG FORM : MOVE x y

SHORT FORM : M x y

HEX FORM : 10 x y

PARAMETER TYPE : x = Real
y = Real

DESCRIPTION : MOVE moves the 2D current point to {x,y}.

EXAMPLE :

CODE :

ASCII : M 10.0 12.0

HEX : 10 0A 00 00 00 0C 00 00 00

RESULT : The current point is moved to {10.0,12.0}.

ERRORS : Arithmetic overflow

RELATED MATERIALS : MOVER, Section 3.6

MOVER

(Move Relative)

COMMAND :

LONG FORM : MOVER Δx Δy

SHORT FORM : MR Δx Δy

HEX FORM : 11 Δx Δy

PARAMETER TYPE : Δx = Real
Δy = Real

DESCRIPTION : MOVER moves the 2D current point to ({Δx,Δy} + the current point).

EXAMPLE :

CODE :

ASCII : MR 10.0 12.0

HEX : 11 0A 00 00 00 0C 00 00 00

RESULT : The current point is moved to ({10.0,12.0} + the current point).

ERRORS : Arithmetic overflow

RELATED MATERIALS : MOVE, Section 3.6

COMMAND :

LONG FORM : MOVE3 x y z

SHORT FORM : M3 x y z

HEX FORM : 12 x y z

PARAMETER TYPE : x = Real
y = Real
z = Real

DESCRIPTION : MOVE3 moves the 3D current point to {x,y,z}.

EXAMPLE :

CODE :

ASCII : M3 5.0 10.0 12.0

HEX : 12 05 00 00 00 0A 00 00 00 0C 00 00 00

RESULT : The 3D current point is moved to {5.0,10.0,12.0}.

ERRORS : Arithmetic overflow

RELATED MATERIALS : MOVER3, Section 3.6

COMMAND :

LONG FORM : MOVER3 Δx Δy Δz

SHORT FORM : MR3 Δx Δy Δz

HEX FORM : 13 Δx Δy Δz

PARAMETER TYPE : Δx = Real
Δy = Real
Δz = Real

DESCRIPTION : MOVER3 moves the 3D current point by the displacement $\{\Delta x, \Delta y, \Delta z\}$.

EXAMPLE :

CODE :

ASCII : MR3 5.0 10.0 12.0

HEX : 13 05 00 00 00 0A 00 00 00 0C 00 00 00

RESULT : The 3D current point is moved to ({5.0,10.0,12.0} + 3D current point).

ERRORS : Arithmetic overflow

RELATED MATERIALS : MOVE3, Section 3.6

NOOP
(No Operation)

COMMAND :

LONG FORM : NOOP

SHORT FORM : NOP

HEX FORM : 01

PARAMETER TYPE : None

DESCRIPTION : NOOP does nothing. It can be used to hold a byte when editing command lists.

EXAMPLE :

CODE :

ASCII : NOP
HEX : 01

RESULT : Nothing.

ERRORS : None

RELATED MATERIALS : CLMOD, Section 3.9

PDRAW
(Poly Draw)

COMMAND :

> ***LONG FORM :*** PDRAW $x_1, y_1, x_2, y_2, \cdots, x_n, y_n$
>
> ***SHORT FORM :*** PD $x_1, y_1, x_2, y_2, \cdots, x_n, y_n$
>
> ***HEX FORM :*** FF $x_1, y_1, x_2, y_2, \cdots, x_n, y_n$

PARAMETER TYPE : x_i = Int
y_i = Int

DESCRIPTION : PDRAW executes a stream of high speed screen moves and vector draws. This command operates in screen mode and consequently affects the 2D current point. The high bit of the x and y coordinates are used as flags. If the high bit of x_i is set to 1 then the command stream is terminated with the i^{th} coordinate pair. Otherwise the coordinate pair is accepted as a move or draw command. The high bit of the y coordinate is used to distinguish between a current point move (high bit set to 1) and a vector draw (high bit set to 0). The PDRAW command allows the highest drawing speeds to be attained.

Note: An easy way to calculate the value of a decimal number with the high bit set is: $n_{set} = n_o - 32768$. For example, to move to {125,340} one would use the $x = 125$ and $y = 340 - 32768 = -32428$.

EXAMPLE :

CODE :

> ***ASCII :*** PD 96 -32672 0 0 -1 0
>
> ***HEX :*** FF 60 00 60 80 00 00 00 00 FF FF 00 00

RESULT : The current point will be moved to {96,96} and a vector will be drawn to {0,0}.

ERRORS : None

RELATED MATERIALS : Section 3.10

COMMAND :

LONG FORM : POINT

SHORT FORM : PT

HEX FORM : 08

PARAMETER TYPE : None

DESCRIPTION : POINT sets the pixel located at the 2D current point to the current color. This command does not move the 2D current point.

EXAMPLE :

CODE :

ASCII : PT

HEX : 08

RESULT : The pixel at the 2D current point is set to the current color.

ERRORS : None

RELATED MATERIALS : LINFUN, LINPAT, Section 3.6

POINT3
(Point in 3D)

COMMAND :

LONG FORM : POINT3

SHORT FORM : PT3

HEX FORM : 09

PARAMETER TYPE : None

DESCRIPTION : POINT3 sets the pixel located at the 3D current point to the current color. This command does not move the 3D current point.

EXAMPLE :

CODE :

ASCII : PT3
HEX : 09

RESULT : The pixel at the 3D current point is set to the current color.

ERRORS : None

RELATED MATERIALS : LINFUN, LINPAT, Section 3.6

COMMAND :

LONG FORM : POLY n x_1 y_1 x_2 y_2 $\cdots$ x_n y_n

SHORT FORM : P n x_1 y_1 x_2 y_2 $\cdots$ x_n y_n

HEX FORM : 30 n x_1 y_1 x_2 y_2 $\cdots$ x_n y_n

PARAMETER TYPE : n = Char
x_i = Real
y_i = Real

DESCRIPTION : POLY draws a closed polygon in 2D. Parameter n is the number of vertices and $\{x_i,y_i\}$ the coordinates of the vertices. The polygon will be filled if the PRMFIL flag is set and subject to the LINPAT if PRMFIL is not set. The 2D current point will not be changed.

EXAMPLE :

CODE :

ASCII : P 4 0 0 16 0 16 16 0 16

HEX :
```
30 04 00 00 00 00 00 00 00 00 10
      00 00 00 00 00 00 00 10 00 00
      00 10 00 00 00 00 00 00 00 10
      00 00 00
```

RESULT : A square, 16 by 16, is drawn.

ERRORS : Not enough memory, arithmetic overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, POLYR, PRMFIL, Section 3.6

POLYR
(Polygon Relative)

COMMAND :

LONG FORM : POLYR n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n

SHORT FORM : PR n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n

HEX FORM : 31 n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n

PARAMETER TYPE : n = Char
Δx_i = Real
Δy_i = Real

DESCRIPTION : POLYR draws a closed polygon in 2D. Parameter n is the number of vertices and $\{\Delta x_i, \Delta y_i\}$ the displacements from the current point of the vertices. The polygon will be filled if the PRMFIL flag is set and subject to the LINPAT if PRMFIL is not set. The 2D current point will not be changed.

EXAMPLE :

CODE :

ASCII : PR 4 0 0 16 0 16 16 0 16

HEX :
```
31  04 00 00 00 00 00 00 00 00 10
    00 00 00 00 00 00 00 10 00 00
    00 10 00 00 00 00 00 00 00 10
    00 00 00
```

RESULT : A square, 16 by 16, is drawn with the lower left corner on the current point.

ERRORS : Not enough memory, arithmetic overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, POLY, PRMFIL, Section 3.6

COMMAND :

LONG FORM : POLY3 n x_1 y_1 z_1 $\cdots$ x_n y_n z_n

SHORT FORM : P3 n x_1 y_1 z_1 $\cdots$ x_n y_n z_n

HEX FORM : 32 n x_1 y_1 z_1 $\cdots$ x_n y_n z_n

PARAMETER TYPE : n = Char
x_i = Real
y_i = Real
z_i = Real

DESCRIPTION : POLY3 draws a closed polygon where n is the number of vertices and $\{x_i, y_i, z_i\}$ the coordinates of the vertices. The polygon is filled if the PRMFIL flag is set and subject to the LINPAT if PRMFIL is not set. The 3D current point is not changed.

EXAMPLE :

CODE :

ASCII : P3 4 0 0 0 16 0 0 16 0 16 0 0 16

HEX : 32 04 00 00 00 00 00 00 00 00 00
00 00 00 10 00 00 00 00 00 00
00 00 00 00 00 10 00 00 00 00
00 00 00 10 00 00 00 00 00 00
00 00 00 00 00 10 00 00 00

RESULT : A square, 16 by 16, is drawn along the xz plane.

ERRORS : Not enough memory, arithmetic overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, POLYR3, PRMFIL, Section 3.6

POLYR3
(Polygon Relative in 3D)

COMMAND :

LONG FORM : POLYR3 n Δx_1 Δy_1 Δz_1 $\cdots$ Δx_n Δy_n Δz_n

SHORT FORM : PR3 n Δx_1 Δy_1 Δz_1 $\cdots$ Δx_n Δy_n Δz_n

HEX FORM : 33 n Δx_1 Δy_1 Δz_1 $\cdots$ Δx_n Δy_n Δz_n

PARAMETER TYPE : n = Char
Δx_i = Real
Δy_i = Real
Δz_i = Real

DESCRIPTION : POLYR3 draws a closed polygon where n is the number of vertices and $\{\Delta x_i, \Delta y_i, \Delta z_i\}$ the displacements from the current point of the vertices. The polygon is filled if the PRMFIL flag is set and subject to LINPAT if PRMFIL is not set. The 3D current point is not changed.

EXAMPLE :

CODE :

ASCII : PR3 4 0 0 0 16 0 0 16 0 16 0 0 16

HEX :
```
33  04 00 00 00 00 00 00 00 00 00
    00 00 00 10 00 00 00 00 00 00
    00 00 00 00 00 10 00 00 00 00
    00 00 00 10 00 00 00 00 00 00
    00 00 00 00 10 00 00 00
```

RESULT : A square, 16 by 16, is drawn along the xz plane with the starting point being the current point.

ERRORS : Not enough memory, arithmetic overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, POLY3, PRMFIL, Section 3.6

COMMAND :

LONG FORM : PRMFIL flag

SHORT FORM : PF flag

HEX FORM : E9 flag

PARAMETER TYPE : flag = Char [0..1]

DESCRIPTION : PRMFIL sets the primitive fill flag to flag. When PRMFIL is set to 0, closed figures are drawn in outline only; when PRMFIL is set to 1, closed figures are filled with the current color in the current area pattern. PRMFIL affects the following commands: CIRCLE, ELIPSE, POLY, POLYR, POLY3, POLYR3, RECT, RECTR, SECTOR, SCIRC, SELIPS, SPOLY, SPOLYR, SRECT, SRECTR, and SSECT.

EXAMPLE :

CODE :

ASCII : PF 0

HEX : E9 00

RESULT : Closed figures are drawn in outline only.

ERRORS : None

RELATED MATERIALS : AREAPT, BCOLOR, COLOR, COLMOD, Section 3.7

PROJCT
(Projection)

COMMAND :

LONG FORM : PROJCT angle

SHORT FORM : PRO angle

HEX FORM : B0 angle

PARAMETER TYPE : angle = Int [0..179]

DESCRIPTION : PROJCT sets the viewing angle used in 3D to 2D transformations. When angle is 0°, an orthogonal projection is produced; otherwise, a perspective projection is produced. The default is 60°.

EXAMPLE :

CODE :

ASCII : PRO 0

HEX : B0 00 00

RESULT : Orthogonal projections are produced.

ERRORS : Value out of range, arithmetic overflow

RELATED MATERIALS : DISTAN, Subsection 3.4.2

COMMAND :

LONG FORM : RASTOP oper srcdir destdir x_0 x_1 y_0 y_1 x'_0 y'_0

SHORT FORM : ROP oper srcdir destdir x_0 x_1 y_0 y_1 x'_0 y'_0

HEX FORM : DA oper srcdir destdir x_0 x_1 y_0 y_1 x'_0 y'_0

PARAMETER TYPE : oper = Char [0..3]
srcdir = Char [0..7]
destdir = Char [0..7]
x_0 = Unsigned Int [0..639]
x_1 = Unsigned Int [0..639]
y_0 = Unsigned Int [0..479]
y_1 = Unsigned Int [0..479]
x'_0 = Unsigned Int [0..639]
y'_0 = Unsigned Int [0..479]

DESCRIPTION : RASTOP copies a rectangular area of the screen, with lower left corner $\{x_0, y_0\}$ and upper right corner $\{x_1, y_1\}$ (specified in pixels), to another area of the screen starting at lower left corner $\{x'_0, y'_0\}$. The corners are included in the region and both rectangles must be on the screen (including hidden space). All bit planes are copied (subject to normal masking as specified by the MASK command). If the rectangles overlap, the user must select appropriate major and minor directions to ensure that the area is copied properly. The raster operation function is selected according to the following table and performed on a pixel by pixel basis on the source and the destination regions.

Raster Operation Functions	
oper	Operation
0	copy
1	or ($\vee$)
2	and ($\wedge$)
3	xor ($\oplus$)

The direction of scanning of the source (input) region is specified by srcdir; the direction of scanning of the destination (output) region is specified by destdir. Both are selected using the following table:

Scanning Direction		
direction	Major Direction	Minor Direction
0	⇒	↑
1	⇒	↓
2	⇐	↑
3	⇐	↓
4	⇑	→
5	⇓	→
6	⇑	←
7	⇓	←

EXAMPLE :

CODE :

ASCII : ROP 0 0 0 320 639 240 479 0 0

HEX : DA 00 00 00 40 01 7F 02 F0 00 DF
01 00 00 00 00

RESULT : The upper right side of the screen is duplicated at the lower left.

ERRORS : Invalid operation, Invalid direction, Will not fit on screen

RELATED MATERIALS : Section 3.10

COMMAND :

LONG FORM : RASTRD dir x_0 x_1 y_0 y_1

SHORT FORM : RRD dir x_0 x_1 y_0 y_1

HEX FORM : DB dir x_0 x_1 y_0 y_1

PARAMETER TYPE : dir = Char [0..3]
x_0 = Unsigned Int [0..639]
x_1 = Unsigned Int [0..639]
y_0 = Unsigned Int [0..479]
y_1 = Unsigned Int [0..479]

DESCRIPTION : RASTRD copies a rectangular area of the screen, with corners $\{x_0, y_0\}$ and $\{x_1, y_1\}$ to the system memory of the system unit. This operation uses the DMA (Direct Memory Access) controller of the system unit. The corners of the area, specified in pixels, are included in the region and all bit planes are copied (subject to normal masking as specified by the MASK command).

This command will transfer $(x_1 - x_0 + 1) \times (y_1 - y_0 + 1)$ bytes. Until all data has been transferred, no commands will be interpreted by the board. To abort an incomplete RASTRD, issue a cold reset by writing a 1 to the Cold Reset Flag.

The direction of scanning the region is specified according to the following table:

Scanning Direction		
direction	Major Direction	Minor Direction
0	⇒	↑
1	⇒	↓
2	⇐	↑
3	⇐	↓

Note:

As this command uses the DMA Controller (8-bit channel 1, 2, or 3) of the PC XT/AT (programmed by the user), transfers are limited to 64 Kbytes.

EXAMPLE :

CODE :

ASCII : **RRD 0 0 639 0 479**

HEX : **DB 00 00 00 7F 02 00 00 DF 01**

RESULT : **Entire screen is read.**

ERRORS : **Value out of range**

RELATED MATERIALS : **RASTWR, Section 3.10**

COMMAND :

LONG FORM : RASTWR oper dir x_0 x_1 y_0 y_1

SHORT FORM : RWR oper dir x_0 x_1 y_0 y_1

HEX FORM : DC oper dir x_0 x_1 y_0 y_1

PARAMETER TYPE : oper = Char [0..3]
dir = Char [0..3]
x_0 = Unsigned Int [0..639]
x_1 = Unsigned Int [0..639]
y_0 = Unsigned Int [0..479]
y_1 = Unsigned Int [0..479]

DESCRIPTION : RASTWR copies a rectangular area of the screen, with corners $\{x_0.y_0\}$ and $\{x_1.y_1\}$ from the system memory of the system unit. This uses the DMA (Direct Memory Access) controller of the system unit. The corners of the area, specified in pixels, are included in the region. All bit planes are copied (subject to normal masking as specified by the MASK command).

The pixel combination operation performed (between old and new pixels) is specified using the following table. Operation 0 will not use the old pixels, but will directly copy new pixel data into the screen memory.

Raster Write Function	
oper	Operation
0	copy
1	or ($\vee$)
2	and ($\wedge$)
3	xor ($\oplus$)

This command will transfer $(x_1 - x_0 + 1) \times (y_1 - y_0 + 1)$ bytes. Until this data is transferred, no commands will be interpreted by the HLGE. To abort an incomplete RASTWR, issue a cold reset.

The direction of scanning the region is specified according to the following table:

Scanning Direction		
dir	Major Direction	Minor Direction
0	$\Rightarrow$	$\uparrow$
1*	$\Rightarrow$	$\downarrow$
2*	$\Leftarrow$	$\uparrow$
3*	$\Leftarrow$	$\downarrow$

* Applicable only for oper = 0

Note:

As this command uses the DMA Controller (8-bit channel 1, 2, or 3) of the PC XT/AT (programmed by the user), transfers are limited to 64 Kbytes.

EXAMPLE :

CODE :

ASCII : RWR 0 0 0 639 0 479

HEX : DC 00 00 00 00 7F 02 00 00 DF 01

RESULT : A 640 by 480 pixel section of the screen is written to from the bus memory.

ERRORS : Value out of range

RELATED MATERIALS : RASTRD, Section 3.10

RBAND
(Rubber Band Cross Hair)

COMMAND :

LONG FORM : **RBAND flag**

SHORT FORM : **RB flag**

HEX FORM : **E1 flag**

PARAMETER TYPE : **flag = Char [0..2]**

DESCRIPTION : **RBAND enables the rubber band vector (flag = 1), the rubber band rectangle (flag = 2), or disables both (flag = 0).**

The cross hair coordinates, at the time when either the rubber band vector or the rubber band rectangle is enabled, becomes the anchor point. When a new set of cross hair coordinates is entered, a vector or a rectangle is drawn from the anchor to the new coordinates in complement mode. As the coordinates are changed the vector or rectangle is erased and redrawn from the anchor to the new cross hair coordinates. When the rubber band is disabled, the vector or rectangle last drawn is erased and the cross hair coordinate is left at the last coordinate pair entered.

When first enabled, the anchor and the cross hair coordinate will be on the same point and the rubber band vector or rectangle will be drawn as a point.

EXAMPLE :

CODE :

ASCII : **RB 2**

HEX : **E1 02**

RESULT : **The rubber band rectangle is enabled.**

ERRORS : **Value out of range**

RELATED MATERIALS : **XHAIR, XMOVE, Section 3.13**

COMMAND :

LONG FORM : RDEFIN font height width size start_char array

SHORT FORM : RDF font height width size start_char array

HEX FORM : 54 font height width size start_char array

PARAMETER TYPE : font = Char [1..15]
height = Char [0..16]
width = Char [0..16]
size = Char
start_char = Char
array = array of Char

DESCRIPTION : The user definable raster fonts 1 to 15 are defined using the RDEFIN command. Each character in the font must have the same cell size, subject to the height and width parameters. The number of characters in the font, minus one, is specified by size and the ASCII code of the first character in the font is specified by start_char. In HEX mode, each row of a character cell is represented by a left justified packed string of bits, each bit representing one pixel.

EXAMPLE :

CODE :

ASCII : RDEFIN 1 7 5 1 65 0 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0

HEX : 54 01 07 05 01 41 70 88 88 F8 88 88 88 F0 88 88 F0 88 88 F0

RESULT : Font 1 is defined with two characters: A and B.

ERRORS : parameter range

RELATED MATERIALS : RFONT, TEXTP, TEXTPC, Subsection 3.8.2

COMMAND FORMAT:

LONG FORM : RFONT font h_aspect w_aspect

SHORT FORM : RFT font h_aspect w_aspect

HEX FORM : 55 font h_aspect w_aspect

PARAMETER TYPE : font = Char [0..15]
h_aspect = Char [0..1]
w_aspect = Char [0..1]

DESCRIPTION : The RFONT command selects the font that will be used to draw user definable raster characters on the screen, using the TEXTP and TEXTPC commands. The font must have been previously defined using either the RDEFIN or TDEFIN commands.

The w_aspect and h_aspect parameters specify the aspect ratio of the characters. A value of 0 indicates single height/width and a value of 1 indicates double height/width.

EXAMPLE :

CODE :

ASCII : RFONT 1 1 0

HEX : 55 01 01 00

RESULT : Font 1 will be selected when using the TEXTP and TEXTPC commands, in double height, and single width aspect ratio.

ERRORS : parameter range

RELATED MATERIALS : RFONT, TEXTP, TEXTPC, Subsection 3.8.2

RECT
(Rectangle)

COMMAND :

LONG FORM : RECT x y

SHORT FORM : R x y

HEX FORM : 34 x y

PARAMETER TYPE : x = Real
y = Real

DESCRIPTION : RECT draws a rectangle with one corner on the 2D current point and the diagonally opposite corner on {x, y}. When the PRMFIL flag is set, the rectangle will be drawn filled; if PRMFIL is not set, drawing will be subject to LINPAT. The 2D current point remains unchanged.

EXAMPLE :

CODE :

ASCII : R 128 64

HEX : 34 80 00 00 00 40 00 00 00

RESULT : A rectangle is drawn with one corner on the 2D current point and the other on {128,64}.

ERRORS : None

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, PRMFIL, RECT
Section 3.6

RECTR
(Rectangle Relative)

COMMAND :

LONG FORM : RECTR Δx Δy

SHORT FORM : RR Δx Δy

HEX FORM : 35 Δx Δy

PARAMETER TYPE : Δx = Real
Δy = Real

DESCRIPTION : RECTR draws a rectangle with one corner on the 2D current point and the diagonally opposite corner displaced from the 2D current point by $\{\Delta x, \Delta y\}$. When the PRMFIL flag is set, the rectangle will be drawn filled; if PRMFIL is not set, drawing will be subject to LINPAT. The 2D current point remains unchanged.

EXAMPLE :

CODE :

ASCII : RR 128 64

HEX : 35 80 00 00 00 40 00 00 00

RESULT : A rectangle is drawn with one corner on the 2D current point and the diagonally opposed corner displaced by {128,64}.

ERRORS : Arithmetic overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, PRMFIL, RECT, Section 3.6

RESETF
(Reset Flags)

COMMAND :

LONG FORM : RESETF

SHORT FORM : RF

HEX FORM : 04

PARAMETER TYPE : None

DESCRIPTION : RESETF resets all flags and parameters to their default values, as specified in the table on the following page. This is done automatically when the board is reset or the power turned on.

EXAMPLE :

CODE :

ASCII : RF
HEX : 04

RESULT : All flags are reset

ERRORS : None

RELATED MATERIALS : FLAGRD

RESETF
(Reset Flags)

Flag	Name	Default Value	Description
1	AREAPT	65535 16 times	solid area
2	CLIPH	0	disabled
3	CLIPY	0	disabled
4	COLOR	255	
5	DISPLA	no change	
6	DISTAN	500	
7	DISTH	-30000	
8	DISTY	30000	
9	FILMSK	255	all planes used
10	LINFUN	0	set mode
11	LINPAT	65535	solid lines
12	MASK	255	all planes on
13	MDORG	(0, 0, 0)	
14	2-D current point	(0, 0)	
15	3-D current point	(0, 0)	
16	PRMFIL	0	off
17	PROJCT	60	
18	TANGLE	0	horizontal
19	TJUST	1,1	left, bottom
20	TSIZE	8	8 by 12 cells
21	VWPORT	0,639,0,479	entire screen
22	VWRPT	(0, 0, 0)	
23	WINDOW	-320,319,-240,239	
24	transformed 3D point	(0, 0, 0)	
25	none	none	used in FLAGRD
26	current positon of XHAIR	320,240	
27	2-D position of XHAIR	0,0	
28	screen current point	320,240	
29	none	none	used in FLAGRD
32	TSTYLE	0	'fat' text
33	TASPCT	1.5	
34	TCHROT	0	

COMMAND :

LONG FORM : SARC radius angle1 angle2

SHORT FORM : SAR radius angle1 angle2

HEX FORM : F4 radius angle1 angle2

PARAMETER TYPE : radius = Int
angle1 = Int
angle2 = Int

DESCRIPTION : SARC draws a circular arc using the currently selected color. The center is on the 2D current point. The radius, and start and finish angles are specified in the command. The angles can be any Int value (angles greater than 360° and less than -360° are handled as modulo 360). Negative radii will result in 180° being added to both angles. This command does not affect the 2D current point.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480 (See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SAR 100 0 180

HEX : F4 64 00 00 00 B4 00

RESULT : An arc with radius 100 from 0° to 180° (a semi-circle) is drawn about the 2D current point.

ERRORS : Overflow

RELATED MATERIALS : SCIRC, COLOR, LINFUN, LINPAT, Section 3.10

COMMAND :

LONG FORM : SBLINK␣

SHORT FORM : SBL␣

HEX FORM : E4

PARAMETER TYPE : None

DESCRIPTION : SBLINK sets all LUT entries currently assigned as blinking, by either the BLINK or the BLINKX commands, as static. If you only want to cancel blinking of one LUT entry you can still use the BLINK and BLINKX commands. SBLINK is useful when you want to stop all blinking on the screen with one instruction.

All blinking colors are restored to their original color.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480 (See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SBL␣

HEX : E4

RESULT : All blinking pixels, if any, will stop blinking.

ERRORS : None

RELATED MATERIALS : BLINK, BLINKX, Subsection 3.5.3

COMMAND :

LONG FORM : SCIRC radius

SHORT FORM : SCI radius

HEX FORM : F2 radius

PARAMETER TYPE : radius = Int

DESCRIPTION : SCIRC draws a circle with radius radius centered on the 2D current point. The circle is filled if the PRMFIL flag is set. This command does not affect the 2D current point.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480 (See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SCI 100

HEX : F2 64 00

RESULT : A circle with radius 100 is drawn from the 2D current point.

ERRORS : Overflow

RELATED MATERIALS : SARC, SELIPS, LINFUN, LINPAT, PRMFIL, SSECT, Section 3.10

COMMAND :

LONG FORM : SDRAW x y

SHORT FORM : SD x y

HEX FORM : FA x y

PARAMETER TYPE : x = Int
y = Int

DESCRIPTION : SDRAW draws a line from the 2D current point to {x,y} and positions the 2D current point to {x,y}. This command does not draw the last pixel of a line.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480 (See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SD 10 12

HEX : FA 0A 00 0C 00

RESULT : A line is drawn from the 2D current point to {10,12}.

ERRORS : Arithmetic overflow

RELATED MATERIALS : SDRAWR, LINFUN, LINPAT, SMOVE, SMOVER, Section 3.10

COMMAND :

LONG FORM : SDRAWR Δx Δy

SHORT FORM : SDR Δx Δy

HEX FORM : FB Δx Δy

PARAMETER TYPE : Δx = Int
Δy = Int

DESCRIPTION : SDRAWR draws a line from the 2D current point to ({Δx,Δy} + current point). The 2D current point is moved to the end of the line. This command does not draw the last pixel of a line.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480 (See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SDR 100 200

HEX : FB 64 00 C8 00

RESULT : A line is drawn from the 2D current point to (the current point + {100,200}).

ERRORS : Arithmetic overflow

RELATED MATERIALS : SDRAW, LINFUN, LINPAT, SMOVE, SMOVE
Section 3.10

COMMAND :

LONG FORM : SECTOR radius angle1 angle2

SHORT FORM : S radius angle1 angle2

HEX FORM : 3D radius angle1 angle2

PARAMETER TYPE : radius = Real
angle1 = Int
angle2 = Int

DESCRIPTION : SECTOR draws a pie shaped figure with the center on the current point, radius radius, and angles angle1 and angle2. If PRMFIL is set then the sector will be filled, otherwise drawing will be subject to LINPAT. If radius is negative then 180° will be added to both angles. The angles are integers and are treated as modulo 360. This command does not affect the current point.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480 (See Section 3.10.).

EXAMPLE :

CODE :

ASCII : S 50.25 45 135

HEX : 3D 32 00 00 40 2D 00 87 00

RESULT : A pie shaped sector is drawn with radius 50.25, starting at 45° and ending at 135°.

ERRORS : Arithmetic overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, PRMFIL, Section 3.6

COMMAND :

LONG FORM : **SELIPS xradius yradius**

SHORT FORM : **SEL xradius yradius**

HEX FORM : **F3 xradius yradius**

PARAMETER TYPE : **xradius = Int**
yradius = Int

DESCRIPTION : **SELIPS draws a 2D ellipse centered on the 2D current point and whose x and y radii are given by xradius and yradius. The ellipse will be filled if drawn while the PRMFIL flag is set. This command does not affect the 2D current point.**

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480 (See Section 3.10.).

EXAMPLE :

CODE :

ASCII : **SEL 32 128**

HEX : **F3 20 00 80 00**

RESULT : **An ellipse is drawn with x radius 32 and y radius 128.**

ERRORS : **Overflow**

RELATED MATERIALS : **AREAPT, LINFUN, LINPAT, PRMFIL, Section 3.10**

COMMAND :

LONG FORM : SMOVE x y

SHORT FORM : SM x y

HEX FORM : F8 x y

PARAMETER TYPE : x = Int
y = Int

DESCRIPTION : SMOVE moves the 2D current point to {x,y}.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480(See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SM 10 12

HEX : F8 0A 00 0C 00

RESULT : The 2D current point is moved to {10,12}.

ERRORS : Arithmetic overflow

RELATED MATERIALS : SMOVER, Section 3.10

SMOVER
(Screen Move Relative)

COMMAND :

LONG FORM : SMOVER Δx Δy

SHORT FORM : SMR Δx Δy

HEX FORM : F9 Δx Δy

PARAMETER TYPE : Δx = Int
Δy = Int

DESCRIPTION : SMOVER moves the 2D current point to ($\{\Delta x, \Delta y\}$ + the current point).

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480(See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SMR 10 12

HEX : F9 0A 00 0C 00

RESULT : The current point is moved to ({10,12} + the current point).

ERRORS : Arithmetic overflow

RELATED MATERIALS : SMOVE, Section 3.10

COMMAND :

LONG FORM : SPOLY n x_1 y_1 x_2 y_2 $\cdots$ x_n y_n

SHORT FORM : SP n x_1 y_1 x_2 y_2 $\cdots$ x_n y_n

HEX FORM : FC n x_1 y_1 x_2 y_2 $\cdots$ x_n y_n

PARAMETER TYPE : n = Char
x_i = Int
y_i = Int

DESCRIPTION : SPOLY draws a closed polygon directly on the screen. Parameter n is the number of vertices and $\{x_i, y_i\}$ the coordinates of the vertices. The polygon will be filled if the PRMFIL flag is set and subject to the LINPAT if PRMFIL is not set. The 2D current point will not be changed.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480(See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SP 4 0 0 16 0 16 16 0 16

HEX : FC 04 00 00 00 00 10 00 00 00 10
00 10 00 00 00 10 00

RESULT : A square, 16 by 16, is drawn.

ERRORS : Not enough memory, arithmetic overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, SPOLYR, PRMFIL, Section 3.10

SPOLYR
(Polygon Relative)

COMMAND :

LONG FORM : SPOLYR n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n

SHORT FORM : SPR n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n

HEX FORM : FD n Δx_1 Δy_1 Δx_2 Δy_2 $\cdots$ Δx_n Δy_n

PARAMETER TYPE : n = Char
Δx_i = Int
Δy_i = Int

DESCRIPTION : SPOLYR draws a closed polygon directly to the screen. Parameter n is the number of vertices and $\{\Delta x_i, \Delta y_i\}$ the displacements of the vertices from the 2D current point. The polygon will be filled if the PRMFIL flag is set and subject to the LINPAT if PRMFIL is not set. The 2D current point will not be changed.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480(See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SPR 4 0 0 16 0 16 16 0 16

HEX : FD 04 00 00 00 00 10 00 00 00 10
00 10 00 00 00 10 00

RESULT : A square, 16 by 16, is drawn with the upper left corner on the 2D current point.

ERRORS : Not enough memory, arithmetic overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, SPOLY, PRMFIL, Section 3.10

COMMAND :

LONG FORM : SRECT x y

SHORT FORM : SR x y

HEX FORM : F0 x y

PARAMETER TYPE : x = Int [0..639]
y = Int [0..479]

DESCRIPTION : SRECT draws a rectangle with one corner on the 2D current point and the diagonally opposite corner on {x,y}. When the PRMFIL flag is set, the rectangle will be drawn filled; if PRMFIL is not set, then drawing will be subject to LINPAT. The 2D current point remains unchanged.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480(See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SR 128 64

HEX : F0 80 00 40 00

RESULT : A rectangle is drawn with one corner on the 2D current point and the other on {128,64}.

ERRORS : None

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, PRMFIL, SRECTR, Section 3.10

SRECTR
(Screen Rectangle Relative)

COMMAND :

LONG FORM : SRECTR Δx Δy

SHORT FORM : SRR Δx Δy

HEX FORM : F1 Δx Δy

PARAMETER TYPE : Δx = Int
Δy = Int

DESCRIPTION : SRECTR draws a rectangle with one corner on the 2D current point and the diagonally opposite corner displaced from the 2D current point by {Δx, Δy}. When the PRMFIL flag is set, the rectangle will be drawn filled. If PRMFIL is not set, then the drawing will be subject to LINPAT. The 2D current point remains unchanged.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480(See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SRR 128 64

HEX : F1 80 00 40 00

RESULT : A rectangle is drawn with one corner on the 2D current point and the other displaced by {128,64}.

ERRORS : Arithmetic overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, PRMFIL, SREC
Section 3.10

COMMAND :

LONG FORM : SSECT radius angle1 angle2

SHORT FORM : SS radius angle1 angle2

HEX FORM : F5 radius angle1 angle2

PARAMETER TYPE : radius = Int
angle1 = Int
angle2 = Int

DESCRIPTION : SSECT draws a pie shaped figure with center on the 2D current point, radius radius, and angles angle1 and angle2. If PRMFIL is set, the sector will be filled; otherwise, drawing will be subject to LINPAT. If radius is negative then 180° will be added to both angles. The angles are integers and are treated as modulo 360. This command does not affect the 2D current point.

Note: The viewport and the window must have exactly the same coordinates for this command to function correctly, and the viewport must be equal to the maximum screen resolution i.e. 640 by 480(See Section 3.10.).

EXAMPLE :

CODE :

ASCII : SS 50 45 135

HEX : F5 32 00 2D 00 87 00

RESULT : A pie shaped sector is drawn having radius 50, starting at 45° and going through to 135°.

ERRORS : Arithmetic overflow

RELATED MATERIALS : AREAPT, LINFUN, LINPAT, PRMFIL, Section 3.10

COMMAND :

LONG FORM : TANGLE angle

SHORT FORM : TA angle

HEX FORM : 82 angle

PARAMETER TYPE : angle = Int

DESCRIPTION : TANGLE sets the rotation angle for text; specifically the angle of the baseline (the imaginary line that characters are drawn on). The angle is specified by angle. The default is the normal left to right drawing angle 0°. TANGLE does not affect the rotation of the individual characters; character rotation is specified using TCHROT.

EXAMPLE :

CODE :

ASCII : TA 270

HEX : 82 0E 01

RESULT : Characters are drawn vertically top to bottom.

ERRORS : None

RELATED MATERIALS : TCHROT, TEXT, TEXTP, Section 3.8

COMMAND :

LONG FORM : TASPCT ratio

SHORT FORM : TASP ratio

HEX FORM : 8B ratio

PARAMETER TYPE : ratio = Real

DESCRIPTION : TASPCT sets the text aspect ratio for style 1 characters (see TSTYLE). The aspect ratio is the ratio of character height to width, the default is 1.5 (when TSIZE = 8, this represents a character 12 pixels high by 8 pixels wide). Parameter ratio must be greater than zero.

EXAMPLE :

CODE :

ASCII : TASP 2

HEX : 8B 02 00 00 00

RESULT : Characters are drawn twice as high as they are wide.

ERRORS : Value out of range

RELATED MATERIALS : TEXT, TEXTP, TSIZE, TSTYLE, Section 3.8

TCHROT
(Text Character Rotation)

COMMAND :

LONG FORM : TCHROT angle

SHORT FORM : TCR angle

HEX FORM : 8A angle

PARAMETER TYPE : angle = Int

DESCRIPTION : TCHROT sets the angle of rotation for characters. Only text of style 1 will be rotated, style 0 will be unaffected. The rotation is independent of the baseline rotation set by TANGLE. Text styles are selected using TSTYLE.

EXAMPLE :

CODE :

ASCII : TCR 90

HEX : 8A 5A 00

RESULT : Characters are rotated by 90°.

ERRORS : None

RELATED MATERIALS : TANGLE, TEXT, TEXTP, TSTYLE, Section 3.8

COMMAND :

LONG FORM : TDEFIN n x y array

SHORT FORM : TD n x y array

HEX FORM : 84 n x y array

PARAMETER TYPE : n
= Char
x = Char
y = Char
array = x columns by y rows of Chars (ASCII mode) or x bits packed left justified in y byte sets (Hex mode)

DESCRIPTION : TDEFIN defines the character given by n to be an array with character cell size x by y and contents array. In ASCII mode, each pixel in the character cell is represented by either the character "0" or the character "1". Where a pixel is set to "0", the character will be transparent, or the current background color (BCOLOR), depending on the current state of COLMOD. Where the pixel is set to "1", the pixel will be the color index last specified by the COLOR command. In Hex mode, each row of the character cell is represented by a packed string of bits, each bit representing one pixel. These bits are left justified so that the first bit is in the highest bit position.

NOTE : If you specify a value of 0 for either the x or the y parameter you will delete the character definition.

EXAMPLE :

CODE :

ASCII :
```
TD 65 5 7  0 1 1 1 0
           1 0 0 0 1
           1 0 0 0 1
           1 1 1 1 1
           1 0 0 0 1
           1 0 0 0 1
           1 0 0 0 1
```

HEX : 84 41 05 07 70 88 88 F8 88 88 88

RESULT : The letter "A" is defined.

ERRORS : Not enough memory

RELATED MATERIALS : TEXTP, COLMOD, Section 3.8

COMMAND :

LONG FORM : TEXT 'string' or "string"

SHORT FORM : T 'string' or "string"

HEX FORM : 80 'string' or "string"

PARAMETER TYPE : string = any number of Chars up to 640

DESCRIPTION : TEXT writes a text string to the screen, justified about the current point as specified in the last TJUST command. The string may be delimited by either double or single quotes. If no quotes are used the string will be terminated by the first delimiter encountered. The text will be in the size and style specified by the last TSIZE and TSTYLE commands. When TSTYLE has been set to 0, fat text will be produced; when TSTYLE has been set to 1, thin rotatable text will be produced. If COLMOD = Replace, the character cell will be drawn according to the current LINFUN and BCOLOR parameters.

Note: The fastest character drawing speed is attained when fat text of size 16 (size 8 if in PG-640 mode) is selected, with the left side of the beginning of the string located on 16-pixel multiples (0, 16, 32, ...) along the x-axis.

EXAMPLE :

CODE :

ASCII : T 'Hello'

HEX : 80 22 48 65 6C 6C 6F 22

RESULT : Hello is printed on the screen.

ERRORS : String too long, Arithmetic overflow

RELATED MATERIALS : TANGLE, TASPCT, TCHROT, TEXTP, TJUST, TSIZE, TSTYLE, Section 3.8

TEXTC
(Fixed Length Text)

COMMAND :

LONG FORM : None

SHORT FORM : None

HEX FORM : 8C count char char ... char

PARAMETER TYPE : count = Unsigned Int [0..640]
char = Char

DESCRIPTION : This command displays a text string of up to 640 characters. The count parameter specifies the number of characters in the string that follows it. Note that this command is restricted to Hex mode.

EXAMPLE :

CODE :

ASCII : None

HEX : 8C 05 00 41 42 43 44 45

RESULT : The text string "ABCDE" is displayed at the current point.

ERRORS : Range

RELATED MATERIALS : TEXT, TANGLE, TSIZE, Section 3.8

COMMAND :

LONG FORM : TEXTP 'string' or "string"

SHORT FORM : TP 'string' or "string"

HEX FORM : 83 'string' or "string"

PARAMETER TYPE : string = any number of Chars up to 640

DESCRIPTION : TEXTP writes a text string to the screen using programmable fonts. The text will be justified about the current point as specified in the last TJUST command, and be in the style specified in the last TSTYLE command. When TSTYLE is set to zero, the text font defined by TDEFIN is used; when TSTYLE is set to one, the text defined by GTDEF is used. The string may be delimited by either double or single quotes. If no quotes are used, the string will be terminated by the first delimiter encountered.

EXAMPLE :

CODE :

ASCII : TP 'Hello'

HEX : 83 22 48 65 6C 6C 6F 22

RESULT : Hello is printed on the screen.

ERRORS : String too long, Arithmetic overflow

RELATED MATERIALS : TASPCT, TANGLE, TCHROT, TDEFIN, TEXT, TJUST, TSIZE, TSTYLE, Section 3.8

TEXTPC
(Fixed Length Programmable Text)

COMMAND :

LONG FORM : None

SHORT FORM : None

HEX FORM : 8D count char ... char

PARAMETER TYPE : count = Unsigned Int [0..640]
char = Char

DESCRIPTION : This command displays a programmable text string at the current point. The count parameter specifies the number of characters in the string that follows. This command is identical to the TEXTC command. Note that this command is restricted to Hex mode.

EXAMPLE :

CODE :

ASCII : None

HEX : 8D 05 00 41 42 43 44 45

RESULT : The programmable text string "ABCDE" is displayed at the current point.

ERRORS : Range

RELATED MATERIALS : TEXTP, TANGLE, TSTYLE, TDEFIN, GTDEF, Section 3.8

COMMAND :

LONG FORM : TJUST horiz vert

SHORT FORM : TJ horiz vert

HEX FORM : 85 horiz vert

PARAMETER TYPE : horiz = Char [1..3]
vert = Char [1..3]

DESCRIPTION : TJUST sets the horizontal and vertical justification as specified in the table below. The default values are: horiz = 1 and vert = 1.

TEXT JUSTIFICATION	
VALUE	ACTION
1	Justify on left or bottom
2	Center
3	Justify on top or right

EXAMPLE :

CODE :

ASCII : TJ 2 1

HEX : 85 02 01

RESULT : Output text is centered horizontally about the current point with its bottom on the current point.

ERRORS : Range error

RELATED MATERIALS : TEXT, TEXTP, Section 3.8

TSIZE
(Text Size)

COMMAND :

LONG FORM : TSIZE size

SHORT FORM : TS size

HEX FORM : 81 size

PARAMETER TYPE : size = Real

DESCRIPTION : TSIZE sets the text size by specifying the virtual distance from one character to the next. The default value is 8. TSIZE directly sets the width of each character and the height is set using TASPCT (*height* = *width* × *aspect ratio*). The size of fat text will be rounded off to a multiple of eight pixels.

EXAMPLE :

CODE :

ASCII : TS 16

HEX : 81 10 00 00 00

RESULT : Text size is doubled from default.

ERRORS : Arithmetic overflow

RELATED MATERIALS : TASPCT, TEXT, TEXTP, TSTYLE, Section 3.8

COMMAND :

LONG FORM : TSTYLE flag

SHORT FORM : TSTY flag

HEX FORM : 88 flag

PARAMETER TYPE : flag = Char [0..1]

DESCRIPTION : TSTYLE sets the style of the text drawn with TEXT or TEXTP commands. When flag is 0, characters will be fat – that is to say the lines forming the characters will become wider as their size is increased by a TSIZE command. When flag is 1, the characters will always be constructed with lines one pixel wide. The default is style 0. The effect of this command is only noticeable when characters are drawn in sizes larger than normal.

EXAMPLE :

CODE :

ASCII : TSTY 1

HEX : 88 01

RESULT : Thin rotatable text is selected.

ERRORS : None

RELATED MATERIALS : TEXT, TEXTP, TSIZE, Section 3.8

TWCOL
(Text Window Color – 8 Bit)

COMMAND :

LONG FORM : TWCOL r g b

SHORT FORM : TWC r g b

HEX FORM : D5 r g b

PARAMETER TYPE : r = Char [0..255]
g = Char [0..255]
b = Char [0..255]

DESCRIPTION : This command sets the foreground color used in text windows. All text windows have a transparent background.

EXAMPLE :

CODE :

ASCII : TWCOL 2 4 8

HEX : D5 02 04 08

RESULT : The foreground color for text windows is changed to r = 2, g = 4, and b = 8.

ERRORS : None

RELATED MATERIALS : TWPOS, TWVIS, Subsection 3.11

COMMAND :

LONG FORM : TWPOS x_0 x_1 y_0 y_1 e_0 e_1

SHORT FORM : TWP x_0 x_1 y_0 y_1 e_0 e_1

HEX FORM : D3 x_0 x_1 y_0 y_1 e_0 e_1

PARAMETER TYPE : x_0 = Unsigned Int [0..639]
x_1 = Unsigned Int [0..639]
y_0 = Unsigned Int [0..479]
y_1 = Unsigned Int [0..479]
e_0 = Unsigned Int [0..79]
e_1 = Unsigned Int [0..24]

DESCRIPTION : TWPOS sets the size and position of the emulator window on the graphics screen. A rectangular region of the emulator screen (in its current mode) with upper left corner $\{e_0,e_1\}$ is mapped onto the high resolution graphics screen from $\{x_0,x_1\}$ to $\{y_0,y_1\}$. All parameters are specified in pixels. The parameters e_0 and e_1 are specified in character cells, based on the 80 by 25 text mode of the CGA Emulator.

TWPOS does not make the text window visible (see TWVIS) but when issuing a TWPOS command while the text window is visible, the text window will appear in its new location immediately.

Restrictions:

- The TWPOS command only works for the 80×25 and 40×25 alphanumeric CGA video modes. To see the full CGA screen in any mode use the DISPLA command.
- The screen positions must be on 16 pixel boundaries; i.e. x_0 and $x_1 + 1$ must be divisible by 16.
- Displaying the emulator window slows down the high resolution drawing rate proportional to the position of the right hand edge of the emulator window. To speed up graphics drawing make x_1 as small as possible.

EXAMPLE :

CODE :

ASCII : TWP 0 639 416 479 0 0

HEX : D3 00 00 7F 02 A0 01 DF 01 00 00 00 00

RESULT : The top four lines of text from the emulator screen are mapped on to the bottom of the graphics screen.

ERRORS : Bad text window position

RELATED MATERIALS : TWVIS, TWCOL, Section 3.11

COMMAND :

LONG FORM : **TWVIS flag**

SHORT FORM : **TWV flag**

HEX FORM : **D4 flag**

PARAMETER TYPE : **flag = Char [0..1]**

DESCRIPTION : **TWVIS enables (flag set to 0) the text window depending on flag. When the text window is enabled, the portion of the emulator screen specified by the last TWPOS command is displayed. The emulator must be enabled.**

Note: Graphics drawing is much faster when the text window is disabled.

EXAMPLE :

CODE :

ASCII : **TWV 1**

HEX : **D4 01**

RESULT : **Emulator screen is made visible.**

ERRORS : **No valid dialogue position specified**

RELATED MATERIALS : **TWPOS, TWCOL, Section 3.11**

VWIDEN
(Viewing Identity)

COMMAND :

LONG FORM : VWIDEN

SHORT FORM : VWI

HEX FORM : A0

PARAMETER TYPE : None

DESCRIPTION : VWIDEN sets the viewing transformation matrix to the identity matrix.

EXAMPLE :

CODE :

ASCII : VWI

HEX : A0

RESULT : Viewing matrix is set to the identity matrix.

ERRORS : None

RELATED MATERIALS : Subsection 3.4.2

COMMAND :

LONG FORM : **VWMATX array**

SHORT FORM : **VWM array**

HEX FORM : **A7 array**

PARAMETER TYPE : **array = 16 Reals**

DESCRIPTION : **VWMATX loads the viewing matrix with the data in array.**

EXAMPLE :

CODE :

ASCII : **VWM**

```
36.25 12.00 128 2
0 36.75 100 0
72.5 0 2.5 0
100.25 0 0 0
```

HEX : **A7**

```
24 00 00 40 0C 00 00 00 80 00 00 00 02
00 00 00 00 00 00 00 24 00 00 C0 64 00
00 00 00 00 00 00 52 00 00 80 00 00 00
00 02 00 00 80 00 00 00 00 64 00 00 40
00 00 00 00 00 00 00 00 00 00 00 00
```

RESULT : **The viewing matrix is set to the above data.**

ERRORS : **Arithmetic overflow**

RELATED MATERIALS : **Subsection 3.4.2**

VWPORT
(Viewport)

COMMAND :

LONG FORM : VWPORT x_1 x_2 y_1 y_2

SHORT FORM : VWP x_1 x_2 y_1 y_2

HEX FORM : B2 x_1 x_2 y_1 y_2

PARAMETER TYPE : x_1 = Unsigned Int [0..639]
x_2 = Unsigned Int [0..639]
y_1 = Unsigned Int [0..479]
y_2 = Unsigned Int [0..479]

DESCRIPTION : VWPORT defines a viewport on the screen where drawing can take place. The viewport is measured in pixels from the bottom left corner. Clipping is always enabled and the default viewport is the entire screen ({0,0} and {639,479}). Parameter x_1 must be less than x_2, and y_1 less than y_2, or else a warning will be generated. The pair that generated the warning will be swapped. A warning is also produced when any coordinate falls outside of the current screen boundary.

EXAMPLE :

CODE :

ASCII : VWP 0 300 0 100

HEX : B2 00 00 2C 01 00 00 64 00

RESULT : Viewport is defined to be from the lower left corner of the screen to {300,100}.

ERRORS : Arithmetic overflow

RELATED MATERIALS : WINDOW, Subsection 3.4.1

COMMAND :

LONG FORM : **VWROTX angle**

SHORT FORM : **VWX angle**

HEX FORM : **A3 angle**

PARAMETER TYPE : **angle = Int**

DESCRIPTION : **VWROTX rotates the x component of the viewing matrix by angle.**

EXAMPLE :

CODE :

ASCII : **VWX 45**

HEX : **A3 1D 00**

RESULT : **The x component is rotated by 45°.**

ERRORS : **Arithmetic overflow**

RELATED MATERIALS : **VWMATX, VWROTY, VWROTZ, Subsection 3.4.2**

VWROTY
(Viewing Rotate Y Axis)

COMMAND :

LONG FORM : VWROTY angle

SHORT FORM : VWY angle

HEX FORM : A4 angle

PARAMETER TYPE : angle = Int

DESCRIPTION : VWROTY rotates the y component of the viewing matrix by angle.

EXAMPLE :

CODE :

ASCII : VWY 45

HEX : A4 1D 00

RESULT : The y component is rotated by 45°.

ERRORS : Arithmetic overflow

RELATED MATERIALS : VWMATX, VWROTX, VWROTZ, Subsection 3.4.2

VWROTZ
(Viewing Rotate Z Axis)

COMMAND :

LONG FORM : VWROTZ angle

SHORT FORM : VWZ angle

HEX FORM : A5 angle

PARAMETER TYPE : angle = Int

DESCRIPTION : VWROTZ rotates the z component of the viewing matrix by angle.

EXAMPLE :

CODE :

ASCII : VWZ 45

HEX : A5 1D 00

RESULT : The z component is rotated by 45°.

ERRORS : Arithmetic overflow

RELATED MATERIALS : VWMATX, VWROTX, VWROTY, Subsection 3.4.2

VWRPT
(Viewing Reference Point)

COMMAND :

LONG FORM : VWRPT x y z

SHORT FORM : VWR x y z

HEX FORM : A1 x y z

PARAMETER TYPE : x = Real
y = Real
z = Real

DESCRIPTION : VWRPT sets the viewing reference point to be {x,y,z}. The viewing reference point is the point that the user is looking at.

EXAMPLE :

CODE :

ASCII : VWR 100 -25 50

HEX : A1 64 00 00 00 E7 FF 00 00 32 00 00 00

RESULT : Viewing reference point is defined to {100,-25,50}.

ERRORS : Arithmetic overflow

RELATED MATERIALS : Subsection 3.4.2

COMMAND :

LONG FORM : **WAIT frames**

SHORT FORM : **W frames**

HEX FORM : **05 frames**

PARAMETER TYPE : **frames = Unsigned Int**

DESCRIPTION : **WAIT produces a delay of frames frames. The value of frames is expressed in $\frac{1}{60}$ seconds (the maximum value of frames 65535 produces a delay of 18 minutes).**

EXAMPLE :

CODE :

ASCII : **W 60**

HEX : **05 3C 00**

RESULT : **A 1 second delay is produced.**

ERRORS : **None**

RELATED MATERIALS : **Subsection 3.3.4**

COMMAND :

LONG FORM : WINDOW x_1 x_2 y_1 y_2

SHORT FORM : WI x_1 x_2 y_1 y_2

HEX FORM : B3 x_1 x_2 y_1 y_2

PARAMETER TYPE : x_1 = Real
x_2 = Real
y_1 = Real
y_2 = Real

DESCRIPTION : WINDOW defines the coordinates of the corners of the window. The window is the section of the virtual workspace that is mapped to the screen's viewport area, which is set by the most recent VWPORT command.

EXAMPLE :

CODE :

ASCII : WI -25 50 75 100

HEX : B3 E7 FF 00 00 32 00 00 00 96 00
00 00 64 00 00 00

RESULT : The x and y coordinates are both defined to be from 0 to 64.

ERRORS : Arithmetic overflow, Range error

RELATED MATERIALS : VWPORT, Subsection 3.4.1

COMMAND :

LONG FORM : **XHAIR flag or flag x_size y_size**

SHORT FORM : **XH flag or flag x_size y_size**

HEX FORM : **E2 flag or flag x_size y_size**

PARAMETER TYPE : **flag = Char [0, 1, 3]**
x_size = Int [0..32767]
y_size = Int [0..32767]

DESCRIPTION : **XHAIR enables (flag = 1 or 3), or disables (flag = 0) the cross hair. When the cross hair is enabled, the two parameters x_size and y_size must be used in order to define the size of the cross hair. The cross hair will have a horizontal length of x_size coordinate units and a vertical length of y_size coordinate units. The cross hair is displayed in complement form with its center on the position specified by the last XMOVE command. Using flag equal to one will display the cross hair clipped by the screen size, flag equal to three produces a cross hair clipped by the current viewport. When the cross hair is disabled, the x_size and y_size parameters are not specified – the cross hair will no longer be displayed.**

EXAMPLE :

CODE :

ASCII : **XH 1 100 100**

HEX : **E2 01 64 00 64 00**

RESULT : **The cross-hair is enabled and defined to be 100 × 100.**

ERRORS : **Value out of range**

RELATED MATERIALS : **RBAND, VWPORT, XMOVE, Section 3.13**

XMOVE
(Cross Hair Move)

COMMAND :

LONG FORM : XMOVE x y

SHORT FORM : XM x y

HEX FORM : E3 x y

PARAMETER TYPE : x = Int [0..639]
y = Int [0..479]

DESCRIPTION : XMOVE changes the cross hair coordinates to {x,y}. The coordinates are specified in screen coordinates.

EXAMPLE :

CODE :

ASCII : XM 5 5

HEX : E3 05 00 05 00

RESULT : The cross hair coordinate is set to {5,5}.

ERRORS : Value out of range

RELATED MATERIALS : RBAND, XHAIR, Section 3.13

Chapter 5

The CGA Emulator

5.1 The Programmer's Model

The PG-640A's color graphics adaptor emulator creates the appearance of a IBM Color Graphics Adaptor in the system unit. The PG-640A emulates the registers of the graphics adaptor, as well as the functions of the 6845 CRT controller. The emulator has 16K $\times$ 8 bits of dedicated display memory. This memory is directly accessible by the system microprocessor and provides the basis for four video modes:

1. 40 $\times$ 25 Alphanumeric
2. 80 $\times$ 25 Alphanumeric
3. 320 $\times$ 200 $\times$ 2 Pixel Addressable Graphics
4. 640 $\times$ 200 $\times$ 1 Pixel Addressable Graphics

The graphics emulator allows the user to run existing software, such as 1-2-3 from LOTUS and Microsoft Flight Simulator. If there is a color

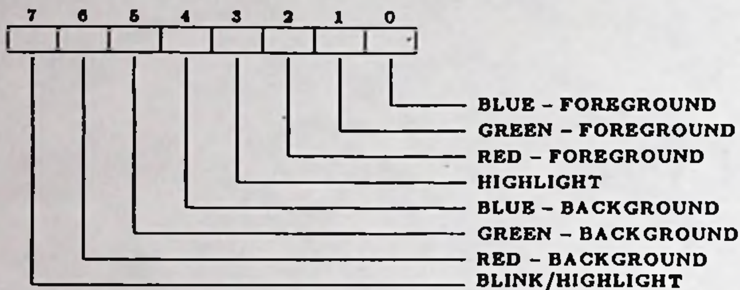


Figure 5.1: Attribute Byte – Alphanumeric Mode

graphics adaptor already present in the system unit, the emulator of the PG-640A can be disabled using the switch described in Appendix A.

5.2 Emulator Access

The emulator is programmed in exactly the same way as the Color Graphics Adaptor. The MS-DOS MODE command can be used to select any of the display modes that are available on the graphics adaptor. Alternately, the mode of the emulator may be altered by writing to the registers described in Section 5.3.

Red	Green	Blue	Highlight	Color
0	0	0	0	Black
0	0	1	0	Blue
0	1	0	0	Green
0	1	1	0	Cyan
1	0	0	0	Red
1	0	1	0	Magenta
1	1	0	0	Brown
1	1	1	0	White
0	0	0	1	Grey
0	0	1	1	Light Blue
0	1	0	1	Light Green
0	1	1	1	Light Cyan
1	0	0	1	Light Red
1	0	1	1	Light Magenta
1	1	0	1	Yellow
1	1	1	1	Bright White

Table 5.1: Alphanumeric Color Table

5.2.1 Video Modes

Alphanumeric Modes

The alphanumeric modes give the user access to 256 extended ASCII characters. This character set includes the standard ASCII numbers and letters (upper and lower case), as well as special characters for graphics and other purposes. The font is illustrated in Figure 3.25. Each character cell is represented in memory by two bytes: one byte for the ASCII code and one byte for the character attribute. This attribute byte allows the user to select the background and character colors, a blink function, and a highlight function. The bit map is illustrated in Figure 5.1.

As each character occupies two bytes, a full screen in 40 × 25 character

Bit 1	Bit 0	color
0	0	Background Color
0	1	Color 0
1	0	Color 1
1	1	Color 2

Table 5.2: 320 × 200 Bit Storage

Number	Color Set 0	Color Set 1
0	Green	Cyan
1	Red	Magenta
2	Brown	White

Table 5.3: 320 × 200 Color Sets

mode takes up only 2 000 bytes of memory and a full screen in 80 × 25 mode: 4 000 bytes. This allows the user to store up to eight screens of 40 × 25 or four screens of 80 × 25 characters at one time. The user also has access to 16 display colors for the foreground, and 16 display colors for the background of each character cell. The color set is illustrated in Table 5.1. Each character cell can also be set to blink off and on using the BLINK bit of the attribute byte.

Graphics Modes

The graphics emulator supports the two pixel addressable graphic modes of the color adaptor, 320 × 200 × 2 and 640 × 200 × 1, both of which require the entire 16Kbytes of the emulator.

In 320 × 200 mode the user can chose one of six pixel colors and one of 16 colors for the background. Each pixel is set using the format laid out

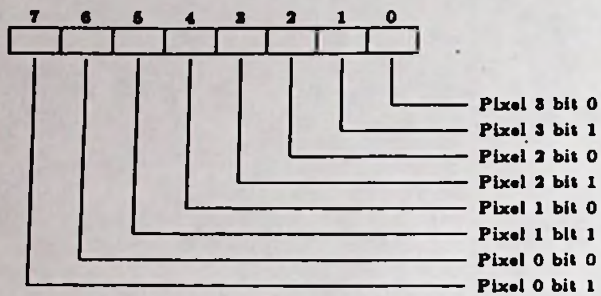


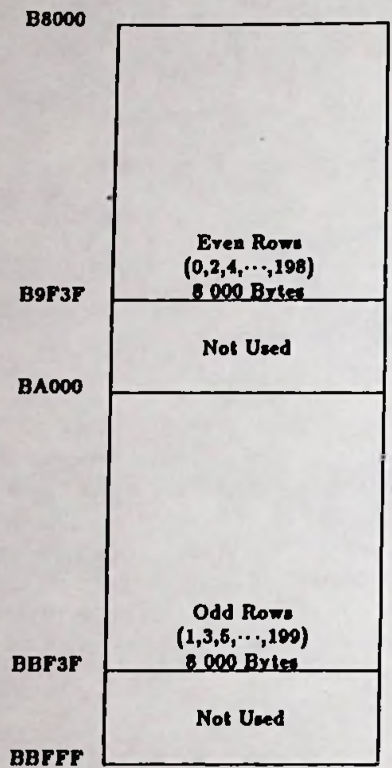
Figure 5.2: 320 × 200 Byte Layout

in Table 5.2. The user can select one of three colors from the current color set, or the background color. There are two color sets, as shown in Table 5.3, one of which is selected using the Color Select Register. Every pixel can be individually addressed from the system unit and in 320 × 200 mode occupies 2 bits of storage. The byte layout is shown in Figure 5.2. The pixel located in the upper left corner of the display is stored at $B8000_H$. Each byte contains data for four pixels and is stored using the format shown in Figure 5.3. The background color is selected using the Color Select Register.

In 640 × 200 mode the memory organisation is much the same as in the 320 × 200 mode, except that each pixel is represented by one bit. This means that each byte stores data for eight pixels (one bit each). Each pixel can be set to the current color or to black – the current color is selected using the Color Select Register.

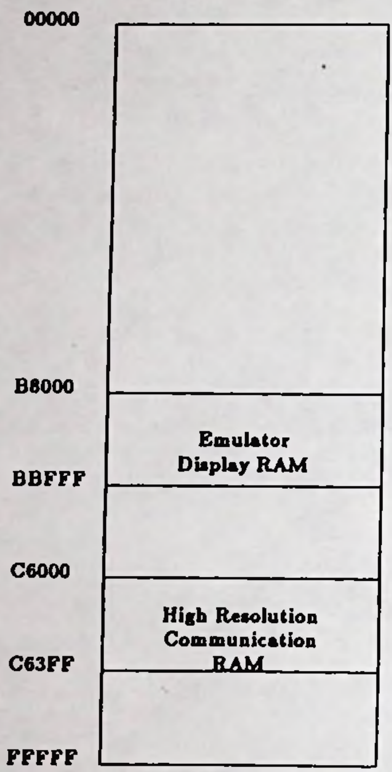
5.2.2 Memory Organisation

The emulator of the PG-640A has 16K by 8 bits of RAM dedicated for emulator display. Where memory is located in the PC's memory map is illustrated in Figure 5.4. The system unit can read or write the



Note: The base address of the CGA RAM can also be set to B0000 (the address of the Monochrome Display Adaptor) using on board straps.

Figure 5.3: Graphics Mode Row Layout



Note: The High Resolution Communications RAM can also be moved from C6000 to C6400 by setting a DIP switch.

Figure 5.4: PG-640A Memory Map

Address	Name
3D4	6845 Index Register
3D5	6845 Data Register
3D8	Mode Control Register
3D9	Color Select Register
3DA	Status Register
3DF	Clear Interrupt Flag

- *Note: These registers can be re-located to 3B0 to 3BF (the location on the Monochrome Display Adaptor) using on board straps. This will allow the user to operate two PG-640A's in the same chassis with emulator windows.*

Table 5.4: Emulator I/O Map

emulator RAM directly, using the CPU address bus, and controls the emulator through the registers described in Section 5.3. The emulator I/O map is illustrated in Table 5.4.

5.3 Register Descriptions

5.3.1 Register Summary

The PG-640A Color Graphics Adaptor Emulator emulates the following registers:

Mode Control Register: Hex address 3D8. This 6 bit write only register controls the display mode of the graphics emulator.

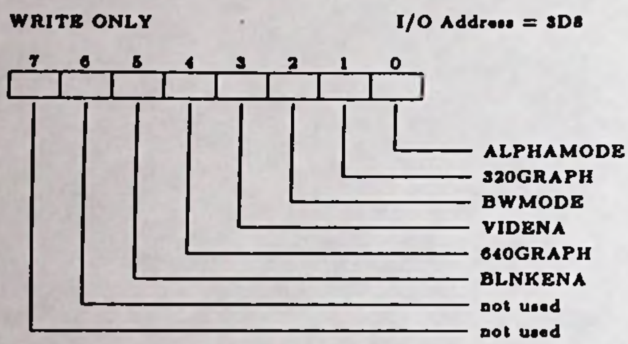
Color Select Register: Hex address 3D9. This 6 bit write only register controls the colors displayed by the graphics emulator.

Status Register: **Hex address 3DA. This 4 bit read only register allows the system unit to read the status of the graphics emulator.**

CRTC Index Register: **Hex address 3D4. This 5 bit write only register is used to point to the internal registers of the 6845 emulator.**

CRTC Data Register: **Hex address 3D5. This 8 bit read/write register is used to indirectly read or write the internal registers of the 6845 emulator.**

5.3.2 Mode Control Register



Bit 0 : **Write a 1 to this bit to select 80 × 25 alphanumeric mode. Write a 0 to select 40 × 25 alphanumeric mode.**

Bit 1 : **Write a 1 to this bit to select 320 × 200 graphics mode. Write a 0 to select alphanumeric mode.**

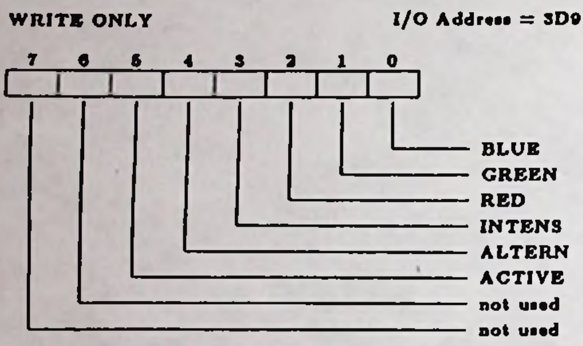
Bit 2 : **Write a 1 to this bit to select black and white mode. Write a 0 to select color mode.**

Bit 3 : **Write a 1 to this bit to enable the video signal. Write a 0 to disable the video signal. The video signal should be disabled when changing modes.**

Bit 4 : Write a 1 to this bit to select 640 × 200 graphics mode. Write a 0 to select alphanumeric mode.

Bit 5 : Write a 1 to this bit to enable the blink function. Write a 0 to disable the blink function. If the blink is disabled, eight intensified colors are made available for the character cell background in the alphanumeric modes.

5.3.3 Color Select Register



Bit 0 : Write a 1 to this bit to select:

1. blue background color in 320 × 200 graphics mode
2. blue foreground color in 640 × 200 graphics mode.

Bit 1 : Write a 1 to this bit to select:

1. green background color in 320 × 200 graphics mode
2. green foreground color in 640 × 200 graphics mode.

Bit 2 : Write a 1 to this bit to select:

1. red background color in 320 × 200 graphics mode

2. red foreground color in 640 × 200 graphics mode.

Bit 3 : Write a 1 to this bit to select:

1. intensified background color in 320 × 200 graphics mode
2. intensified foreground color in 640 × 200 graphics mode.

Bit 4 : Write a 1 to this bit to select:

1. alternate, intensified set of colors in 320 × 200 graphics mode.

Bit 5 : Use this bit to select the active color set in 320 × 200 graphics mode according to the following tables:

1. Bit 5 set to 1:

Bit 1	Bit 0	Set Selected
0	0	Background (Defined by bits 0–3 of port $3D9_H$
0	1	Cyan
1	0	Magenta
1	1	White

2. Bit 5 set to 0:

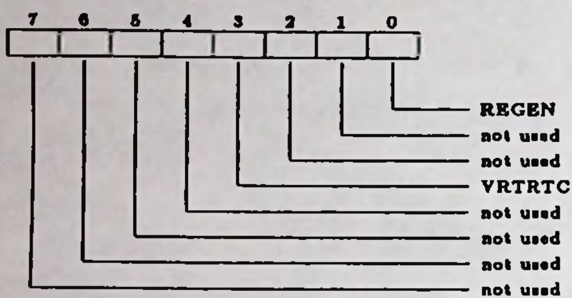
Bit 1	Bit 0	Set Selected
0	0	Background (Defined by bits 0–3 of port $3D9_H$
0	1	Green
1	0	Red
1	1	Brown

3. Bit 5 set to 0 and Bit 2 of the Mode Register set to 1:

Bit 1	Bit 0	Set Selected
0	0	Background (Defined by bits 0–3 of port $3D9_H$
0	1	Cyan
1	0	Red
1	1	White

5.3.4 Status Register

READ ONLY **I/O Address = 3DA**

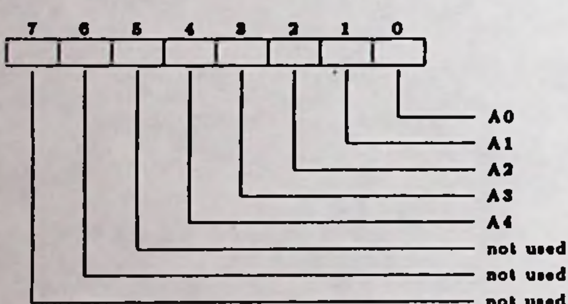


Bit 0 : A 1 in this bit indicates that a emulator buffer memory access can be made without causing disruptions on the display.

Bit 3 : A 1 in this bit indicates that the raster is in vertical retrace – screen buffer updating can be performed at this time.

5.3.5 CRTC Index Register

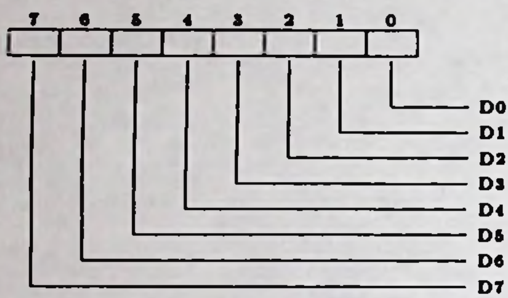
WRITE ONLY **I/O Address = 3D4**



This 5 bit write only register is used as a pointer to the CRT controller's internal registers when initialising the CRT controller for use.

5.3.6 CRTC Data Register

READ/WRITE **I/O Address = 3D5**



This 8-bit read/write register is used to indirectly load data to the CRT controller's internal registers when configuring the CRT controller for use.

5.3.7 6845 CRT Controller Emulator

The 6845 CRT Controller Emulator has ten accessible internal registers which are used to define and control a raster scan CRT display. One of these registers, the Index Register, is used as a pointer for the Data Register which is used to load the other internal registers. See Sections 5.3.5 and 5.3.6.

In order to load any of the other registers the Index Register is first loaded with the necessary pointer then the Data Register is loaded with the data to be placed in the selected register. Likewise the internal registers can be read (if applicable) by writing their address to the Index Register and then reading the Data Register.

Reg Addr	Reg No.	Register Type	Unit	I/O Type	40 by 25 Alpha	80 by 25 Alpha	Graphic Modes
4	R4	Vertical Total	Char Row	Write	1F	1F	1F
5	R5	Vertical Total Adjust	Scan Line	Write	06	06	06
6	R6	Vertical Displayed	Char Row	Write	19	19	19
7	R7	Vertical Sync Position	Char	Write	1C	1C	1C
A	R10	Cursor Start	Scan Line	Write	06	06	06
B	R11	Cursor End	Scan Line	Write	07	07	07
C	R12	Displayed Start Address (H)	—	Write	00	00	00
D	R13	Displayed Start Address (L)	—	Write	00	00	00
E	R14	Cursor Position (H)	—	Read/write	XX	XX	XX
F	R15	Cursor Position (L)	—	Read/write	XX	XX	XX

Table 5.5: 6845 CRT Controller Emulated Registers

Chapter 6

Maintenance and Warranty

Matrox products are warranted against defects in materials and workmanship for a period of 180 days from date of delivery. We will repair or replace products which prove to be defective during the warranty period, provided they are returned to Matrox Electronic Systems Limited. No other warranty is expressed or implied. We are not liable for consequential damages.

To return units for repair:

1. Obtain a Return Materials Acceptance (RMA) Number from our Applications Engineering Department.
2. Fill out the Product Failure Report found at the back of this manual and write the RMA number in the top margin.
3. Return the unit and the completed Product Failure Report to MATROX.

MAINTENANCE AND WARRANTY

U. S. customers are to return their products to our U. S. warehouse, at the following address:

Matrox International Corporation,
Trimex Building,
Mooers, N. Y.
12958.

Appendix A

Installation

A.1 Configuration

A.1.1 CPU Board

Options on the PG-640A are selected using four DIP switches on the CPU board, eight DIP switches on the video board and 12 jumpers on the video board. The switches on the CPU board are:

1. RESERVED. This switch must be OFF.
2. ADDRESS SELECT. When this switch is OFF, the base address of the communications FIFO queue is set to $C6000_H$, when the switch is ON the base address is set to $C6400_H$. This allows two PG-640A's to be installed in the same system unit.
3. COLOR GRAPHICS ADAPTOR ENABLE. When this switch is ON, the color graphics adaptor emulator is enabled. If there already is an IBM color Graphics Adaptor, or equivalent, in the

system unit, the emulator section of the PG-640A should be disabled(switch is OFF).

4. TEST/. This switch is always left OFF. See Appendix G for information on the diagnostics programme.

The CGA Emulator's base address can be strapped to one of two locations: that normally occupied by the CGA (Memory Address B8000, I/O Address 3D0) and that normally occupied by the Monochrome Display Adaptor (Memory Address B0000, I/O Address 3B0). If the CGA Emulator is strapped to B0000, the user is responsible for initialising the CRTC registers. The CGA Emulator's base address is set using the following jumpers:

Configuration	Jumper Settings
Memory B8000, I/O 3D0	1-4, 2-3, 7-8 IN
Memory B0000, I/O 3B0	1-2, 4-5, 6-7 IN

A.1.2 Video Board

DMA Channel Select Switches

The DIP switches on the video board are used to select the DMA channel used by the PG-640A. Follow the table below to choose the appropriate channel. Note : No other board in the system unit may use the same DMA channel. Switch 5 is not used. Switch 1 should be OFF.

Channel	SW2	SW3	SW4	SW6	SW7	SW8
1	OFF	OFF	ON	OFF	OFF	ON
2	OFF	ON	OFF	OFF	ON	OFF
3	ON	OFF	OFF	ON	OFF	OFF

Note: The PG-640A is shipped with Channel 1 selected.

Sync Output Jumpers

The video sync is normally only available on Pin 4 of the video connector. A jumper can be set to have a composite sync added to the green video signal found on Pin 2 of the video connector. See the following table.

Sync	Pins Connections
Normal	5-6 OUT
Sync On Green	5-6 IN

A.2 Installation

To separate the two boards in order to adjust jumpers and switches on the CPU board follow this procedure:

You will need:

- a small Philips screwdriver
- a small (1/4) wrench, or suitable pliers

You should work in a static-free area (avoid carpeting, and don't wear sweaters or other static-generating clothing).

1. Turn off the power on the PC. Remove the PG-640A from your system. If it is hot, let it cool down for a few minutes.
2. Place the PG-640A with the solder side down, component side up on a work bench. (It will scratch a table, so put something underneath it).
3. Remove the four philips screws, and their washers (there is one in each corner). Save the screws and washers.

4. Remove the two small bolts that hold the video connector to the bracket (on the outside side of the bracket). Save them.

5. CAREFULLY, and slowly, separate the two boards. Start at the end furthest from the bracket, and pull the two boards apart. Try not to bend any pins.

6. Make the changes to the switches and jumpers on the CPU board.

7. Now comes the tricky part: putting the two boards back together. First, put the video connector into its hole in the bracket. Then, working from that end, slowly bring the boards together, making sure that the pins, one by one, go into their respective holes. This is tricky, and you may not get it the first time: go slowly. Try not to bend any of the pins.

8. Once all the pins are in their holes, press the two boards together until the tips of the pins just comes through the blue connector. This should not take a great deal of force.

9. Replace the bolts into the video connector. Replace the four philips screws and their washers. Double check that no pins are bent.

To install the PG-640A follow these steps:

1. Turn the PC off and remove the screws at the back of the system unit or the expansion unit and remove the cover.

2. Remove the back panel covers from two adjacent slots.

3. Configure the PG-640A using the jumpers and DIP switches described in the previous section.

4. Firmly press the two boards into the two adjacent slots. Replace screws.

5. If the PG-640A emulator section is enabled, set the DIP switches on the system unit to reflect the addition (if the PG-640A is installed on an IBM PC AT, run the installation program provided with the AT to reconfigure it – in any case, refer to the installation manual which came with your computer).

6. Replace the system unit or expansion unit cover and screws.

7. Plug the video cable from your display into the nine pin connector on the back of the PG-640A.

8. Turn on the power, boot with DOS (version 2.0 or higher), and run STARTUP, which is found on the diskette provided with the PG-640A. STARTUP will test the PG-640A and demonstrate the capabilities of the board.

A.3 Connectors

A.3.1 Video Output

The following table gives the pin numbers and functions for the video output connector.

Pin No.	Signal Name
1	Red Video
2	Green Video
3	Blue Video
4	Horizontal and Vertical Sync
5	Mode Control
6	Ground for Pin 1
7	Ground for Pin 2
8	Ground for Pin 3
9	Ground for Pins 4 & 5

A.3.2 PC Bus Connector

Pin No.	Name	Pin No.	Name
A1	I/O CHECK/	B1	GND
A2	D7	B2	RESET DRV
A3	D6	B3	+5V
A4	D5	B4	IRQ2
A5	D4	B5	-5VDC
A6	D3	B6	DRQ2
A7	D2	B7	-12V
A8	D1	B8	CARD SELECTED/
A9	D0	B9	+12V
A10	I/O CH RDY	B10	GND
A11	AEN	B11	MEMW/
A12	A19	B12	MEMR/
A13	A18	B13	IOW/
A14	A17	B14	IOR/
A15	A16	B15	DACK3/
A16	A15	B16	DRQ3
A17	A14	B17	DACK1/
A18	A13	B18	DRQ1
A19	A12	B19	DACK0/
A20	A11	B20	CLOCK
A21	A10	B21	IRQ7
A22	A9	B22	IRQ6
A23	A8	B23	IRQ5
A24	A7	B24	IRQ4
A25	A6	B25	IRQ3
A26	A5	B26	DACK2/
A27	A4	B27	T/C
A28	A3	B28	ALE
A29	A2	B29	+5V
A30	A1	B30	OSC
A31	A0	B31	GND

Appendix B

Default Parameters

The following table represents the default values after a cold reset of the various matrices, flags and patterns used in the PG-640A.

Name	Default Value	Description
AREAPT	65535 16 times	solid area
CLIPH	0	disabled
CLIPY	0	disabled
COLOR	255	
cross hair screen position	$(320, 240)$	
cross hair coordinate position	$(0, 0)$	
DISPLA	SW3 on CPU board	
DISTAN	500	
DISTH	-30000	
DISTY	30000	
FILMSK	255	all planes used
LINFUN	0	set mode
LINPAT	65535	solid lines
MASK	255	all planes on
MDORG	$(0, 0, 0)$	
2-D current point	$(0, 0)$	
2-D current point screen position	$(320, 240)$	
3-D current point	$(0, 0, 0)$	
PRMFIL	0	off
PROJCT	60	
TANGLE	0	horizontal
TJUST	1,1	left, bottom
TSIZE	8	8 by 12 cells
VWPORT	0,639,0,479	entire screen
VWRPT	$(0, 0, 0)$	
WINDOW	-320,319,-240,239	
transformed 3D point	$(0, 0, 0)$	
TWVIS	0	disabled
TWPOS	all 0	not set up
TWCOL	255,255,255	white
TSTYLE	0	'fat' text
TASPCT	1.5	
TCHROT	0	

Table B.1: Default Values for the PG-640A

Offset*	Location Name	Type	Default
300	Output FIFO write pointer	R/W	0
301	Output FIFO read pointer	R	0
302	Input FIFO write pointer	R	0
303	Input FIFO read pointer	R/W	0
304	Error FIFO write pointer	R	0
305	Error FIFO read pointer	R/W	0
306	Cold Reset	R/W	0
307	Warm Reset	R/W	0
30B	Emulator Switch	R	set by switch
30C	Set Emulator	W	N/A
30D	Emulator Status	R	set by switch
310	DMA Status	R/W	-1

* The address is the Communication Base Address plus the Stated Offset

Table B.2: Communications Area Default Values

DEFAULT PARAMETERS

Appendix C

Specifications

- Ordering Information :
  - PG-640A
  - PG-OCABLE
- Bus :
  - IBM XT or AT, or expansion unit plug-in: uses two adjacent slots with 0.8 inch spacing
- High Resolution Mode :
  - 640 × 480 pixels × 8 bits
  - 256 colours from a palette of more then 16 million
- Emulator Mode :
  - Resolution :
    1. 80 × 25 characters × 16 colours
    2. 40 × 25 characters × 16 colours
    3. 640 × 200 pixels × 1 colour
    4. 320 × 200 pixels × 4 colours

SPECIFICATIONS

- Display Memory Access :
  - pixel access using high level graphics commands
  - DMA Transfers to and from display data storage (Video RAM)
- Performance – High Level Graphics Engine :
  - 40 000 vectors/second (1cm)
  - 5 000 characters/second
  - complete screen image dump : 0.8 second
  - BITBLT : 1 200 000 pixels/second
- Special Functions :
  - IBM Colour Graphics Adaptor Emulation
  - Colour Graphics Emulator window
  - Lookup table with 256 colours from a palette of more then 16 million
  - 320KB of display data storage
  - 128KB of storage for display lists, fonts, and internal variables
  - 1KB FIFO queue for command and data input/output
- Video Timing :
  - Refresh Rate : 60Hz non-interlaced
  - Video Frequency : 25MHz
  - Horizontal Scan Frequency : 30.63kHz
  - Vertical Frame Rate : 60.07Hz
- Video Memory DMA :
  - CPU can read or write any block of pixel
  - Uninterrupted display of memory while processing

- Connectors :
  - One DB9 IBM PGC pin-out RGB output with separate sync and/or composite sync on green
  - 62 pin IBM bus connector
- Power Requirements :
  - +5VDC 4.5A (maximum)
- Dimensions :
  - 335.30mm (13.69in) length
  - 106.77mm (4.35in) height
  - 32.7mm (1.33in) thickness
- Environment :
  - 0°C to 55°C operating temperature
  - 0% to 95% humidity - noncondensing
- Storage :
  - -40°C to 60°C
  - 5% to 100% humidity - noncondensing

Appendix D

The Monitor Program

A monitor program is provided with the PG-640A. This program is in the file PGMON.EXE and allows the user to enter HLGE commands directly into the FIFO buffer. Although both communication modes can be used, hex mode requires the user to type the characters whose ASCII code equals the hex number the user wishes to enter into the FIFO.

D.1 Start Up Procedure

To enter the monitor program, first boot the PC using MS-DOS. Place the diskette provided with the PG-640A into drive A and type PGMON.

D.2 Command Entry

The user enters commands with parameters and commands separated with the delimiters described in Section 3.2. These parameters are entered directly into the FIFO queue and subsequently executed. If an

Function Key	Purpose
F1	Send File
F2	Address C6000/C6400
F3	Cold Reset of PG-640A
F4	Warm Reset of PG-640A
F5	Turn Off CGA Window
F6	Turn On CGA Window
F7	Display CGA Screen (If Enabled)
F8	Display the HLGE Screen
F9	ASCII/Hex Input
F10	ASCII/Hex Output

Table D.1: Function Key Summary

error occurs the message will be displayed on the screen using the current read back mode.

The function keys are used to perform the tasks outlined in Table D.1. Most of the tasks are self-expanatory, F1 will transfer a file to the command buffer of the controller, F2 sets the software to the address of the PG-640A to its initial state, F4 performs a warm reset, F9 determines the mode in which data is sent to the HLGE, and F10 determines how data is returned in hex or ASCII format.

Appendix E

Lookup Table Data

This chapter contains the lookup table data that is provided in ROM on the PG-640A. These tables contain three decimal numbers per entry. The entries are: red, green, and blue (from left to right). These values are given in the format used by the LUTX command (i.e., 8-bit values).

LOOKUP TABLE DATA

State 0 : red, green, blue intensity

```
Entry 0 :    0,   0,   0
Entry 1 :   16,  16,  16
Entry 2 :   32,  32,  32
Entry 3 :   48,  48,  48
Entry 4 :   64,  64,  64
Entry 5 :   80,  80,  80
Entry 6 :   96,  96,  96
Entry 7 :  112, 112, 112
Entry 8 :  128, 128, 128
Entry 9 :  144, 144, 144
Entry 10 : 160, 160, 160
Entry 11 : 176, 176, 176
Entry 12 : 192, 192, 192
Entry 13 : 208, 208, 208
Entry 14 : 224, 224, 224
Entry 15 : 240, 240, 240
Entry 16 :   0,   0,   0
Entry 17 :  32,   0,   0
Entry 18 :  64,   0,   0
Entry 19 :  96,   0,   0
Entry 20 : 128,   0,   0
Entry 21 : 160,   0,   0
Entry 22 : 192,   0,   0
Entry 23 : 224,   0,   0
Entry 24 : 240,   0,   0
Entry 25 : 240,  32,  32
Entry 26 : 240,  64,  64
Entry 27 : 240,  96,  96
Entry 28 : 240, 128, 128
Entry 29 : 240, 160, 160
Entry 30 : 240, 192, 192
Entry 31 : 240, 224, 224
Entry 32 :   0,   0,   0
Entry 33 :  32,   0,  16
Entry 34 :  64,   0,  32
Entry 35 :  96,   0,  48
Entry 36 : 128,   0,  64
Entry 37 : 160,   0,  80
Entry 38 : 192,   0,  96
Entry 39 : 224,   0, 112
Entry 40 : 240,   0, 128
Entry 41 : 240,  32, 144
Entry 42 : 240,  64, 160
Entry 43 : 240,  96, 176
Entry 44 : 240, 128, 192
Entry 45 : 240, 160, 208
Entry 46 : 240, 192, 224
Entry 47 : 240, 224, 240
Entry 48 :   0,   0,   0
Entry 49 :  32,   0,  32
Entry 50 :  64,   0,  64
Entry 51 :  96,   0,  96
Entry 52 : 128,   0, 128
Entry 53 : 160,   0, 160
Entry 54 : 192,   0, 192
Entry 55 : 224,   0, 224
Entry 56 : 240,   0, 240
Entry 57 : 240,  32, 240
Entry 58 : 240,  64, 240
Entry 59 : 240,  96, 240
Entry 60 : 240, 128, 240
Entry 61 : 240, 160, 240
Entry 62 : 240, 192, 240
Entry 63 : 240, 224, 240
Entry 64 :   0,   0,   0
Entry 65 :  16,   0,  32
Entry 66 :  32,   0,  64
Entry 67 :  48,   0,  96
Entry 68 :  64,   0, 128
Entry 69 :  80,   0, 160
Entry 70 :  96,   0, 192
Entry 71 : 112,   0, 224
Entry 72 : 128,   0, 240
Entry 73 : 144,  32, 240
Entry 74 : 160,  64, 240
Entry 75 : 176,  96, 240
Entry 76 : 192, 128, 240
Entry 77 : 208, 160, 240
Entry 78 : 224, 192, 240
Entry 79 : 240, 224, 240
Entry 80 :   0,   0,   0
Entry 81 :   0,   0,  32
Entry 82 :   0,   0,  64
Entry 83 :   0,   0,  96
Entry 84 :   0,   0, 128
Entry 85 :   0,   0, 160
Entry 86 :   0,   0, 192
Entry 87 :   0,   0, 224
Entry 88 :   0,   0, 240
Entry 89 :  32,  32, 240
Entry 90 :  64,  64, 240
```

Entry 91 : 96, 96, 240
Entry 92 : 128, 128, 240
Entry 93 : 160, 160, 240
Entry 94 : 192, 192, 240
Entry 95 : 224, 224, 240
Entry 96 : 0, 0, 0
Entry 97 : 0, 16, 32
Entry 98 : 0, 32, 64
Entry 99 : 0, 48, 96
Entry 100 : 0, 64, 128
Entry 101 : 0, 80, 160
Entry 102 : 0, 96, 192
Entry 103 : 0, 112, 224
Entry 104 : 0, 128, 240
Entry 105 : 32, 144, 240
Entry 106 : 64, 160, 240
Entry 107 : 96, 176, 240
Entry 108 : 128, 192, 240
Entry 109 : 160, 208, 240
Entry 110 : 192, 224, 240
Entry 111 : 224, 240, 240
Entry 112 : 0, 0, 0
Entry 113 : 0, 32, 32
Entry 114 : 0, 64, 64
Entry 115 : 0, 96, 96
Entry 116 : 0, 128, 128
Entry 117 : 0, 160, 160
Entry 118 : 0, 192, 192
Entry 119 : 0, 224, 224
Entry 120 : 0, 240, 240
Entry 121 : 32, 240, 240
Entry 122 : 64, 240, 240
Entry 123 : 96, 240, 240
Entry 124 : 128, 240, 240
Entry 125 : 160, 240, 240
Entry 126 : 192, 240, 240
Entry 127 : 224, 240, 240
Entry 128 : 0, 0, 0
Entry 129 : 0, 32, 16
Entry 130 : 0, 64, 32
Entry 131 : 0, 96, 48
Entry 132 : 0, 128, 64
Entry 133 : 0, 160, 80
Entry 134 : 0, 192, 96
Entry 135 : 0, 224, 112
Entry 136 : 0, 240, 128
Entry 137 : 32, 240, 144
Entry 138 : 64, 240, 160
Entry 139 : 96, 240, 176
Entry 140 : 128, 240, 192
Entry 141 : 160, 240, 208
Entry 142 : 192, 240, 224
Entry 143 : 224, 240, 240
Entry 144 : 0, 0, 0
Entry 145 : 0, 32, 0
Entry 146 : 0, 64, 0
Entry 147 : 0, 96, 0
Entry 148 : 0, 128, 0
Entry 149 : 0, 160, 0
Entry 150 : 0, 192, 0
Entry 151 : 0, 224, 0
Entry 152 : 0, 240, 0
Entry 153 : 32, 240, 32
Entry 154 : 64, 240, 64
Entry 155 : 96, 240, 96
Entry 156 : 128, 240, 128
Entry 157 : 160, 240, 160
Entry 158 : 192, 240, 192
Entry 159 : 224, 240, 224
Entry 160 : 0, 0, 0
Entry 161 : 16, 32, 0
Entry 162 : 32, 64, 0
Entry 163 : 48, 96, 0
Entry 164 : 64, 128, 0
Entry 165 : 80, 160, 0
Entry 166 : 96, 192, 0
Entry 167 : 112, 224, 0
Entry 168 : 128, 240, 0
Entry 169 : 144, 240, 32
Entry 170 : 160, 240, 64
Entry 171 : 176, 240, 96
Entry 172 : 192, 240, 128
Entry 173 : 208, 240, 160
Entry 174 : 224, 240, 192
Entry 175 : 240, 240, 224
Entry 176 : 0, 0, 0
Entry 177 : 32, 32, 0
Entry 178 : 64, 64, 0
Entry 179 : 96, 96, 0
Entry 180 : 128, 128, 0
Entry 181 : 160, 160, 0
Entry 182 : 192, 192, 0
Entry 183 : 224, 224, 0
Entry 184 : 240, 240, 0

LOOKUP TABLE DATA

Entry 185 : 240, 240, 32
Entry 186 : 240, 240, 64
Entry 187 : 240, 240, 96
Entry 188 : 240, 240, 128
Entry 189 : 240, 240, 160
Entry 190 : 240, 240, 192
Entry 191 : 240, 240, 224
Entry 192 : 0, 0, 0
Entry 193 : 32, 16, 0
Entry 194 : 64, 32, 0
Entry 195 : 96, 48, 0
Entry 196 : 128, 64, 0
Entry 197 : 160, 80, 0
Entry 198 : 192, 96, 0
Entry 199 : 224, 112, 0
Entry 200 : 240, 128, 0
Entry 201 : 240, 144, 32
Entry 202 : 240, 160, 64
Entry 203 : 240, 176, 96
Entry 204 : 240, 192, 128
Entry 205 : 240, 208, 160
Entry 206 : 240, 224, 192
Entry 207 : 240, 240, 224
Entry 208 : 0, 0, 0
Entry 209 : 16, 0, 0
Entry 210 : 48, 16, 16
Entry 211 : 64, 16, 16
Entry 212 : 96, 32, 32
Entry 213 : 112, 32, 32
Entry 214 : 144, 48, 48
Entry 215 : 160, 48, 48
Entry 216 : 192, 64, 64
Entry 217 : 192, 80, 80
Entry 218 : 208, 112, 112
Entry 219 : 208, 128, 128
Entry 220 : 224, 160, 160
Entry 221 : 224, 176, 176
Entry 222 : 240, 208, 208
Entry 223 : 240, 224, 224
Entry 224 : 0, 0, 0
Entry 225 : 0, 16, 0
Entry 226 : 16, 48, 16
Entry 227 : 16, 64, 16
Entry 228 : 32, 96, 32
Entry 229 : 32, 112, 32
Entry 230 : 48, 144, 48
Entry 231 : 48, 160, 48
Entry 232 : 64, 192, 64
Entry 233 : 80, 192, 80
Entry 234 : 112, 208, 112
Entry 235 : 128, 208, 128
Entry 236 : 160, 224, 160
Entry 237 : 176, 224, 176
Entry 238 : 208, 240, 208
Entry 239 : 224, 240, 224
Entry 240 : 0, 0, 0
Entry 241 : 0, 0, 16
Entry 242 : 16, 16, 48
Entry 243 : 16, 16, 64
Entry 244 : 32, 32, 96
Entry 245 : 32, 32, 112
Entry 246 : 48, 48, 144
Entry 247 : 48, 48, 160
Entry 248 : 64, 64, 192
Entry 249 : 80, 80, 192
Entry 250 : 112, 112, 208
Entry 251 : 128, 128, 208
Entry 252 : 160, 160, 224
Entry 253 : 176, 176, 224
Entry 254 : 208, 208, 240
Entry 255 : 224, 224, 240

State 1 : red, green, blue intensity

Entry 0 : 96, 128, 208
Entry 1 : 0, 0, 0
Entry 2 : 112, 64, 32
Entry 3 : 160, 112, 64
Entry 4 : 112, 0, 0
Entry 5 : 240, 0, 0
Entry 6 : 240, 112, 0
Entry 7 : 240, 240, 0
Entry 8 : 160, 240, 0
Entry 9 : 0, 240, 0
Entry 10 : 0, 112, 0
Entry 11 : 0, 112, 112
Entry 12 : 0, 0, 112
Entry 13 : 224, 144, 96
Entry 14 : 112, 112, 112
Entry 15 : 240, 240, 240
Entry 16 : 0, 0, 0
Entry 17 : 0, 0, 0

```
Entry 18 :   0,   0,   0
Entry 19 :   0,   0,   0
Entry 20 :   0,   0,   0
Entry 21 :   0,   0,   0
Entry 22 :   0,   0,   0
Entry 23 :   0,   0,   0
Entry 24 :   0,   0,   0
Entry 25 :   0,   0,   0
Entry 26 :   0,   0,   0
Entry 27 :   0,   0,   0
Entry 28 :   0,   0,   0
Entry 29 :   0,   0,   0
Entry 30 :   0,   0,   0
Entry 31 :   0,   0,   0
Entry 32 : 112,  64,  32
Entry 33 : 112,  64,  32
Entry 34 : 112,  64,  32
Entry 35 : 112,  64,  32
Entry 36 : 112,  64,  32
Entry 37 : 112,  64,  32
Entry 38 : 112,  64,  32
Entry 39 : 112,  64,  32
Entry 40 : 112,  64,  32
Entry 41 : 112,  64,  32
Entry 42 : 112,  64,  32
Entry 43 : 112,  64,  32
Entry 44 : 112,  64,  32
Entry 45 : 112,  64,  32
Entry 46 : 112,  64,  32
Entry 47 : 112,  64,  32
Entry 48 : 160, 112,  64
Entry 49 : 160, 112,  64
Entry 50 : 160, 112,  64
Entry 51 : 160, 112,  64
Entry 52 : 160, 112,  64
Entry 53 : 160, 112,  64
Entry 54 : 160, 112,  64
Entry 55 : 160, 112,  64
Entry 56 : 160, 112,  64
Entry 57 : 160, 112,  64
Entry 58 : 160, 112,  64
Entry 59 : 160, 112,  64
Entry 60 : 160, 112,  64
Entry 61 : 160, 112,  64
Entry 62 : 160, 112,  64
Entry 63 : 160, 112,  64
Entry 64 : 112,   0,   0
Entry 65 : 112,   0,   0
Entry 66 : 112,   0,   0
Entry 67 : 112,   0,   0
Entry 68 : 112,   0,   0
Entry 69 : 112,   0,   0
Entry 70 : 112,   0,   0
Entry 71 : 112,   0,   0
Entry 72 : 112,   0,   0
Entry 73 : 112,   0,   0
Entry 74 : 112,   0,   0
Entry 75 : 112,   0,   0
Entry 76 : 112,   0,   0
Entry 77 : 112,   0,   0
Entry 78 : 112,   0,   0
Entry 79 : 112,   0,   0
Entry 80 : 240,   0,   0
Entry 81 : 240,   0,   0
Entry 82 : 240,   0,   0
Entry 83 : 240,   0,   0
Entry 84 : 240,   0,   0
Entry 85 : 240,   0,   0
Entry 86 : 240,   0,   0
Entry 87 : 240,   0,   0
Entry 88 : 240,   0,   0
Entry 89 : 240,   0,   0
Entry 90 : 240,   0,   0
Entry 91 : 240,   0,   0
Entry 92 : 240,   0,   0
Entry 93 : 240,   0,   0
Entry 94 : 240,   0,   0
Entry 95 : 240,   0,   0
Entry 96 : 240, 112,   0
Entry 97 : 240, 112,   0
Entry 98 : 240, 112,   0
Entry 99 : 240, 112,   0
Entry 100 : 240, 112,   0
Entry 101 : 240, 112,   0
Entry 102 : 240, 112,   0
Entry 103 : 240, 112,   0
Entry 104 : 240, 112,   0
Entry 105 : 240, 112,   0
Entry 106 : 240, 112,   0
Entry 107 : 240, 112,   0
Entry 108 : 240, 112,   0
Entry 109 : 240, 112,   0
Entry 110 : 240, 112,   0
Entry 111 : 240, 112,   0
```

LOOKUP TABLE DATA

Entry 112 :	240,	240,	0
Entry 113 :	240,	240,	0
Entry 114 :	240,	240,	0
Entry 115 :	240,	240,	0
Entry 116 :	240,	240,	0
Entry 117 :	240,	240,	0
Entry 118 :	240,	240,	0
Entry 119 :	240,	240,	0
Entry 120 :	240,	240,	0
Entry 121 :	240,	240,	0
Entry 122 :	240,	240,	0
Entry 123 :	240,	240,	0
Entry 124 :	240,	240,	0
Entry 125 :	240,	240,	0
Entry 126 :	240,	240,	0
Entry 127 :	240,	240,	0
Entry 128 :	160,	240,	0
Entry 129 :	160,	240,	0
Entry 130 :	160,	240,	0
Entry 131 :	160,	240,	0
Entry 132 :	160,	240,	0
Entry 133 :	160,	240,	0
Entry 134 :	160,	240,	0
Entry 135 :	160,	240,	0
Entry 136 :	160,	240,	0
Entry 137 :	160,	240,	0
Entry 138 :	160,	240,	0
Entry 139 :	160,	240,	0
Entry 140 :	160,	240,	0
Entry 141 :	160,	240,	0
Entry 142 :	160,	240,	0
Entry 143 :	160,	240,	0
Entry 144 :	0,	240,	0
Entry 145 :	0,	240,	0
Entry 146 :	0,	240,	0
Entry 147 :	0,	240,	0
Entry 148 :	0,	240,	0
Entry 149 :	0,	240,	0
Entry 150 :	0,	240,	0
Entry 151 :	0,	240,	0
Entry 152 :	0,	240,	0
Entry 153 :	0,	240,	0
Entry 154 :	0,	240,	0
Entry 155 :	0,	240,	0
Entry 156 :	0,	240,	0
Entry 157 :	0,	240,	0
Entry 158 :	0,	240,	0
Entry 159 :	0,	240,	0
Entry 160 :	0,	112,	0
Entry 161 :	0,	112,	0
Entry 162 :	0,	112,	0
Entry 163 :	0,	112,	0
Entry 164 :	0,	112,	0
Entry 165 :	0,	112,	0
Entry 166 :	0,	112,	0
Entry 167 :	0,	112,	0
Entry 168 :	0,	112,	0
Entry 169 :	0,	112,	0
Entry 170 :	0,	112,	0
Entry 171 :	0,	112,	0
Entry 172 :	0,	112,	0
Entry 173 :	0,	112,	0
Entry 174 :	0,	112,	0
Entry 175 :	0,	112,	0
Entry 176 :	0,	112,	112
Entry 177 :	0,	112,	112
Entry 178 :	0,	112,	112
Entry 179 :	0,	112,	112
Entry 180 :	0,	112,	112
Entry 181 :	0,	112,	112
Entry 182 :	0,	112,	112
Entry 183 :	0,	112,	112
Entry 184 :	0,	112,	112
Entry 185 :	0,	112,	112
Entry 186 :	0,	112,	112
Entry 187 :	0,	112,	112
Entry 188 :	0,	112,	112
Entry 189 :	0,	112,	112
Entry 190 :	0,	112,	112
Entry 191 :	0,	112,	112
Entry 192 :	0,	0,	112
Entry 193 :	0,	0,	112
Entry 194 :	0,	0,	112
Entry 195 :	0,	0,	112
Entry 196 :	0,	0,	112
Entry 197 :	0,	0,	112
Entry 198 :	0,	0,	112
Entry 199 :	0,	0,	112
Entry 200 :	0,	0,	112
Entry 201 :	0,	0,	112
Entry 202 :	0,	0,	112
Entry 203 :	0,	0,	112
Entry 204 :	0,	0,	112
Entry 205 :	0,	0,	112

Entry 205 : 0, 0, 112
Entry 207 : 0, 0, 112
Entry 208 : 224, 144, 96
Entry 209 : 224, 144, 96
Entry 210 : 224, 144, 96
Entry 211 : 224, 144, 96
Entry 212 : 224, 144, 96
Entry 213 : 224, 144, 96
Entry 214 : 224, 144, 96
Entry 215 : 224, 144, 96
Entry 216 : 224, 144, 96
Entry 217 : 224, 144, 96
Entry 218 : 224, 144, 96
Entry 219 : 224, 144, 96
Entry 220 : 224, 144, 96
Entry 221 : 224, 144, 96
Entry 222 : 224, 144, 96
Entry 223 : 224, 144, 96
Entry 224 : 112, 112, 112
Entry 225 : 112, 112, 112
Entry 226 : 112, 112, 112
Entry 227 : 112, 112, 112
Entry 228 : 112, 112, 112
Entry 229 : 112, 112, 112
Entry 230 : 112, 112, 112
Entry 231 : 112, 112, 112
Entry 232 : 112, 112, 112
Entry 233 : 112, 112, 112
Entry 234 : 112, 112, 112
Entry 235 : 112, 112, 112
Entry 236 : 112, 112, 112
Entry 237 : 112, 112, 112
Entry 238 : 112, 112, 112
Entry 239 : 112, 112, 112
Entry 240 : 240, 240, 240
Entry 241 : 240, 240, 240
Entry 242 : 240, 240, 240
Entry 243 : 240, 240, 240
Entry 244 : 240, 240, 240
Entry 245 : 240, 240, 240
Entry 246 : 240, 240, 240
Entry 247 : 240, 240, 240
Entry 248 : 240, 240, 240
Entry 249 : 240, 240, 240
Entry 250 : 240, 240, 240
Entry 251 : 240, 240, 240
Entry 252 : 240, 240, 240
Entry 253 : 240, 240, 240
Entry 254 : 240, 240, 240
Entry 255 : 240, 240, 240

State 2 : red, green, blue intensity

Entry 0 : 0, 0, 0
Entry 1 : 0, 0, 48
Entry 2 : 0, 0, 80
Entry 3 : 0, 0, 112
Entry 4 : 0, 0, 144
Entry 5 : 0, 0, 176
Entry 6 : 0, 0, 208
Entry 7 : 0, 0, 240
Entry 8 : 0, 48, 0
Entry 9 : 0, 48, 48
Entry 10 : 0, 48, 80
Entry 11 : 0, 48, 112
Entry 12 : 0, 48, 144
Entry 13 : 0, 48, 176
Entry 14 : 0, 48, 208
Entry 15 : 0, 48, 240
Entry 16 : 0, 80, 0
Entry 17 : 0, 80, 48
Entry 18 : 0, 80, 80
Entry 19 : 0, 80, 112
Entry 20 : 0, 80, 144
Entry 21 : 0, 80, 176
Entry 22 : 0, 80, 208
Entry 23 : 0, 80, 240
Entry 24 : 0, 112, 0
Entry 25 : 0, 112, 48
Entry 26 : 0, 112, 80
Entry 27 : 0, 112, 112
Entry 28 : 0, 112, 144
Entry 29 : 0, 112, 176
Entry 30 : 0, 112, 208
Entry 31 : 0, 112, 240
Entry 32 : 0, 144, 0
Entry 33 : 0, 144, 48
Entry 34 : 0, 144, 80
Entry 35 : 0, 144, 112
Entry 36 : 0, 144, 144
Entry 37 : 0, 144, 176
Entry 38 : 0, 144, 208

LOOKUP TABLE DATA

```
Entry 39 :   0, 144, 240
Entry 40 :   0, 176,   0
Entry 41 :   0, 176,  48
Entry 42 :   0, 176,  80
Entry 43 :   0, 176, 112
Entry 44 :   0, 176, 144
Entry 45 :   0, 176, 176
Entry 46 :   0, 176, 208
Entry 47 :   0, 176, 240
Entry 48 :   0, 208,   0
Entry 49 :   0, 208,  48
Entry 50 :   0, 208,  80
Entry 51 :   0, 208, 112
Entry 52 :   0, 208, 144
Entry 53 :   0, 208, 176
Entry 54 :   0, 208, 208
Entry 55 :   0, 208, 240
Entry 56 :   0, 240,   0
Entry 57 :   0, 240,  48
Entry 58 :   0, 240,  80
Entry 59 :   0, 240, 112
Entry 60 :   0, 240, 144
Entry 61 :   0, 240, 176
Entry 62 :   0, 240, 208
Entry 63 :   0, 240, 240
Entry 64 :  80,   0,   0
Entry 65 :  80,   0,  48
Entry 66 :  80,   0,  80
Entry 67 :  80,   0, 112
Entry 68 :  80,   0, 144
Entry 69 :  80,   0, 176
Entry 70 :  80,   0, 208
Entry 71 :  80,   0, 240
Entry 72 :  80,  48,   0
Entry 73 :  80,  48,  48
Entry 74 :  80,  48,  80
Entry 75 :  80,  48, 112
Entry 76 :  80,  48, 144
Entry 77 :  80,  48, 176
Entry 78 :  80,  48, 208
Entry 79 :  80,  48, 240
Entry 80 :  80,  80,   0
Entry 81 :  80,  80,  48
Entry 82 :  80,  80,  80
Entry 83 :  80,  80, 112
Entry 84 :  80,  80, 144
Entry 85 :  80,  80, 176
Entry 86 :  80,  80, 208
Entry 87 :  80,  80, 240
Entry 88 :  80, 112,   0
Entry 89 :  80, 112,  48
Entry 90 :  80, 112,  80
Entry 91 :  80, 112, 112
Entry 92 :  80, 112, 144
Entry 93 :  80, 112, 176
Entry 94 :  80, 112, 208
Entry 95 :  80, 112, 240
Entry 96 :  80, 144,   0
Entry 97 :  80, 144,  48
Entry 98 :  80, 144,  80
Entry 99 :  80, 144, 112
Entry 100 :  80, 144, 144
Entry 101 :  80, 144, 176
Entry 102 :  80, 144, 208
Entry 103 :  80, 144, 240
Entry 104 :  80, 176,   0
Entry 105 :  80, 176,  48
Entry 106 :  80, 176,  80
Entry 107 :  80, 176, 112
Entry 108 :  80, 176, 144
Entry 109 :  80, 176, 176
Entry 110 :  80, 176, 208
Entry 111 :  80, 176, 240
Entry 112 :  80, 208,   0
Entry 113 :  80, 208,  48
Entry 114 :  80, 208,  80
Entry 115 :  80, 208, 112
Entry 116 :  80, 208, 144
Entry 117 :  80, 208, 176
Entry 118 :  80, 208, 208
Entry 119 :  80, 208, 240
Entry 120 :  80, 240,   0
Entry 121 :  80, 240,  48
Entry 122 :  80, 240,  80
Entry 123 :  80, 240, 112
Entry 124 :  80, 240, 144
Entry 125 :  80, 240, 176
Entry 126 :  80, 240, 208
Entry 127 :  80, 240, 240
Entry 128 : 160,   0,   0
Entry 129 : 160,   0,  48
Entry 130 : 160,   0,  80
Entry 131 : 160,   0, 112
Entry 132 : 160,   0, 144
```

Entry 133 : 160, 0, 176
Entry 134 : 160, 0, 208
Entry 135 : 160, 0, 240
Entry 136 : 160, 48, 0
Entry 137 : 160, 48, 48
Entry 138 : 160, 48, 80
Entry 139 : 160, 48, 112
Entry 140 : 160, 48, 144
Entry 141 : 160, 48, 176
Entry 142 : 160, 48, 208
Entry 143 : 160, 48, 240
Entry 144 : 160, 80, 0
Entry 145 : 160, 80, 48
Entry 146 : 160, 80, 80
Entry 147 : 160, 80, 112
Entry 148 : 160, 80, 144
Entry 149 : 160, 80, 176
Entry 150 : 160, 80, 208
Entry 151 : 160, 80, 240
Entry 152 : 160, 112, 0
Entry 153 : 160, 112, 48
Entry 154 : 160, 112, 80
Entry 155 : 160, 112, 112
Entry 156 : 160, 112, 144
Entry 157 : 160, 112, 176
Entry 158 : 160, 112, 208
Entry 159 : 160, 112, 240
Entry 160 : 160, 144, 0
Entry 161 : 160, 144, 48
Entry 162 : 160, 144, 80
Entry 163 : 160, 144, 112
Entry 164 : 160, 144, 144
Entry 165 : 160, 144, 176
Entry 166 : 160, 144, 208
Entry 167 : 160, 144, 240
Entry 168 : 160, 176, 0
Entry 169 : 160, 176, 48
Entry 170 : 160, 176, 80
Entry 171 : 160, 176, 112
Entry 172 : 160, 176, 144
Entry 173 : 160, 176, 176
Entry 174 : 160, 176, 208
Entry 175 : 160, 176, 240
Entry 176 : 160, 208, 0
Entry 177 : 160, 208, 48
Entry 178 : 160, 208, 80
Entry 179 : 160, 208, 112
Entry 180 : 160, 208, 144
Entry 181 : 160, 208, 176
Entry 182 : 160, 208, 208
Entry 183 : 160, 208, 240
Entry 184 : 160, 240, 0
Entry 185 : 160, 240, 48
Entry 186 : 160, 240, 80
Entry 187 : 160, 240, 112
Entry 188 : 160, 240, 144
Entry 189 : 160, 240, 176
Entry 190 : 160, 240, 208
Entry 191 : 160, 240, 224
Entry 192 : 240, 0, 0
Entry 193 : 240, 0, 48
Entry 194 : 240, 0, 80
Entry 195 : 240, 0, 112
Entry 196 : 240, 0, 144
Entry 197 : 240, 0, 176
Entry 198 : 240, 0, 208
Entry 199 : 240, 0, 240
Entry 200 : 240, 48, 0
Entry 201 : 240, 48, 48
Entry 202 : 240, 48, 80
Entry 203 : 240, 48, 112
Entry 204 : 240, 48, 144
Entry 205 : 240, 48, 176
Entry 206 : 240, 48, 208
Entry 207 : 240, 48, 240
Entry 208 : 240, 80, 0
Entry 209 : 240, 80, 48
Entry 210 : 240, 80, 80
Entry 211 : 240, 80, 112
Entry 212 : 240, 80, 144
Entry 213 : 240, 80, 176
Entry 214 : 240, 80, 208
Entry 215 : 240, 80, 240
Entry 216 : 240, 112, 0
Entry 217 : 240, 112, 48
Entry 218 : 240, 112, 80
Entry 219 : 240, 112, 112
Entry 220 : 240, 112, 144
Entry 221 : 240, 112, 176
Entry 222 : 240, 112, 208
Entry 223 : 240, 112, 240
Entry 224 : 240, 144, 0
Entry 225 : 240, 144, 48
Entry 226 : 240, 144, 80

LOOKUP TABLE DATA

```
Entry 227 : 240, 144, 112
Entry 228 : 240, 144, 144
Entry 229 : 240, 144, 176
Entry 230 : 240, 144, 208
Entry 231 : 240, 144, 240
Entry 232 : 240, 176,   0
Entry 233 : 240, 176,  48
Entry 234 : 240, 176,  80
Entry 235 : 240, 176, 112
Entry 236 : 240, 176, 144
Entry 237 : 240, 176, 176
Entry 238 : 240, 176, 208
Entry 239 : 240, 176, 240
Entry 240 : 240, 208,   0
Entry 241 : 240, 208,  48
Entry 242 : 240, 208,  80
Entry 243 : 240, 208, 112
Entry 244 : 240, 208, 144
Entry 245 : 240, 208, 176
Entry 246 : 240, 208, 208
Entry 247 : 240, 208, 240
Entry 248 : 240, 240,   0
Entry 249 : 240, 240,  48
Entry 250 : 240, 240,  80
Entry 251 : 240, 240, 112
Entry 252 : 240, 240, 144
Entry 253 : 240, 240, 176
Entry 254 : 240, 240, 208
Entry 255 : 240, 240, 240


 State 3 : red, green, blue intensity

Entry 0 :   0,   0,   0
Entry 1 :   0,   0,  48
Entry 2 :   0,   0,  80
Entry 3 :   0,   0, 112
Entry 4 :   0,   0, 144
Entry 5 :   0,   0, 176
Entry 6 :   0,   0, 208
Entry 7 :   0,   0, 240
Entry 8 :   0,  80,   0
Entry 9 :   0,  80,  48
Entry 10 :   0,  80,  80
Entry 11 :   0,  80, 112
Entry 12 :   0,  80, 144
Entry 13 :   0,  80, 176
Entry 14 :   0,  80, 208
Entry 15 :   0,  80, 240
Entry 16 :   0, 160,   0
Entry 17 :   0, 160,  48
Entry 18 :   0, 160,  80
Entry 19 :   0, 160, 112
Entry 20 :   0, 160, 144
Entry 21 :   0, 160, 176
Entry 22 :   0, 160, 208
Entry 23 :   0, 160, 240
Entry 24 :   0, 240,   0
Entry 25 :   0, 240,  48
Entry 26 :   0, 240,  80
Entry 27 :   0, 240, 112
Entry 28 :   0, 240, 144
Entry 29 :   0, 240, 176
Entry 30 :   0, 240, 208
Entry 31 :   0, 240, 240
Entry 32 :  48,   0,   0
Entry 33 :  48,   0,  48
Entry 34 :  48,   0,  80
Entry 35 :  48,   0, 112
Entry 36 :  48,   0, 144
Entry 37 :  48,   0, 176
Entry 38 :  48,   0, 208
Entry 39 :  48,   0, 240
Entry 40 :  48,  80,   0
Entry 41 :  48,  80,  48
Entry 42 :  48,  80,  80
Entry 43 :  48,  80, 112
Entry 44 :  48,  80, 144
Entry 45 :  48,  80, 176
Entry 46 :  48,  80, 208
Entry 47 :  48,  80, 240
Entry 48 :  48, 160,   0
Entry 49 :  48, 160,  48
Entry 50 :  48, 160,  80
Entry 51 :  48, 160, 112
Entry 52 :  48, 160, 144
Entry 53 :  48, 160, 176
Entry 54 :  48, 160, 208
Entry 55 :  48, 160, 240
Entry 56 :  48, 240,   0
Entry 57 :  48, 240,  48
Entry 58 :  48, 240,  80
Entry 59 :  48, 240, 112
```

Entry 60 : 48, 240, 144
Entry 61 : 48, 240, 176
Entry 62 : 48, 240, 208
Entry 63 : 48, 240, 240
Entry 64 : 80, 0, 0
Entry 65 : 80, 0, 48
Entry 66 : 80, 0, 80
Entry 67 : 80, 0, 112
Entry 68 : 80, 0, 144
Entry 69 : 80, 0, 176
Entry 70 : 80, 0, 208
Entry 71 : 80, 0, 240
Entry 72 : 80, 80, 0
Entry 73 : 80, 80, 48
Entry 74 : 80, 80, 80
Entry 75 : 80, 80, 112
Entry 76 : 80, 80, 144
Entry 77 : 80, 80, 176
Entry 78 : 80, 80, 208
Entry 79 : 80, 80, 240
Entry 80 : 80, 160, 0
Entry 81 : 80, 160, 48
Entry 82 : 80, 160, 80
Entry 83 : 80, 160, 112
Entry 84 : 80, 160, 144
Entry 85 : 80, 160, 176
Entry 86 : 80, 160, 208
Entry 87 : 80, 160, 240
Entry 88 : 80, 240, 0
Entry 89 : 80, 240, 48
Entry 90 : 80, 240, 80
Entry 91 : 80, 240, 112
Entry 92 : 80, 240, 144
Entry 93 : 80, 240, 176
Entry 94 : 80, 240, 208
Entry 95 : 80, 240, 240
Entry 96 : 112, 0, 0
Entry 97 : 112, 0, 48
Entry 98 : 112, 0, 80
Entry 99 : 112, 0, 112
Entry 100 : 112, 0, 144
Entry 101 : 112, 0, 176
Entry 102 : 112, 0, 208
Entry 103 : 112, 0, 240
Entry 104 : 112, 80, 0
Entry 105 : 112, 80, 48
Entry 106 : 112, 80, 80
Entry 107 : 112, 80, 112
Entry 108 : 112, 80, 144
Entry 109 : 112, 80, 176
Entry 110 : 112, 80, 208
Entry 111 : 112, 80, 240
Entry 112 : 112, 160, 0
Entry 113 : 112, 160, 48
Entry 114 : 112, 160, 80
Entry 115 : 112, 160, 112
Entry 116 : 112, 160, 144
Entry 117 : 112, 160, 176
Entry 118 : 112, 160, 208
Entry 119 : 112, 160, 240
Entry 120 : 112, 240, 0
Entry 121 : 112, 240, 48
Entry 122 : 112, 240, 80
Entry 123 : 112, 240, 112
Entry 124 : 112, 240, 144
Entry 125 : 112, 240, 176
Entry 126 : 112, 240, 208
Entry 127 : 112, 240, 240
Entry 128 : 144, 0, 0
Entry 129 : 144, 0, 48
Entry 130 : 144, 0, 80
Entry 131 : 144, 0, 112
Entry 132 : 144, 0, 144
Entry 133 : 144, 0, 176
Entry 134 : 144, 0, 208
Entry 135 : 144, 0, 240
Entry 136 : 144, 80, 0
Entry 137 : 144, 80, 48
Entry 138 : 144, 80, 80
Entry 139 : 144, 80, 112
Entry 140 : 144, 80, 144
Entry 141 : 144, 80, 176
Entry 142 : 144, 80, 208
Entry 143 : 144, 80, 240
Entry 144 : 144, 160, 0
Entry 145 : 144, 160, 48
Entry 146 : 144, 160, 80
Entry 147 : 144, 160, 112
Entry 148 : 144, 160, 144
Entry 149 : 144, 160, 176
Entry 150 : 144, 160, 208
Entry 151 : 144, 160, 240
Entry 152 : 144, 240, 0
Entry 153 : 144, 240, 48

LOOKUP TABLE DATA

```
Entry 154 : 144, 240,  80
Entry 155 : 144, 240, 112
Entry 156 : 144, 240, 144
Entry 157 : 144, 240, 176
Entry 158 : 144, 240, 208
Entry 159 : 144, 240, 240
Entry 160 : 176,   0,   0
Entry 161 : 176,   0,  48
Entry 162 : 176,   0,  80
Entry 163 : 176,   0, 112
Entry 164 : 176,   0, 144
Entry 165 : 176,   0, 176
Entry 166 : 176,   0, 208
Entry 167 : 176,   0, 240
Entry 168 : 176,  80,   0
Entry 169 : 176,  80,  48
Entry 170 : 176,  80,  80
Entry 171 : 176,  80, 112
Entry 172 : 176,  80, 144
Entry 173 : 176,  80, 176
Entry 174 : 176,  80, 208
Entry 175 : 176,  80, 240
Entry 176 : 176, 160,   0
Entry 177 : 176, 160,  48
Entry 178 : 176, 160,  80
Entry 179 : 176, 160, 112
Entry 180 : 176, 160, 144
Entry 181 : 176, 160, 176
Entry 182 : 176, 160, 208
Entry 183 : 176, 160, 240
Entry 184 : 176, 240,   0
Entry 185 : 176, 240,  48
Entry 186 : 176, 240,  80
Entry 187 : 176, 240, 112
Entry 188 : 176, 240, 144
Entry 189 : 176, 240, 176
Entry 190 : 176, 240, 208
Entry 191 : 176, 240, 240
Entry 192 : 208,   0,   0
Entry 193 : 208,   0,  48
Entry 194 : 208,   0,  80
Entry 195 : 208,   0, 112
Entry 196 : 208,   0, 144
Entry 197 : 208,   0, 176
Entry 198 : 208,   0, 208
Entry 199 : 208,   0, 240
Entry 200 : 208,  80,   0
Entry 201 : 208,  80,  48
Entry 202 : 208,  80,  80
Entry 203 : 208,  80, 112
Entry 204 : 208,  80, 144
Entry 205 : 208,  80, 176
Entry 206 : 208,  80, 208
Entry 207 : 208,  80, 240
Entry 208 : 208, 160,   0
Entry 209 : 208, 160,  48
Entry 210 : 208, 160,  80
Entry 211 : 208, 160, 112
Entry 212 : 208, 160, 144
Entry 213 : 208, 160, 176
Entry 214 : 208, 160, 208
Entry 215 : 208, 160, 240
Entry 216 : 208, 240,   0
Entry 217 : 208, 240,  48
Entry 218 : 208, 240,  80
Entry 219 : 208, 240, 112
Entry 220 : 208, 240, 144
Entry 221 : 208, 240, 176
Entry 222 : 208, 240, 208
Entry 223 : 208, 240, 240
Entry 224 : 240,   0,   0
Entry 225 : 240,   0,  48
Entry 226 : 240,   0,  80
Entry 227 : 240,   0, 112
Entry 228 : 240,   0, 144
Entry 229 : 240,   0, 176
Entry 230 : 240,   0, 208
Entry 231 : 240,   0, 240
Entry 232 : 240,  80,   0
Entry 233 : 240,  80,  48
Entry 234 : 240,  80,  80
Entry 235 : 240,  80, 112
Entry 236 : 240,  80, 144
Entry 237 : 240,  80, 176
Entry 238 : 240,  80, 208
Entry 239 : 240,  80, 240
Entry 240 : 240, 160,   0
Entry 241 : 240, 160,  48
Entry 242 : 240, 160,  80
Entry 243 : 240, 160, 112
Entry 244 : 240, 160, 144
Entry 245 : 240, 160, 176
Entry 246 : 240, 160, 208
Entry 247 : 240, 160, 240
```

Entry 248 : 240, 240, 0
Entry 249 : 240, 240, 48
Entry 250 : 240, 240, 80
Entry 251 : 240, 240, 112
Entry 252 : 240, 240, 144
Entry 253 : 240, 240, 176
Entry 254 : 240, 240, 208
Entry 255 : 240, 240, 240

State 4 : red, green, blue intensity

Entry 0 : 0, 0, 0
Entry 1 : 0, 0, 80
Entry 2 : 0, 0, 160
Entry 3 : 0, 0, 240
Entry 4 : 0, 48, 0
Entry 5 : 0, 48, 80
Entry 6 : 0, 48, 160
Entry 7 : 0, 48, 240
Entry 8 : 0, 80, 0
Entry 9 : 0, 80, 80
Entry 10 : 0, 80, 160
Entry 11 : 0, 80, 240
Entry 12 : 0, 112, 0
Entry 13 : 0, 112, 80
Entry 14 : 0, 112, 160
Entry 15 : 0, 112, 240
Entry 16 : 0, 144, 0
Entry 17 : 0, 144, 80
Entry 18 : 0, 144, 160
Entry 19 : 0, 144, 240
Entry 20 : 0, 176, 0
Entry 21 : 0, 176, 80
Entry 22 : 0, 176, 160
Entry 23 : 0, 176, 240
Entry 24 : 0, 208, 0
Entry 25 : 0, 208, 80
Entry 26 : 0, 208, 160
Entry 27 : 0, 208, 240
Entry 28 : 0, 240, 0
Entry 29 : 0, 240, 80
Entry 30 : 0, 240, 160
Entry 31 : 0, 240, 240
Entry 32 : 48, 0, 0
Entry 33 : 48, 0, 80
Entry 34 : 48, 0, 160
Entry 35 : 48, 0, 240
Entry 36 : 48, 48, 0
Entry 37 : 48, 48, 80
Entry 38 : 48, 48, 160
Entry 39 : 48, 48, 240
Entry 40 : 48, 80, 0
Entry 41 : 48, 80, 80
Entry 42 : 48, 80, 160
Entry 43 : 48, 80, 240
Entry 44 : 48, 112, 0
Entry 45 : 48, 112, 80
Entry 46 : 48, 112, 160
Entry 47 : 48, 112, 240
Entry 48 : 48, 144, 0
Entry 49 : 48, 144, 80
Entry 50 : 48, 144, 160
Entry 51 : 48, 144, 240
Entry 52 : 48, 176, 0
Entry 53 : 48, 176, 80
Entry 54 : 48, 176, 160
Entry 55 : 48, 176, 240
Entry 56 : 48, 208, 0
Entry 57 : 48, 208, 80
Entry 58 : 48, 208, 160
Entry 59 : 48, 208, 240
Entry 60 : 48, 240, 0
Entry 61 : 48, 240, 80
Entry 62 : 48, 240, 160
Entry 63 : 48, 240, 240
Entry 64 : 80, 0, 0
Entry 65 : 80, 0, 80
Entry 66 : 80, 0, 160
Entry 67 : 80, 0, 240
Entry 68 : 80, 48, 0
Entry 69 : 80, 48, 80
Entry 70 : 80, 48, 160
Entry 71 : 80, 48, 240
Entry 72 : 80, 80, 0
Entry 73 : 80, 80, 80
Entry 74 : 80, 80, 160
Entry 75 : 80, 80, 240
Entry 76 : 80, 112, 0
Entry 77 : 80, 112, 80
Entry 78 : 80, 112, 160
Entry 79 : 80, 112, 240
Entry 80 : 80, 144, 0

LOOKUP TABLE DATA

Entry 81 :	80,	144,	80
Entry 82 :	80,	144,	160
Entry 83 :	80,	144,	240
Entry 84 :	80,	176,	0
Entry 85 :	80,	176,	80
Entry 86 :	80,	176,	160
Entry 87 :	80,	176,	240
Entry 88 :	80,	208,	0
Entry 89 :	80,	208,	80
Entry 90 :	80,	208,	160
Entry 91 :	80,	208,	240
Entry 92 :	80,	240,	0
Entry 93 :	80,	240,	80
Entry 94 :	80,	240,	160
Entry 95 :	80,	240,	240
Entry 96 :	112,	0,	0
Entry 97 :	112,	0,	80
Entry 98 :	112,	0,	160
Entry 99 :	112,	0,	240
Entry 100 :	112,	48,	0
Entry 101 :	112,	48,	80
Entry 102 :	112,	48,	160
Entry 103 :	112,	48,	240
Entry 104 :	112,	80,	0
Entry 105 :	112,	80,	80
Entry 106 :	112,	80,	160
Entry 107 :	112,	80,	240
Entry 108 :	112,	112,	0
Entry 109 :	112,	112,	80
Entry 110 :	112,	112,	160
Entry 111 :	112,	112,	240
Entry 112 :	112,	144,	0
Entry 113 :	112,	144,	80
Entry 114 :	112,	144,	160
Entry 115 :	112,	144,	240
Entry 116 :	112,	176,	0
Entry 117 :	112,	176,	80
Entry 118 :	112,	176,	160
Entry 119 :	112,	176,	240
Entry 120 :	112,	208,	0
Entry 121 :	112,	208,	80
Entry 122 :	112,	208,	160
Entry 123 :	112,	208,	240
Entry 124 :	112,	240,	0
Entry 125 :	112,	240,	80
Entry 126 :	112,	240,	160
Entry 127 :	112,	240,	240
Entry 128 :	144,	0,	0
Entry 129 :	144,	0,	80
Entry 130 :	144,	0,	160
Entry 131 :	144,	0,	240
Entry 132 :	144,	48,	0
Entry 133 :	144,	48,	80
Entry 134 :	144,	48,	160
Entry 135 :	144,	48,	240
Entry 136 :	144,	80,	0
Entry 137 :	144,	80,	80
Entry 138 :	144,	80,	160
Entry 139 :	144,	80,	240
Entry 140 :	144,	112,	0
Entry 141 :	144,	112,	80
Entry 142 :	144,	112,	160
Entry 143 :	144,	112,	240
Entry 144 :	144,	144,	0
Entry 145 :	144,	144,	80
Entry 146 :	144,	144,	160
Entry 147 :	144,	144,	240
Entry 148 :	144,	176,	0
Entry 149 :	144,	176,	80
Entry 150 :	144,	176,	160
Entry 151 :	144,	176,	240
Entry 152 :	144,	208,	0
Entry 153 :	144,	208,	80
Entry 154 :	144,	208,	160
Entry 155 :	144,	208,	240
Entry 156 :	144,	240,	0
Entry 157 :	144,	240,	80
Entry 158 :	144,	240,	160
Entry 159 :	144,	240,	240
Entry 160 :	176,	0,	0
Entry 161 :	176,	0,	80
Entry 162 :	176,	0,	160
Entry 163 :	176,	0,	240
Entry 164 :	176,	48,	0
Entry 165 :	176,	48,	80
Entry 166 :	176,	48,	160
Entry 167 :	176,	48,	240
Entry 168 :	176,	80,	0
Entry 169 :	176,	80,	80
Entry 170 :	176,	80,	160
Entry 171 :	176,	80,	240
Entry 172 :	176,	112,	0
Entry 173 :	176,	112,	80
Entry 174 :	176,	112,	160

Entry 175 : 176, 112, 240
Entry 176 : 176, 144, 0
Entry 177 : 176, 144, 80
Entry 178 : 176, 144, 160
Entry 179 : 176, 144, 240
Entry 180 : 176, 176, 0
Entry 181 : 176, 176, 80
Entry 182 : 176, 176, 160
Entry 183 : 176, 176, 240
Entry 184 : 176, 208, 0
Entry 185 : 176, 208, 80
Entry 186 : 176, 208, 160
Entry 187 : 176, 208, 240
Entry 188 : 176, 240, 0
Entry 189 : 176, 240, 80
Entry 190 : 176, 240, 160
Entry 191 : 176, 240, 240
Entry 192 : 208, 0, 0
Entry 193 : 208, 0, 80
Entry 194 : 208, 0, 160
Entry 195 : 208, 0, 240
Entry 196 : 208, 48, 0
Entry 197 : 208, 48, 80
Entry 198 : 208, 48, 160
Entry 199 : 208, 48, 240
Entry 200 : 208, 80, 0
Entry 201 : 208, 80, 80
Entry 202 : 208, 80, 160
Entry 203 : 208, 80, 240
Entry 204 : 208, 112, 0
Entry 205 : 208, 112, 80
Entry 206 : 208, 112, 160
Entry 207 : 208, 112, 240
Entry 208 : 208, 144, 0
Entry 209 : 208, 144, 80
Entry 210 : 208, 144, 160
Entry 211 : 208, 144, 240
Entry 212 : 208, 176, 0
Entry 213 : 208, 176, 80
Entry 214 : 208, 176, 160
Entry 215 : 208, 176, 240
Entry 216 : 208, 208, 0
Entry 217 : 208, 208, 80
Entry 218 : 208, 208, 160
Entry 219 : 208, 208, 240
Entry 220 : 208, 240, 0
Entry 221 : 208, 240, 80
Entry 222 : 208, 240, 160
Entry 223 : 208, 240, 240
Entry 224 : 240, 0, 0
Entry 225 : 240, 0, 80
Entry 226 : 240, 0, 160
Entry 227 : 240, 0, 240
Entry 228 : 240, 48, 0
Entry 229 : 240, 48, 80
Entry 230 : 240, 48, 160
Entry 231 : 240, 48, 240
Entry 232 : 240, 80, 0
Entry 233 : 240, 80, 80
Entry 234 : 240, 80, 160
Entry 235 : 240, 80, 240
Entry 236 : 240, 112, 0
Entry 237 : 240, 112, 80
Entry 238 : 240, 112, 160
Entry 239 : 240, 112, 240
Entry 240 : 240, 144, 0
Entry 241 : 240, 144, 80
Entry 242 : 240, 144, 160
Entry 243 : 240, 144, 240
Entry 244 : 240, 176, 0
Entry 245 : 240, 176, 80
Entry 246 : 240, 176, 160
Entry 247 : 240, 176, 240
Entry 248 : 240, 208, 0
Entry 249 : 240, 208, 80
Entry 250 : 240, 208, 160
Entry 251 : 240, 208, 240
Entry 252 : 240, 240, 0
Entry 253 : 240, 240, 80
Entry 254 : 240, 240, 160
Entry 255 : 240, 240, 240

State 5 : red, green, blue intensity

Entry 0 : 0, 0, 0
Entry 1 : 0, 0, 48
Entry 2 : 0, 0, 96
Entry 3 : 0, 0, 144
Entry 4 : 0, 0, 192
Entry 5 : 0, 0, 240
Entry 6 : 0, 48, 0
Entry 7 : 0, 48, 48

LOOKUP TABLE DATA

```
Entry 8 :   0,  48,  96
Entry 9 :   0,  48, 144
Entry 10 :  0,  48, 192
Entry 11 :  0,  48, 240
Entry 12 :  0,  96,   0
Entry 13 :  0,  96,  48
Entry 14 :  0,  96,  96
Entry 15 :  0,  96, 144
Entry 16 :  0,  96, 192
Entry 17 :  0,  96, 240
Entry 18 :  0, 144,   0
Entry 19 :  0, 144,  48
Entry 20 :  0, 144,  96
Entry 21 :  0, 144, 144
Entry 22 :  0, 144, 192
Entry 23 :  0, 144, 240
Entry 24 :  0, 192,   0
Entry 25 :  0, 192,  48
Entry 26 :  0, 192,  96
Entry 27 :  0, 192, 144
Entry 28 :  0, 192, 192
Entry 29 :  0, 192, 240
Entry 30 :  0, 240,   0
Entry 31 :  0, 240,  48
Entry 32 :  0, 240,  96
Entry 33 :  0, 240, 144
Entry 34 :  0, 240, 192
Entry 35 :  0, 240, 240
Entry 36 : 48,   0,   0
Entry 37 : 48,   0,  48
Entry 38 : 48,   0,  96
Entry 39 : 48,   0, 144
Entry 40 : 48,   0, 192
Entry 41 : 48,   0, 240
Entry 42 : 48,  48,   0
Entry 43 : 48,  48,  48
Entry 44 : 48,  48,  96
Entry 45 : 48,  48, 144
Entry 46 : 48,  48, 192
Entry 47 : 48,  48, 240
Entry 48 : 48,  96,   0
Entry 49 : 48,  96,  48
Entry 50 : 48,  96,  96
Entry 51 : 48,  96, 144
Entry 52 : 48,  96, 192
Entry 53 : 48,  96, 240
Entry 54 : 48, 144,   0
Entry 55 : 48, 144,  48
Entry 56 : 48, 144,  96
Entry 57 : 48, 144, 144
Entry 58 : 48, 144, 192
Entry 59 : 48, 144, 240
Entry 60 : 48, 192,   0
Entry 61 : 48, 192,  48
Entry 62 : 48, 192,  96
Entry 63 : 48, 192, 144
Entry 64 : 48, 192, 192
Entry 65 : 48, 192, 240
Entry 66 : 48, 240,   0
Entry 67 : 48, 240,  48
Entry 68 : 48, 240,  96
Entry 69 : 48, 240, 144
Entry 70 : 48, 240, 192
Entry 71 : 48, 240, 240
Entry 72 : 96,   0,   0
Entry 73 : 96,   0,  48
Entry 74 : 96,   0,  96
Entry 75 : 96,   0, 144
Entry 76 : 96,   0, 192
Entry 77 : 96,   0, 240
Entry 78 : 96,  48,   0
Entry 79 : 96,  48,  48
Entry 80 : 96,  48,  96
Entry 81 : 96,  48, 144
Entry 82 : 96,  48, 192
Entry 83 : 96,  48, 240
Entry 84 : 96,  96,   0
Entry 85 : 96,  96,  48
Entry 86 : 96,  96,  96
Entry 87 : 96,  96, 144
Entry 88 : 96,  96, 192
Entry 89 : 96,  96, 240
Entry 90 : 96, 144,   0
Entry 91 : 96, 144,  48
Entry 92 : 96, 144,  96
Entry 93 : 96, 144, 144
Entry 94 : 96, 144, 192
Entry 95 : 96, 144, 240
Entry 96 : 96, 192,   0
Entry 97 : 96, 192,  48
Entry 98 : 96, 192,  96
Entry 99 : 96, 192, 144
Entry 100 :  96, 192, 192
Entry 101 :  96, 192, 240
```

Entry 102 : 96, 240, 0
Entry 103 : 96, 240, 48
Entry 104 : 96, 240, 96
Entry 105 : 96, 240, 144
Entry 106 : 96, 240, 192
Entry 107 : 96, 240, 240
Entry 108 : 144, 0, 0
Entry 109 : 144, 0, 48
Entry 110 : 144, 0, 96
Entry 111 : 144, 0, 144
Entry 112 : 144, 0, 192
Entry 113 : 144, 0, 240
Entry 114 : 144, 48, 0
Entry 115 : 144, 48, 48
Entry 116 : 144, 48, 96
Entry 117 : 144, 48, 144
Entry 118 : 144, 48, 192
Entry 119 : 144, 48, 240
Entry 120 : 144, 96, 0
Entry 121 : 144, 96, 48
Entry 122 : 144, 96, 96
Entry 123 : 144, 96, 144
Entry 124 : 144, 96, 192
Entry 125 : 144, 96, 240
Entry 126 : 144, 144, 0
Entry 127 : 144, 144, 48
Entry 128 : 144, 144, 96
Entry 129 : 144, 144, 144
Entry 130 : 144, 144, 192
Entry 131 : 144, 144, 240
Entry 132 : 144, 192, 0
Entry 133 : 144, 192, 48
Entry 134 : 144, 192, 96
Entry 135 : 144, 192, 144
Entry 136 : 144, 192, 192
Entry 137 : 144, 192, 240
Entry 138 : 144, 240, 0
Entry 139 : 144, 240, 48
Entry 140 : 144, 240, 96
Entry 141 : 144, 240, 144
Entry 142 : 144, 240, 192
Entry 143 : 144, 240, 240
Entry 144 : 192, 0, 0
Entry 145 : 192, 0, 48
Entry 146 : 192, 0, 96
Entry 147 : 192, 0, 144
Entry 148 : 192, 0, 192
Entry 149 : 192, 0, 240
Entry 150 : 192, 48, 0
Entry 151 : 192, 48, 48
Entry 152 : 192, 48, 96
Entry 153 : 192, 48, 144
Entry 154 : 192, 48, 192
Entry 155 : 192, 48, 240
Entry 156 : 192, 96, 0
Entry 157 : 192, 96, 48
Entry 158 : 192, 96, 96
Entry 159 : 192, 96, 144
Entry 160 : 192, 96, 192
Entry 161 : 192, 96, 240
Entry 162 : 192, 144, 0
Entry 163 : 192, 144, 48
Entry 164 : 192, 144, 96
Entry 165 : 192, 144, 144
Entry 166 : 192, 144, 192
Entry 167 : 192, 144, 240
Entry 168 : 192, 192, 0
Entry 169 : 192, 192, 48
Entry 170 : 192, 192, 96
Entry 171 : 192, 192, 144
Entry 172 : 192, 192, 192
Entry 173 : 192, 192, 240
Entry 174 : 192, 240, 0
Entry 175 : 192, 240, 48
Entry 176 : 192, 240, 96
Entry 177 : 192, 240, 144
Entry 178 : 192, 240, 192
Entry 179 : 192, 240, 240
Entry 180 : 240, 0, 0
Entry 181 : 240, 0, 48
Entry 182 : 240, 0, 96
Entry 183 : 240, 0, 144
Entry 184 : 240, 0, 192
Entry 185 : 240, 0, 240
Entry 186 : 240, 48, 0
Entry 187 : 240, 48, 48
Entry 188 : 240, 48, 96
Entry 189 : 240, 48, 144
Entry 190 : 240, 48, 192
Entry 191 : 240, 48, 240
Entry 192 : 240, 96, 0
Entry 193 : 240, 96, 48
Entry 194 : 240, 96, 96
Entry 195 : 240, 96, 144

LOOKUP TABLE DATA

Entry 196	:	240,	96,	192
Entry 197	:	240,	96,	240
Entry 198	:	240,	144,	0
Entry 199	:	240,	144,	48
Entry 200	:	240,	144,	96
Entry 201	:	240,	144,	144
Entry 202	:	240,	144,	192
Entry 203	:	240,	144,	240
Entry 204	:	240,	192,	0
Entry 205	:	240,	192,	48
Entry 206	:	240,	192,	96
Entry 207	:	240,	192,	144
Entry 208	:	240,	192,	192
Entry 209	:	240,	192,	240
Entry 210	:	240,	240,	0
Entry 211	:	240,	240,	48
Entry 212	:	240,	240,	96
Entry 213	:	240,	240,	144
Entry 214	:	240,	240,	192
Entry 215	:	240,	240,	240
Entry 216	:	0,	0,	0
Entry 217	:	0,	0,	0
Entry 218	:	0,	0,	0
Entry 219	:	0,	0,	0
Entry 220	:	0,	0,	0
Entry 221	:	0,	0,	0
Entry 222	:	0,	0,	0
Entry 223	:	0,	0,	0
Entry 224	:	0,	0,	0
Entry 225	:	0,	0,	0
Entry 226	:	0,	0,	0
Entry 227	:	0,	0,	0
Entry 228	:	0,	0,	0
Entry 229	:	0,	0,	0
Entry 230	:	0,	0,	0
Entry 231	:	0,	0,	0
Entry 232	:	0,	0,	0
Entry 233	:	0,	0,	0
Entry 234	:	0,	0,	0
Entry 235	:	0,	0,	0
Entry 236	:	0,	0,	0
Entry 237	:	0,	0,	0
Entry 238	:	0,	0,	0
Entry 239	:	0,	0,	0
Entry 240	:	0,	0,	0
Entry 241	:	0,	0,	0
Entry 242	:	0,	0,	0
Entry 243	:	0,	0,	0
Entry 244	:	0,	0,	0
Entry 245	:	0,	0,	0
Entry 246	:	0,	0,	0
Entry 247	:	0,	0,	0
Entry 248	:	0,	0,	0
Entry 249	:	0,	0,	0
Entry 250	:	0,	0,	0
Entry 251	:	0,	0,	0
Entry 252	:	0,	0,	0
Entry 253	:	0,	0,	0
Entry 254	:	0,	0,	0
Entry 255	:	0,	0,	0

Appendix F

Diagnostics and LED's

F.1 Diagnostic Programme

A set of diagnostics programmes are provided with the PG-640A to allow the user to perform some preliminary testing of the board set in the unlikely event of a hardware error. These tests are menu driven and expect the user to answer each time regarding whether or not the display is correct. If a hardware error is detected the user should get in contact with the Applications Engineering Department of MATROX in order to determine what procedure should be followed.

F.1.1 Main Menu

The main menu displays the following information:

```
0. test for CGA emulator
1. test for high level graphic
2. self test
```

```
3. exit to DOS
```

Each of the menu choices is self explanatory. The instructions on the screen should be followed. The remainder of this Appendix lists each sub-menu and gives a brief description of what the user should expect.

F.1.2 CGA Emulator Test

The menu for the CGA Emulator tests has the following choices:

```
0. Emulator test equal spacing
1. Emulator blank display
2. Emulator checker board
3. Emulator cursor display
4. Emulator 40 x 25 display
5. Emulator display attributes
6. Emulator character set
7. Emulator 80 x 25 display
8. Emulator screen paging
9. Emulator 320 x 200 graphics
10. Emulator 640 x 200 graphics
11. Emulator video colour
12. Emulator very fast mode
13. Run all
14. Exit to main menu
```

The user should expect to see the following for each test:

Equal Spacing First a screen with equally spaced vertical bars, then a screen with equally spaced horizontal bars.

Blank Display A screen that is blank exept for instructions.

Checker Board A screen containing a checker board pattern.

Cursor Display A box in the centre of the screen containing a blinking cursor. First the underscore cursor will be displayed, then a block cursor.

40 × 25 Display A 40 column display of the standard characters in a barber pole pattern.

Display Attributes A set of lines of text in the various type modes.

Character Set The full character set is displayed.

80 × 25 Display A 80 column display of the standard characters in a barber pole pattern.

Screen Paging Each of the eight graphics pages are displayed.

320 × 200 Graphic Two screens are displayed, each having three different coloured boxes.

640 × 200 Graphic A screen is displayed with three white boxes.

Video Colour Sixteen screens are displayed, each filled with a different colour.

Very Fast Mode The screen will flash and then clear.

F.1.3 High Level Graphics Test

The menu for the high level graphics diagnostics contains the following choices:

```
0. PG Display
1. PG Bit Planes
2. PG Video RAM ACRTC Access
3. PG Video RAM CPU Access
4. PG Colour Grid
5. PG Colour Shading
6. PG LUT Fast Change
```

```
7. PG Blink
8. PG DMA
9. Run all
10.Exit to main menu
```

The user should expect to see the following for each test:

PG Display Four sentences are displayed, one in each of red, blue, green, and white.

PG Bit Planes Eight overlaping boxes are displayed, one for each bit plane.

PG Video RAM ACTRC Access This is a self contained test, if an error occurs, an error message will be displayed.

PG Video RAM CPU Access The screen is filled with red.

PG Colour Grid The six LUT's are displayed, 256 squares of different colours, arranged 16 by 16 will be displayed with the following patterns:

colours get progressively brighter from left to right;

colours get progressively brighter from top to bottom;

colours get progressively brighter from top to bottom twice;

colours get progressively brighter from top to bottom twice;

colours get progressively brighter from top to bottom four times;

colours are arranged randomly.

PG Colour Shading Four lines, white; blue; green; and red, of sixteen boxes are displayed. The boxes get brighter from left to right.

PG LUT Fast Change The same display as previous, but the boxes shift rapidly from right to left.

PG Blink Four filled squares are displayed that blink at different rates.

PG DMA This test will copy an image from the PC to the PG-640A and then an image from the PG-640A to the PC. If this test fails, ensure that the PG-640A is configured with DMA enabled on Channel 2 before assuming a hardware error has occured.

F.1.4 Self Test

This test will ask the user to change the settings on the block of four DIP switches. Internal tests will be performed and the user asked to reset the DIP switches. The test will terminate with messages stating that the areas test is functioning correctly.

F.2 LED's

There are four LED's on the PG-640A board set, three of which provide information about the board's status. The LED's are:

1. Heartbeat: the light blinks on and off to tell the user that the board is functioning properly.
2. Output of Error FIFO Full: this LED lights up when either of the read back FIFO's are full. The board will wait for space in a full read back FIFO before processing further.
3. Input FIFO Empty: this LED lights up when the Input FIFO is empty.
4. RESERVED: this LED is for MATROX use only.

Appendix G

Diskette Directory

The PG-640A is supplied with two diskettes (in the back of the manual). This appendix consists of directories for those diskettes, the contents of their READ.ME files, plus other pertinent information that will help the user to exploit the diskettes.

G.1 Directory

G.1.1 Directory of Utilities Diskette

SHOWLUT	EXE
DI	EXE
SELFTEST	EXE
PGRESET	EXE
TOPG	EXE
PGMON	EXE
PGTOFILE	EXE
FILETOPG	EXE
DIAG	EXE
OTTAWA2	PGH
VDIPG	SYS
INVASM	EXE
HOUSE	PGA
DEMO	BAT
READ	ME
3DCITY	PGH
PROCESS	PGH
MARQUIS	PGH

G.1.2 Directory of Demo Diskette

DEMOEND	PGH
TEXT10	PGH
TEXT20	PGH
TEXT40	PGH
RECTPF0	PGH
RECTPF1	PGH
CIPF0	PGH
CIPF1	PGH
SEEDS	PGH
SEEDP	PGH
POLYPF0	PGH
ELPF0	PGH
SECTPF0	PGH
ARC	PGH
POINT	PGH
LINEH	PGH
LINEV	PGH
HOUSE	PGH
3D	PGH
CLOCK	PGH
HEAD3D	PGH
BLOCK	PGH
PROCESS	PGH
LINES	PGH

MARQUIS	PGH
PAGE1	PGH
PAGE2	PGH
PAGE3	PGH
PAGE4	PGH
PAGE5	PGH
PAGE6	PGH
PRIM	PGH
3DCITY	PGH
WAIT5	PGH
ALLSAT	PGA
HD	BAT
COMP	BAT
DEMO	BAT
DEMOL	BAT
DEMOCOMP	BAT
MAYFLOWE	SCH
READ	ME
TOPG	EXE
DI	EXE
PGRESET	EXE

G.2 Read.Me Files

G.2.1 Utility Diskette Read.Me File

The following programs and data files are supplied by MATROX to give the PG-640A user and programmer a starting point for writing code. These programs are not supported by MATROX.

pgmon.exe - Interactive program to send PG-640A high level graphic commands

use: A⟩pgmon [-a] the optional -a flag must be used if the board uses the alternate address, C64000, rather than the default address, C6000.

a menu shall come up with the following options

F1	- send file	F2	- addr C6000/C6400
F3	- cold reset	F4	- warm reset
F5	- TXTWDW off	F6	- TXTWDW on
F7	- CGA display	F8	- High-res graphics display
F9	- ASCII/HEX input	F10	- ASCII/HEX output

Exit program with ^C

NB: do not use the F2 option unless there is a PG-640A at both the default and the alternate address

topg.exe - Program to send the PG-640A a file containing hex or ascii commands

use: A⟩topg [-a] house.pga

send the file house.pga to the board the optional -a flag must be used if the PG-640A is at the alternate address

di.exe - Program to switch between high level graphocs screen and CGA screen

use: A⟩di [-a] 0 (enable high level graphics screen) A⟩di [-a] 1 (enable CGA screen)

use the -a flag if the PG-640A uses the alternate address

pgtofile.exe - Program to send a raster image of the high resolution display to disk file

use: A⟩pgtofile [-a] test1.dat

stores the current screen image in the file test1.dat use the -a flag if the PG-640A uses the alternate address

filetopg.exe - Program to send raster image from a file to the PG-640A high resolution display

use: A⟩filetopg [-a] test1.dat

displays the raster image stored in test1.dat use the -a flag if the PG-640A uses the alternate address

DISKETTE DIRECTORY

selftest.exe - PG-640A selftest program (see appendix F sec. F.1.4)

use: A)selftest [-a]

use the -a flag if the PG-640A uses the alternate address

showlut.exe - Program to display various predefined lookup tables on PG-640A see LUTINT command

use: A)showlut [-a]

use the -a flag if the PG-640A uses the alternate address

pgreset.exe - causes a cold reset of the PG-640A.

use: A)pgreset [-a]

use the -a flag if the PG-640A uses the alternate address

invasm.exe - A file that will convert ascii graphic commands into into binary code, or convert binary code back to ascii graphics code.

use: a) invasm -o[h,a] -b[h,a] -f[s,l] infile outfile

example ; a) invasm -oa -bh file.pgh file.pga

will take a binary file (file.pgh) as the input and will output an ascii file (file.pga)

flag options:

-bx :	x = a : begin translation with comm type in ASCII (default).
	x = h : begin translation with comm type in HEX.
-ox :	x = a : output in ascii.
	x = h : output in binary (default).
-fx :	x = s : short form ascii opcode output.
	x = l : long form ascii opcode output (default).
-hx :	x = x : binary hex output (default).
	x = 2 : ASCII hex output.

diag.com - PG-640A diagnostic program (see appendix F)

use: A)diag

house.pga - PG-640A ASCII data file of 3D house used in chapter 3.4 of PG-640A manual

ottawa2.pgh - Data file used during dma diagnostic of PG-640A

3dcity.pgh - 3d demonstration file

process.pgh - process control example file

marquis.pgh - demonstration file

demo.bat [-a] - batch file to provide a short demonstration

- use the -a flag if the PG-640A uses the alternate address

vdipg.sys - Matrox VDI driver(see Appendix H)

G.2.2 Demo Diskette Read.Me File

This diskette contains demonstration programs and picture files used in the demos.

demo.bat - Run the standard demo once.

- use -a flag if PG-640A at alternate address

demol.bat - Run a continous loop of the standard demo.

- use -a flag if PG-640A at alternate address

comp.bat - Run PG-640A speed comparison demo.

- use -a flag if PG-640A at alternate address

democomp.bat - Run standard demo followed by speed comparison demo

- use -a flag if PG-640A at alternate address

hd.bat - Install demo onto hard disc

topg.exe - Program to send a picture file to the PG-640A

- use -a flag if PG-640A at alternate address

pgreset.exe - Program to reset the PG-640A

- use -a flag if PG-640A at alternate address

DISKETTE DIRECTORY

di.exe - Program to select between High resolution mode and CGA mode on the PG-640A

- use -a flag if PG-640A at alternate address

****.pgh*** - Picture files (Note that this rel includes an improved version of 3DCITY.PGH)

****.pga*** - Picture files

****.sch*** - Picture files

Appendix H

Installing The PG-640A Device Driver

H.1 Introduction

This appendix explains how to install PG-640A VDI Device Driver and summarizes the VDI opcodes that it supports.

We assume that you already have VDI software installed in your system and are familiar with it. If this is not the case, you will need to obtain it. It may be purchased from either Graphic Software Systems of Wilsonville Oregon or IBM. For more detailed information on the VDI please refer to the Professional Graphics Series manuals from IBM.

H.2 Installation

Use the following procedure to install the PG-640A Device Driver and to initialize the VDI.

1. The PG-640A Device Driver is the file VDIPG.SYS on the utilities diskette supplied with the PG-640A. Your first step should be to find this diskette and make a back up copy of it. Use the DIR command to confirm that you have the correct diskette, and use the COPY command to make the backup copy. Store the original diskette in a safe place and use the backup copy for the next step in this procedure.

2. Use the DOS COPY command to copy the VDIPG.SYS file to your system disk (the diskette or Winchester with the operating system and other device drivers). You may copy it to either a root directory or a subdirectory.

3. Using EDLIN or a similar editor, add lines with the following format to the end of your CONFIG.SYS file. The CONFIG.SYS file should already be present on your system disk:

   DEVICE=C:[path] VDIPG.SYS [/R]
   DEVICE=C:[path]VDI.SYS [/G][:group name]

   For example:

   DEVICE=C:\GSS\DRIVERS\VDIPG.SYS
   DEVICE=C:\GSS\DRIVERS\VDI.SYS

   Note: (1) It is important that the VDI.SYS file be listed after all of the device drivers have been listed.

   (2) For more information on the command format see the Graphic Development Toolkit Manual from Graphic Software Systems or IBM.

4. Add a line with the folowing format to your AUTOEXEC.BAT file:

   [path]INIT_VDI

   For example:

   GSS\DRIVERS\INIT_VDI

5. Verify that all the files are where they should be, then reset the system to initialize the driver. The DOS will find CONFIG.SYS and use the information therein to configure the system. Then it will process the AUTOEXEC.BAT file and, in so doing, execute the init_vdi command, which initializes the VDI.

H.3 VDI Opcodes

This section lists the VDI commands supported by the PG-640A and its device driver.

Control

- Clear Workstation
- Close Workstation
- Cursor Down
- Cursor Left
- Cursor Right
- Cursor Up
- Direct Cursor Address
- Enter Cursor Addressing Mode
- Erase to End of Line
- Erase to End of Screen
- Exit Cursor Addressing Mode
- Home Cursor
- Open Workstation

- **Display Graphic Input Cursor**
- **Remove Graphic Input Cursor**
- **Set Alpha Text Position**
- **Set Line Edit Characters**
- **Update Workstation - No action is performed**
- **Set Writing Mode - Only the following writing modes are supported:**

Mode	Boolean Operation Chart
1	D = 0 (color 0)
2	D = D AND S (AND)
4	D = S (Replace)
5	D = D AND (NOT S)
6	D = D (no change)
7	D = D XOR S (exclusive OR)
8	D = D OR S (overstrike)
11	D = NOT D
13	D = NOT S
14	D= D OR (NOT S)
16	D = 1 (color 255)

Output Primitives

- **Arc (uses polyline)**
- **Bar (uses filled area attributes)**
- **Cell Array**
- **Circle (uses filled area attributes)**
- **Filled Area**
- **Graphic Text**

- Output Alpha Text
- Output Cursor Addressable Text
- Pie Slice (uses filled area attributes)
- Polyline
- Polymarker

Output Attributes

- Reverse Video Off
- Reverse Video On
- Set Alpha Text Color
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode - Returns default value
- Set Alpha Text Quality - Returns default value
- Set Alpha Text Script Mode
- Set Alpha Text Underline Mode
- Set Background Color Index
- Set Character Baseline Rotation
- Set Character Height
- Set Color Representation - Returns default settings
- Set Cursor Text Attributes

- Set Fill Color Index
- Set Fill Interior style
- Set Fill Style Index
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font - Returns default setting
- Set Polyline Color Index
- Set Polyline Type
- Set Polyline Width - Returns default setting
- Set Polymarker Type
- Set Polymarker Scale
- Set Polymarker Color Index

Input

- Input Locator - Request Mode
- Input Choice - Request Mode
- Input String - Request Mode
- Input String - Sample Mode
- Read Cursor Movement Keys

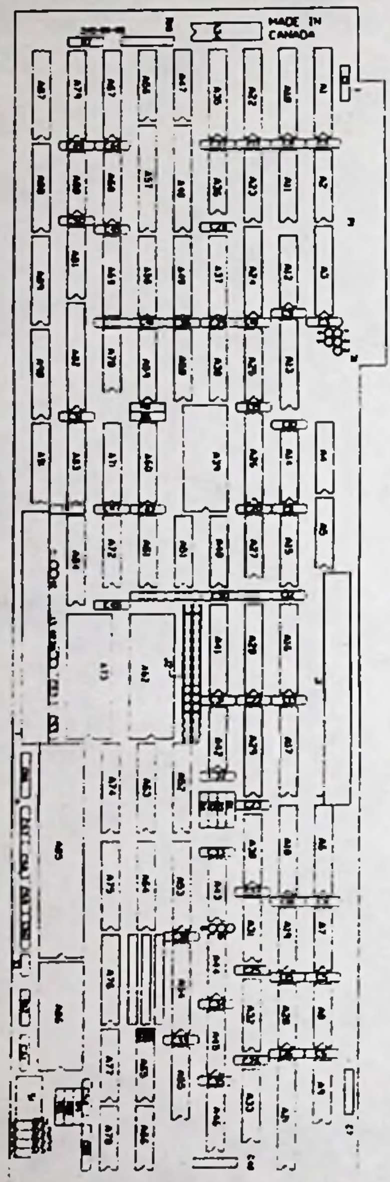
Inquiries

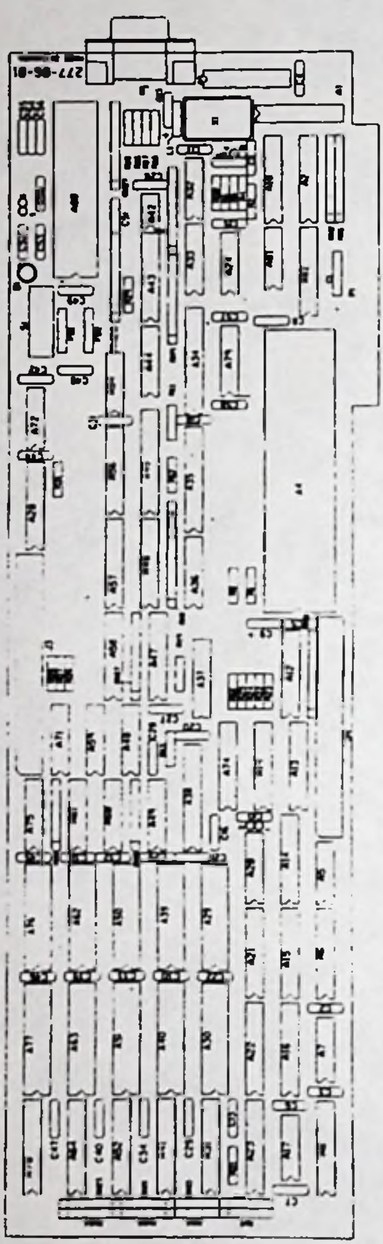
- Inquire Alpha Text Capabilities
- Inquire Alpha Cell Location
- Inquire Alpha Font Availability
- Inquire Alpha Text Position
- Inquire String Extent
- Inquire Addressable Character Cells
- Inquire Color Representation
- Inquire Current Cursor Address
- Inquire Current Fill Area Attributes
- Inquire Current Graphic Text Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Inquire Cell Array

Appendix I

Board Layout

BOARD LAYOUT



CPU Board Components

Video Board Components

BOARD LAYOUT

Appendix J

Fast Execution "Local Pipes"

This chapter describes the fast execution families of graphic commands, optimized to work together as a group, in the firmware for the PG-640A. These families of graphic commands use local command decoders to offer greatly increased command decoding speed. Section J.1 explains the concept of "local pipes" and Section J.2 describes the "local pipe" Command Sets.

J.1 Description of Local Pipes

The PG-640A contains fast execution "local pipes" in its firmware. The term "pipe" is used here to describe a subset of the board's full set of graphic commands which has been optimized to work as a group. Special areas of the firmware contain local command decoders which bypass the normal, lengthy highlevel decoding overhead. These local command decoders are, therefore, capable of decoding a small, fixed number of commands very quickly. If only graphic commands which are part of the local pipe's command set are issued to the board, decoding stays within the pipe and executes much faster than would normally be possible.

Entry to a local pipe is automatically achieved by sending the PG-640A one of a local pipe's *Entry Point Commands*. As soon as a command outside of the local pipe's command set is issued to the board, the local pipe is exited and decoding of commands through the highlevel command decoder resumes.

NOTE:
— Local pipes are accessed through *Entry Point Commands* only. — Commands in a local pipe's command set are not all *Entry Point Commands*. — Certain local pipes are not accessible from command lists. — No local pipes are accessible from ASCII input mode.

J.2 Local Pipe Command Set Descriptions

Screen Coordinate Drawing Command Pipe

Command Set:

SMOVE $^{En.}$

SMOVER $^{En.}$

SDRAW $^{En.}$

SDRAWR $^{En.}$

COLOR

$^{En.}$ denotes an *Entry Point Command* in the Pipe Command Set.

Access from:

Hex Input Mode

Command Lists

User Definable Raster Text Command Pipe

Command Set:

TEXTP *En.*

TEXTPC *En.*

SMOVE †

SMOVER †

COLOR

BCOLOR

RFONT

En. denotes an *Entry Point Command* in the Pipe Command Set.
† denotes an *Entry Point Command* for the Screen Coordinate Drawing Command Pipe.

Access from:

Hex Input Mode only

NOTE:

The User Definable Raster Text Command Pipe is a two-level local pipe in which two of the commands in the command set, SMOVE and SMOVER, are also part of the Screen Coordinate Drawing Command Pipe. The following shows the process flow when either one of these commands is invoked. Note the two-level pipelining in Example 1.

COMMAND	COURSE OF ACTION
	Example 1
TEXTP	Enters User Definable Raster Text Command Pipe.
SMOVE	Enters Screen Coordinate Drawing Command Pipe.
SDRAW	Remains in Screen Coordinate Drawing Command Pipe.
TEXTP	Exits back to User Definable Raster Text Command Pipe.
	Example 2
TEXTP	Enters User Definable Raster Text Command Pipe.
SMOVE	Enters Screen Coordinate Drawing Command Pipe.
TEXTP	Exits back to User Definable Raster Text Command Pipe.

Any number of the commands from the Screen Coordinate Drawing Command Pipe command set may be used directly following the SMOVE or SMOVER commands. Once the flow has exited the Screen Coordinate Drawing Command Pipe, invoking any of the Screen commands will cause the program to exit the User Definable Raster Text Command Pipe and return to highlevel command decoding.

FAST EXECUTION "LOCAL PIPES"

World Coordinate 2D Drawing Command Pipe

Command Set:

MOVE [En.]

MOVER [En.]

DRAW [En.]

DRAWR [En.]

COLOR

[En.] denotes an *Entry Point Command* in the Pipe Command Set.

Access from:

Hex Input Mode

Command Lists

World Coordinate 3D Drawing Command Pipe

Command Set:

MOVES3 *En.*

MOVER3 *En.*

DRAWS3 *En.*

DRAWR3 *En.*

En. denotes an *Entry Point Command* in the Pipe Command Set.

Access from:

Hex Input Mode

Command Lists

IMAGEW Command Pipe

Command Set:

IMAGEW *En.*

En. denotes an *Entry Point Command* in the Pipe Command Set.

Access from:

Hex Input Mode only

PDRAW Command Pipe

Command Set:

PDRAW *En.*

COLOR

NOP

En. denotes an *Entry Point Command* in the Pipe Command Set.

Access from:

Hex Input Mode

Command Lists

Appendix K

Command Reference Card

The following page contains two summaries of commands — one arranged by name, the other by hex opcode. These summaries are just that, summaries. For complete command descriptions refer to Chapter 4.

K.1 Commands by Name

Name	Opcode	Name	Opcode	Name	Opcode
ARC	3C	LUT	EE	SARC	F4
AREA	C0	LUTINT	EC	SBLINK	E4
AREABC	C1	LUTRD	50	SCIRC	F2
AREAPT	E7	LUTSAV	ED	SDRAW	FA
BCOLOR	CB	LUTSTO	C9	SDRAWR	FB
BLINK	C8	LUTX	E6	SECTOR	3D
BLINKX	E5	LUTXRD	53	SELIPS	F3
CA	43 41 20	MASK	E8	SMOVE	F8
CIRCLE	38	MATXRD	52	SMOVER	F9
CLBEG	70	MDIDEN	90	SPOLY	FC
CLDEL	74	MDMATX	97	SPOLYR	FD
CLEARS	0F	MDORG	91	SRECT	F0
CLEND	71	MDROTX	93	SRECTR	F1
CLIPH	AA	MDROTY	94	SSECT	F5
CLIPY	AB	MDROTZ	95	TANGLE	82
CLOOP	73	MDSCAL	92	TASPCT	8B
CLMOD	78	MDTRAN	96	TCHROT	8A
CLRD	76	MOVE	10	TDEFIN	84
CLRUN	72	MOVER	11	TEXT	80
COLMOD	CA	MOVE3	12	TEXTC	8C
COLOR	06	MOVER3	13	TEXTP	83
CONVRT	AF	NOOP	01	TEXTPC	8D
CX	43 58 20	PDRAW	FF	TJUST	85
DISPLA	D0	POINT	08	TSIZE	81
DISTAN	B1	POINT3	09	TSTYLE	88
DISTH	A8	POLY	30	TWCOL	D5
DISTY	A9	POLYR	31	TWPOS	D3
DRAW	28	POLY3	32	TWVIS	D4
DRAWR	29	POLYR3	33	VWIDEN	A0
DRAW3	2A	PRMFIL	E9	VWMATX	A7
DRAWR3	2B	PROJCT	B0	VWPORT	B2
ELIPSE	39	RASTOP	DA	VWROTX	A3
FILMSK	EF	RASTRD	DB	VWROTY	A4
FLAGRD	51	RASTWR	DC	VWROTZ	A5
FLOOD	07	RBAND	E1	VWRPT	A1
GTDEF	89	RDEFIN	54	WAIT	05
IMAGER	D8	RFONT	55	WINDOW	B3
IMAGEW	D9	RECT	34	XHAIR	E2
LINFUN	EB	RECTR	35	XMOVE	E3
LINPAT	EA	RESETF	04		

K.2 Commands by Hex Opcode

Opcode	Name	Opcode	Name	Opcode	Name
01	NOOP	78	CLMOD	CA	COLMOD
04	RESETF	80	TEXT	CB	BCOLOR
05	WAIT	81	TSIZE	D0	DISPLA
06	COLOR	82	TANGLE	D3	TWPOS
07	FLOOD	83	TEXTP	D4	TWVIS
08	POINT	84	TDEFIN	D5	TWCOL
09	POINT3	85	TJUST	D8	IMAGER
0F	CLEARS	88	TSTYLE	D9	IMAGEW
10	MOVE	89	GTDEF	DA	RASTOP
11	MOVER	8A	TCHROT	DB	RASTRD
12	MOVE3	8B	TASPCT	DC	RASTWR
13	MOVER3	8C	TEXTC	E1	RBAND
28	DRAW	8D	TEXTPC	E2	XHAIR
29	DRAWR	90	MDIDEN	E3	XMOVE
2A	DRAW3	91	MDORG	E4	SBLINK
2B	DRAWR3	92	MDSCAL	E5	BLINKX
30	POLY	93	MDROTX	E6	LUTX
31	POLYR	94	MDROTY	E7	AREAPT
32	POLY3	95	MDROTZ	E8	MASK
33	POLYR3	96	MDTRAN	E9	PRMFIL
34	RECT	97	MDMATX	EA	LINPAT
35	RECTR	A0	VWIDEN	EB	LINFUN
38	CIRCLE	A1	VWRPT	EC	LUTINT
39	ELIPSE	A3	VWROTX	ED	LUTSAV
3C	ARC	A4	VWROTY	EE	LUT
3D	SECTOR	A5	VWROTZ	EF	FILMSK
43 41 20	CA	A7	VWMATX	F0	SRECT
43 58 20	CX	A8	DISTH	F1	SRECTR
50	LUTRD	A9	DISTY	F2	SCIRC
51	FLAGRD	AA	CLIPH	F3	SELIPS
52	MATXRD	AB	CLIPY	F4	SARC
53	LUTXRD	AF	CONVRT	F5	SSECT
54	RDEFIN	B0	PROJCT	F8	SMOVE
55	RFONT	B1	DISTAN	F9	SMOVER
70	CLBEG	B2	VWPORT	FA	SDRAW
71	CLEND	B3	WINDOW	FB	SDRAWR
72	CLRUN	C0	AREA	FC	SPOLY
73	CLOOP	C1	AREABC	FD	SPOLYR
74	CLDEL	C8	BLINK	FF	PDRAW
75	CLRD	C9	LUTSTO		

PRODUCT FAILURE REPORT

If you are returning one of our products for repair, you must fill out this form and return it with the defective unit. The information so provided is necessary for us to provide a high standard of service.

COMPANY NAME AND ADDRESS: ______________________________

____

NAME OF UNIT: ______________________________
MODEL NO.(on silkscreen): ______________________________
SERIAL NO.(on label): ______________________________
DATE UNIT RECEIVED: ____________ DATE UNIT FAILED: ____________
OR DEAD ON ARRIVAL ☐

MEMORY BASE ADDRESS USED: ______________________________
I/O BASE ADDRESS USED: ______________________________
PLEASE DESCRIBE THE SYSTEM THAT THE UNIT IS USED IN (CPU, BUS, MEMORY, ETC.): ______________________________

____

____

____

UNIT CONFIGURATION (50 or 60 Hz, attributes used, display resolution selected, etc.): ______________________________

____

____

PLEASE DESCRIBE THE FAULT: ______________________________

____

____

____

FAULT IS CONSTANT ☐ FAULT IS INTERMITTENT ☐

NOTE: No merchandise will be accepted by MATROX for replacement or repair unless accompanied by an RMA number obtained from our Application Engineering Department.
RMA Number: ______________________________

THE FOLLOWING SPACE IS FOR FACTORY USE ONLY

CORRECTIVE STEPS TAKEN: ______________________________

____

____

____

____

MATROX Electronic Systems Limited,
1055 St. Regis Boulevard,
Dorval, Quebec,
CANADA H9P 2T4
Telephone: (514)685-2630 Telex: 05-822798 FAX: (514)685-2853

matrox
electronic systems ltd.

1055 ST. RÉGIS BLVD., DORVAL, QUÉBEC H9P 2T4, CANADA
TEL.: (514) 685-2630 TELEX: 05-822798