SIRIUS 1

GRAFIX

sirius™

SIRIUS GRAFIX

MANUAL AND SPECIFICATIONS

Zeroth Edition, April 1982

Michael M. Cirovic

SIRIUS GRAFIX

This manual describes the Sirius Grafix Kernel. The kernel is intended to provide a user-friendly interface to the Sirius One Computer and allow for simple utilization of the High Resolution capability of the Computer. This is a preliminary copy - any suggestions for improvement are welcomed.

April 13, 1982
Michael M. Cirovic
Director, R&D Division
Sirius Systems Technology
864 Osos Street, Suite C
San Luis Obispo, California 93401

USING HIGH RESOLUTION GRAPHICS

Introduction

The high resolution screen is made up of three hundred twenty thousand (320,000) individual dots. These dots are arranged 800 per horizontal row with 400 rows vertically. The basic idea is shown in Figure 1. The coordinates of the screen are as follows: the origin, x = 0, y = 0, is at the upper left hand corner of the screen. The upper right hand corner of the screen is specified by the x value of 799 and the y value of 0; the lower left hand corner of the screen has an x value of 0 and a y value of 399; the lower right hand corner of the screen has an x value of 799 and a y value of 399.

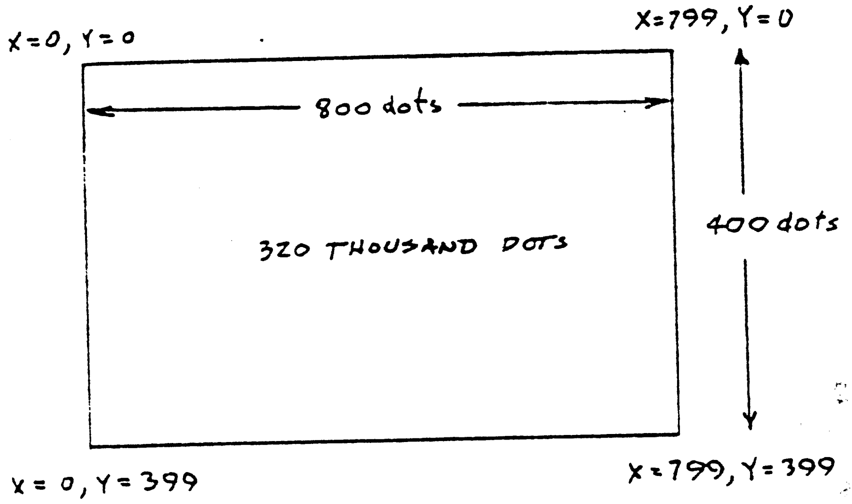


FIGURE 1 - THE SCREEN

The screen has an aspect ratio of 3 to 2 where the aspect ratio is defined as the ratio of x units to y units to provide equal length. Figure 2 shows graphically the basic idea of the aspect ratio. Note that in order to define a block of equal size in the x and y direction three dots are necessary in the x direction and only 2 dots in the y direction. As a consequence if one wished to draw a square of 10 units on the screen one would have to draw a horizontal line of 15 units in the x direction, a vertical line of 10 units in the y direction, a second horizontal line of 15 units in the x direction and finally a vertical line of 10 units in the y direction. Also note that in order to display a circle on the screen in reality a mathematical elipse has to be drawn.

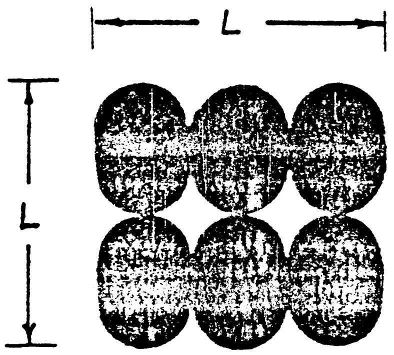


FIGURE 2 - THE ASPECT RATIO

Overview

The Grafix Package allows the user up to 10 different character sets of 128 characters each and up to eight full screens. (One screen uses exactly 40,000 bytes; a character set uses slightly over 4k bytes). The specific number of screens available to the user is determined by the total amount of available memory in the system. The following table gives some examples assuming four character sets are used. Note that in a system with 512k bytes of RAM, all ten character sets and all eight screens may be used leaving a user RAM in excess of 128k. For smaller RAM configurations the user can limit himself to the appropriate number of screens and character sets. (The approximate values given in Table I are based on the following: The system requires approximately 30k bytes, the Grafix module requires approximately 10k and four character sets require approximately 16k). In terms of using the screens it must be realized that the Screen 0, enabled by default, is a viewable screen; that is that set of 320,000 dots is displayed on the CRT. Screen 1 may also be selected as a viewable screen. These are the only two screens that the current hardware can display. Any other screens that are defined may be plotted and written to and then transferred to either Screen 0 or 1 to be viewed.

TABLE I

APPROXIMATE USER SPACE WITH GRAPHICS

No. of	SYSTEM RAM			
Screens	128k	256k	384k	512k
1	32k	180k	288k	416k
2		141k	249k	377k
3		102k	210k	338k
4		63k	171k	299k
5		23k	132k	260k
6			93k	221k
7			43k	182k
8			14k	143k

Note that the values listed in Table I represent the remaining RAM and in a given application the user must make certain that there is sufficient RAM left for his program as well as his run time package, or interpreter, depending on the language used. Also note that the Grafix package may be used together with any language that is capable of executing a print statement.

Startup

When enabling the Grafix Package the user must pass two parameters. These are the maximum number of character sets that can be in use at any one time, and the maximum number of screens that the user wishes to enable. The default minimum number of character sets available is two. These are the system character sets consisting of the first 128 ASCII characters and the upper 128 graphics characters; neither of these two sets are ever overwritten. The maximum number of character sets that may be specified is ten; the default minimum number of screens is one and the maximum number that can be specified is eight. The user is advised to consult Table I above in making the selection of the number of screens to be used. In selecting the maximum number of character sets it should be realized that even with only three character sets enabled any number of character sets may be used within a program. Whenever a new character set is necessary it is just called, the Grafix Kernel takes over and loads it as character set three, accessed and specified by its name (the previously loaded character set is over-written). Note that a given character set need not be resident in RAM in order for the characters to be displayed on screen. That is, once a character from any given character set has been placed on the Hi-Res Screen, that character set need no longer be in RAM for the character to remain on the screen (this is in sharp contrast to normal or character mode display, where a character set must be resident in RAM in order for that character to be displayed on screen). The only reason for specifying a maximum number of character sets larger than two in a given application that uses multiple character sets is to minimize disk access. That is, all the character sets that are going to be used in an application can be loaded once and remain resident in RAM and may be accessed at any time without the necessity of loading from disk.

Character Sets and Printing

A character set as referred to in the Grafix Kernel contains 128 characters. Each character is defined within an array which is 16 dots wide and 16 dots high. The Character Height is a parameter which specifies the height of the character measured in dots starting from the bottom row of 16 dots upward; a number which is constant for all the characters within a character set. The Character Width is a parameter which specifies the width of the character measured in dots starting in the left most column of 16 dots and moving to the right. For a given character set, this parameter may be fixed or take on different values for each character.

Each character set contains within it the information used by the Grafix Kernel to properly print characters from that character set. The character set file HEADER contains the following flags specifying a character to be: Normal or special, horizontal or vertical, fixed width or proportionally spaced. A normal character set is one with a width of 10 dots, a height of 16 dots, of the type that is used to construct a system and is booted at the same time the system is brought up.

The attributes horizontal, vertical and proportional, non-proportional are handled by the Grafix Kernel automatically. For example, if a proportionally spaced character set is enabled and the Hi-Res print function called and the string that is desired to be printed passed, the Grafix Kernel will automatically print the string starting at the current cursor location and proportionally space the characters within the string.

Due to the approximately 3:2 aspect ratio of the screen different character sets must be used for horizontal and vertical printing. These are attributes of the character set that the Grafix Kernel treats accordingly. That is, a horizontal character set is normally printed horizontally (left to right), a vertical character set is normally printed vertically (default upwards). To emphasize, if one desires to print vertically one merely needs to specify a vertical character set and from that point on, the printing will be vertical from whatever the cursor position is just by sending the appropriate escape sequence to enable the Hi-Res print as well as the string of characters to be printed. The portion of the 16 by 16 dot character that is printed, together with the direction of print is shown below in Figure 3.

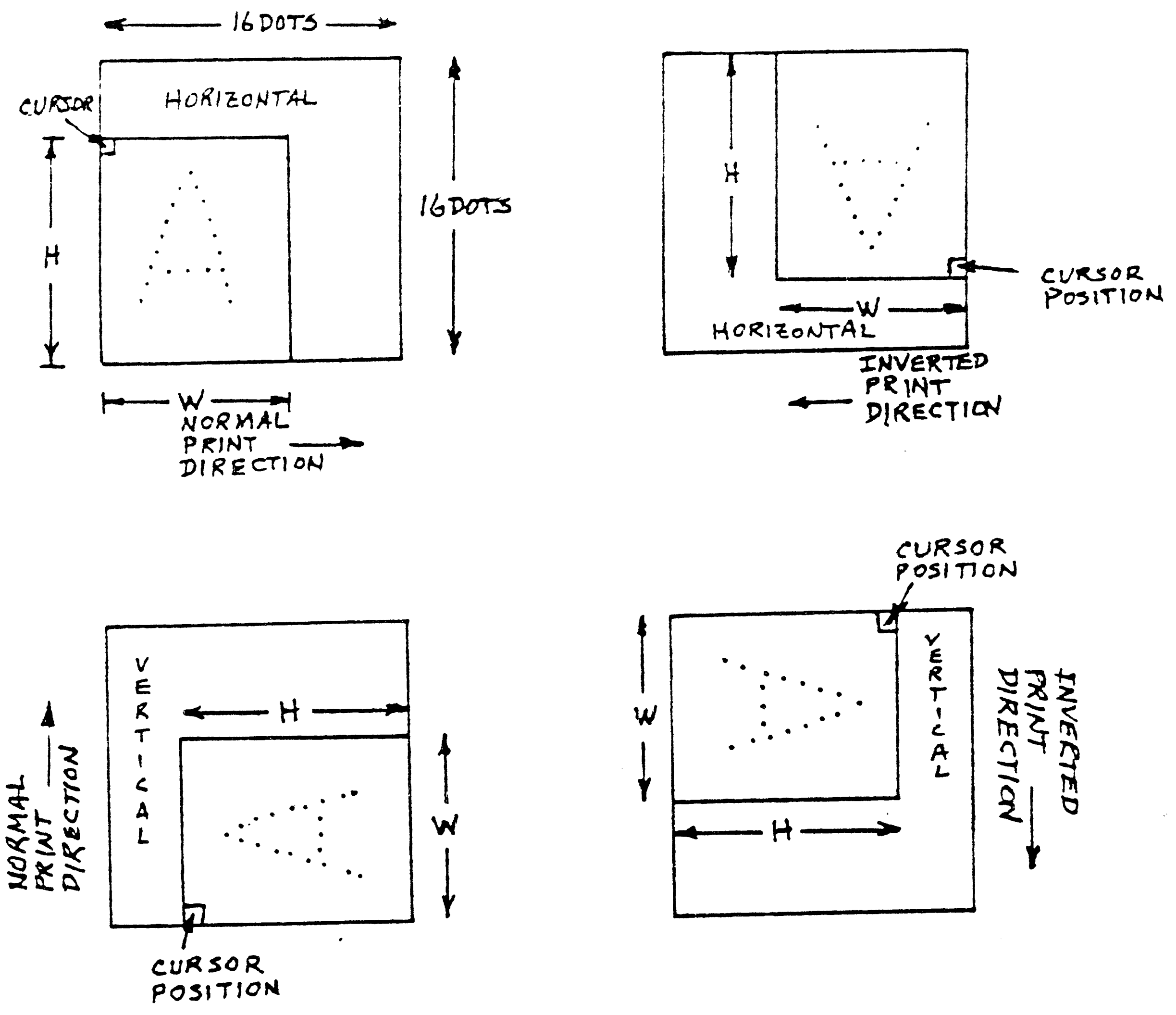


FIGURE 3 CHARACTER DEFINITIONS

Grafix Commands

The Grafix Kernel has four types of commands. They are:

1. Parameter Input
2. Parameter Return
3. Action - Screen Draw, Print or Move
4. File Maintainance

Under the **Parameter Input** commands we can:

Select the screen to draw on

Select the screen to be displayed

Select the fill pattern

Define a screen window

Define a text window

Select a character set

Select a cursor type

Select a combination rule

Set relative and absolute cursor positioning

Set line width

Set line type

Set a dot

Set and reset superscript mode

Set and reset subscript mode

Set and reset invert character and print direction

Set and reset double character size mode

Define user cursor

Define user fill pattern.

The **Parameter Return** commands are:

- Get enabled screen number
- Get displayed screen number
- Get window size
- Get dot
- Get character width
- Get character height
- Get character type

The **Action** commands are:

- Fill a pie section
- Fill a bar
- Draw a circle
- Draw an elipse
- Absolute and relative line draw
- Move a window or a screen.

The **File Maintainance** commands are:

- Save a window on disk
- Load a window from disk
- Select character set (if that character Set is not already in memory).

Software Development Using The Grafix Kernel

The Grafix Kernel was designed to optimize software development using Hi-Res graphics. First notice the text window feature. This allows for the definition of any number of lines any where on the screen as a text window. If a system error should occur under most languages (interpreted or compiled) the system generates a carriage return line feed. This automatically exits the Grafix Kernel and any subsequent print as in the case of an error message will be using normal characters (character set 0 which is booted with a system) and be directed to the text window. In addition, even while one is in Hi-Res, one can list a program and that listing will be printed within the text window. The text window behaves as a normal terminal and supports most of the VT52 escape sequences. When the bottom of the text window is reached a scroll within the text window is energized. To debug programs using Hi-Res graphics the following proceedure is suggested; do all the Hi-Res printing, drawing, etc., on a background screen leaving Screen 0 the viewed screen, totally in text mode. If no errors occur, it is easy to bring the gragix to the viewed screen or select the screen the graphics is being drawn on as the viewed screen to see the results. However, if an an error does occur the graphics is not disturbed and the error is printed using normal characters on the viewed screen.

Secondly, screen file routines are provided so that Hi-Res graphics, screens, or portions of screens called windows, can be created and stored on disk. Within the execution of the program such screen files may be called up as needed.

In addtion, some higher level routines are provided to make the software development easier: drawing of multiple line types and widths is directly supported, text functions are easily implemented and a graphics move function allows for one of 16 logical combinations of two graphics areas. For example, the ability to set the left margin on the Hi-Res screen allows for multiple column printing on the screen. A left margin of 0 allows the formation of the first column (the user must manage to line width) then the left margin can be redefined to say, 399, and printing resumed in the second column on the screen. When the code for graphics carriage return line feed is sent under graphics print the cursor is automatically positioned to the next line (adjusted for the character set, height) and at the x position specified by the left margin.

Using the Text Window

While in graphics mode (screen in dot addressable mode as opposed to character mode) a text window can be defined considering the screen to be made up of 25 lines of 80 characters. The default for the text window is the bottom three lines of the display. However, by calling the appropriate escape sequence (see Table III), the text window can be set to anywhere on the screen. The text window is always on the currently viewed screen.

The idea of the text window is to simulate normal terminal operation under Hi-Res. Most of the normal functions and their escape sequences are implemented for the text window. The only difference from normal character mode operation is that all cursor positioning, insert and delete functions refer to the text window rather than to the whole screen. For example, escape H, which normally sets the cursor at Home Position, sets the cursor to the upper left-hand corner of whatever text window had been defined. Table II below gives a listing of all escape sequences supported in normal character mode with their corresponding interpretation in graphics mode.

Table II

TEXT WINDOW

Escape Sequences Interpreted by the Grafix Kernel

Cursor Functions

Esc H - Sets cursor at home position. Implemented. Sets cursor to the upper left-hand corner of the currently defined text window.

Esc C - Moves cursor forward one character position. Implemented.

Esc D - Moves cursor backward one character position. Implemented.

Esc B - Moves cursor down one line. Implemented.

Esc A - Moves cursor up one line and beginning of line. Implemented.

Esc I - Moves cursor to the same horizontal position of the preceding line. Implemented.

Esc n - Reports cursor position. Implemented. Returns the row and column of cursor position.

Esc j - Saves current cursor position. Implemented.

Esc k - Returns cursor to previously saved cursor position. Implemented.

Esc Y - Moves the cursor for direct cursor addressing. Implemented. Cursor positioning is within the currently enabled text window. Row 0 Column 0 are at the upper left-hand corner of the currently defined text window. All addressing is relative to that starting point.

Esc E - Erases entire screen. Implemented. Comment: the currently enabled text window is erased (not the whole graphics screen).

Esc b - Erases from the start of screen up to and including cursor position. Implemented. Erases from the upper left-hand corner currently defined text window to and including the cursor position.

Esc J - Erases from the cursor position to the end of the page. Implemented. Erases from the current cursor position to the end of currently enable text window.

Esc l - Erases entire line. Implemented.

Esc o - Erases beginning of line up to and including cursor position. Implemented.

Esc K - Erases from cursor position to the end of line. Implemented.

Esc L - Inserts a blank line, etc. Implemented.

Esc M - Deletes the current line, etc. Implemented.

Esc N - Deletes cursor-position character and shifts the rest of the line one character position to the left. Implemented.

Esc @ - Enters the insert character mode, etc. Implemented.

Esc O - Exits from the insert character mode. Implemented.

Configuration Functions

Esc xPs - All values of Ps with the exception of 5 are ignored. Five turns the cursor off.

Esc yPs - All values of Ps with the exception of 5 are ignored. Five turns the cursor back on.

Operation Mode Functions

Esc [- (Set hold mode). Not implemented.

Esc / - (Clear hold mode). Not implemented.

Esc p - Enters reverse video mode. Implemented.

Esc q - Exits reverse video mode. Implemented.

Esc F - Enters character graphics mode. Implemented.

Action: Selects graphics characters (normal system character set, upper 128 characters) from the system character set for access from the keyboard.

Esc G - Exits graphics mode. Returns to normal character mode.

Esc t - (Enter keypad shift mode). Not implemented.

Esc u - (Exit keypad shift mode). Not implemented.

Esc = - (ALT. keypad on). Not implemented.

Esc > - (ALT. keypad off). Not implemented.

Esc - Disables keyboard. Implemented.

Action: Keyboard input is ignored.

Esc - Enables keyboard. Implemented.

Esc v - Enable word wrap at end of line. Implemented.

Esc w - Disable word wrap at end of line. Implemented.

Esc z - (Resets to power-on configuration). Not implemented.

Esc (- (Sets high intensity). Not implemented.

Esc) - (Sets low intensity). Not implemented.

Esc # - (Transmits page). Not implemented.

Accessing the Grafix Kernel

An escape sequence beginning with Esc 5 is used to access the graphics routines. Any function call (escape sequence) must be terminated with a carriage return line feed (ASCII 0D,0A). The functions must be called in sequence, that is the first function must be terminated prior to calling the next one.

Table III

Grafix Functions: Escape Sequences

Esc 5 A - Select Screen.

Parameter passed: Single ASCII number corresponding to the screen desired. The number must be between 0 and 7.

Action: Selects a screen number which is the currently active screen, and to which all line draws and prints are directed.

Esc 5 B - Select Display Screen.

Parameter passed: An ASCII number specifying which of the enabled screens is to be viewed on the CRT. Current hardware supports selection of screen 0 or screen 1 only if RAM permits.

Action: The specified screen is displayed.

Esc 5 C - Set Superscript Shift Mode.

Parameters passed: None.

Action: Shifts up the cursor by the number dots specified in the currently enabled character set.

Esc 5 D - Reset Superscript Shift Mode.

Parameters passed: None.

Action: Moves cursor down a number of dots specified in the table for the currently enabled character set.

Esc 5 E - Set Subscript Shift Mode.

Parameters passed: None.

Action: Moves cursor down a number of dots specified in the table for the currently enabled character set.

Esc 5 F - Reset Subscript Shift Mode.

Parameters passed: None.

Action: Moves cursor up a number of dots specified in the table for the currently enabled character set.

Esc 5 G - Set Double Character Size Mode.

Parameters passed: None.

Action: Doubles the size of whatever character is printed from this point on. A single character dot is blown up into four dots thus doubling the effective size of the character.

Esc 5 H - Reset Double Character Size Mode.

Parameters passed: None.

Action: Returns all print to normal size.

Esc 5 I - Define Screen Window.

Parameters passed: Two ASCII numbers: One specifying the width of the window (x extent), one specifying the height of the window (y extent). The current cursor position is used to specify the upper left-hand corner of the window.

Action: Stores current window dimensions in table for subsequent move instruction.

Esc 5 J - Set Invert Character and Print Direction.

Parameters passed: None.

Action: If the currently enabled character set is a horizontal character set the action is to flip the characters up side down and proceed printing from right to left. (Not very useful). If the currently accessed character set is a sideways character set, which normally prints from the bottom up (default), the action is to flip it over and print from the top down.

Esc 5 K - Reset Invert Character and Print Direction.

Parameters passed: None.

Action: Returns to normal (default) print direction.

Esc 5 L - Select Fill Pattern.

Parameter passed: A single ASCII number specifying the number for the fill pattern. The number must be between 0 and 8.

Action: Stores the selected fill pattern number in table so that any subsequent fill command will use that pattern.

Esc 5 M - Fill Polygon.

Parameters Passed: None.

Action: Fills the enclosed polygon specified by the cursor position within the polygon with the currently enabled pattern, (eight default patterns are available plus one user specified pattern.)

Esc 5 N - Fill Bar.

Parameters passed: 2 ASCII numbers; the first specifies the width of the bar, the second specifies the height of the bar. The bar is located by cursor positioning to the upper-left corner of the desired bar prior to invoking the Fill Bar function.

Action: The fill is accomplished by using the currently enabled fill pattern.

Esc 5 O - Set Left Margin

Parameters passed: ASCII number ($0 \leq n \geq 799$)

Action: Stores left margin

Esc 5 P - Draw Circle.

Parameters passed: An ASCII number specifying radius in Y units.

Action: Draws a circle centered at the current cursor position.

Esc 5 Q - Position Graphics Cursor.

Parameters passed: Two ASCII numbers specifying the x and the y position on the screen. This is an absolute positioning of the cursor.

Action: Positions the graphics cursor to the absolute x,y position on the currently enabled screen.

Esc 5 R - Relative Position Graphics Cursor.

Parameters passed: Two ASCII numbers specifying the x and the y increments.

Action: Moves the graphics cursor relative to the current cursor position by the increments specified.

Esc 5 S - Save Window on Disk.

Parameters passed: Eight ASCII characters specifying the file name (may be preceded by A: or B: to specify drive).

Action: Saves the contents of the currently defined window from the currently enabled screen on disk in the named file having extent ".SCR". Note that a whole screen can be defined as the window, and therefore saved on disk by invoking this function.

Esc 5 T - Load Window From Disk.

Parameter passed: Eight ASCII characters specifying the file name (may be preceeded by A: or B: to specify drive).

Action: Loads the contents of the named file to the currently enabled screen with the upper left hand corner of the window coinciding with the current cursor position. The file may contain a whole screen or a partial screen; in either case, if the file exceeds the screen boundary, it is clipped. (Extent of file must be ".SCR" and the cursor must be positioned on the screen).

Esc 5 U - Draw Line (absolute).

Parameters passed: Two ASCII numbers corresponding to the absolute x and y position on the screen.

Action: Draws a line from the cursor position to the specified x and y coordinate using the current combination rule, line width and line type. The cursor is moved to the new x,y position.

Esc 5 V - Move Window.

Parameters passed: One ASCII number specifying the destination screen.

Action: Moves a region from the enabled (source) screen defined by the currently defined window to the destination screen using the cursor position as the upper left-hand corner, using the currently set combination rule. Selects the specified screen as the enabled screen when the move is done. Note that the window can be moved from one location to another on the same screen, as well as from one screen to another.

Action: Moves the window using currently enabled combination rule.

Esc 5 W - Move Screen.

Parameter passed: Single ASCII number specifying destination screen.

Action: Moves the currently enabled screen to the screen specified using the currently enabled combination rule.

Esc 5 X - Set Combination Rule.

Parameters passed: ASCII number corresponding to one of sixteen combination rules (these are listed in a later section).

Action: Stores the currently selected combination rule in table - this rule applies to all moves, character prints and line draws.

Esc 5 Y - Set Line Width

Parameters passed: ASCII number corresponding to the width (in y units) in dots desired for the line drawing routine. The supported numbers are 1, 2, 4 and 6 specifying the width of a horizontal line in dots (other than horizontal lines are automatically scaled to give equal widths).

Action: Stores value in table. This value is used for all line drawing, circles , ellipses, and arcs.

Esc 5 Z - Set Line Type

Parameters passed: ASCII number corresponding to one of five types of lines

Type 1 - solid line contiuous between the cursor position the currently specified x and y.

Type 2 - dashed line: four dots on four dots off.

Type 3 - dashed line: 8 dots on 4 dots off.

Type 4 - dot dash: 4 dots on 2 dots off 2 on 2 off.

Type 5 - dot dash: 8 on 3 off 3 on 3 off.

Action: Stores value in table.

Esc 5 a - Get Character Width.

Parameters passed: single ASCII character.

Action: Reads the width of the specified character and passes it back as an ASCII character.

Esc 5 b - Get Dot.

Parameters passed: None.

Action: Passes back attribute of dot pointed to by the graphics cursor.

Esc 5 c - Set Dot.

Parameters passed: Single ASCII character (1 or 0) specifying the source dot as on or off.

Action: Combines the source dot as specified with the dot pointed to by the cursor according to the currently set combination rule.

Esc 5 d - Define Text Window.

Parameters passed: Two ASCII numbers - the first specifies the starting line number, the second the height of the window in lines. The screen is assumed to be 25 lines by 80 characters.

Action: Enables a window for printing non-proportionally spaced characters (default character set 0). The text window screen is accessed automatically for all non-hi-res printing. Scrolling is enabled when the bottom line of the window is reached. The text window is always on the viewed screen.

Esc 5 e - Get Window.

Parameters passed: None.

Action: Returns four ASCII number specifying the currently enabled graphics window - the first pair of numbers specifies the x and y position of the upper-left corner of the window, the next pair of numbers specifies the width and the height of the graphics window.

Esc 5 f - Draw Line (relative).

Parameters passed: Two ASCII numbers specifying the x,y increments.

Action: Draws line starting at cursor position and incremeting x and y by argument specified.

Esc 5 g - Draw Ellipse

Parameters passed: Two ASCII numbers specifying the length of x-axis and the length of the y-axis

Action: Draws an ellipse centered at the current cursor position.

Esc 5 h - Draw Arc.

Parameters passed: Three ASCII numbers specifying the radius (in y units), and the two angles between which the arc is to be drawn.

Action: Draws a circular arc between the specified angles in a counterclockwise direction.

Esc 5 i - Select Character Set.

Parameters passed: <d:> filename of desired character set.

Action: Selects the named character set for all subsequent Hi-Res print. If character set is not in memory it is loaded from disk. If all allocated character set locations are used, the least recently accessed character set is overwritten. (character set 0-the system set is protected from overwriting and is accessed by passing a blank filename.) d: specifies the drive to load from if present otherwise the default drive is assumed.

Esc 5 j - Get Selected Screen Number.

Parameters Passed: None.

Action: Returns ASCII number of currently selected screen.

Esc 5 k - Get Viewed Screen Number.

Parameters Passed: None.

Action: Returns ASCII number (0 or 1 for current hardware) indicating currently viewed screen.

Esc 5 l - Get Character Height.

Parameters Passed: None.

Action: Returns ASCII value specifying the height of the currently selected character set.

Esc 5 m - Define User Cursor.

Parameters Passed: 34 ASCII numbers; the first two represent the desired x and y offset from the upper left hand corner to the cursor dot within the pattern, the last 32 represent the cursor pattern ($0 \leq n \leq 255$).

Action: Stores specified pattern as cursor type 3 in the table for future use.

Esc 5 n - Define User Fill Pattern.

Parameters Passed: 33 ASCII numbers; the first 32 specify the 16 by 16 dot pattern to be used as the fill pattern, the last number specifies the fill number that this user defined fill pattern is to be assigned to (normally 8, but user may redefine any fill pattern).

Action: Stores the specified fill pattern in the appropriate location in the table.

Esc 5 o - Get Character Set Type.

Parameters Passed: None.

Action: Returns an ASCII number specifying the selected character set type as follows:

0 - normal char. set booted with system (10 dots wide, 16 dots high)

10 - horizontal, non-proportional

11 - horizontal, proportional

12 - vertical, non-proportional

13 - vertical, proportional.

20 - special, horizontal, non-proportional

21 - special horizontal proportional

22 - special vertical non-proportional

23 - special vertical proportional

Esc 5 p - Hi-Res Print.

Parameters passed: Printable ASCII characters terminated by carriage return line feed.

Action: Prints character at current graphics cursor position and updates cursor position.

Esc 5 q - Enable Cursor.

Parameters passed: None.

Action: Prints the currently selected cursor at the cursor position.

Esc 5 r - Disable Cursor.

Parameters passed: None.

Action: Does not print the cursor.

Esc 5 s - Enable Shadow Print.

Paramers passed: None.

Action: Causes shadow printing of all characters using Hi-Res print from this point on.

Esc 5 t - Disable Shadow Print.

Parameters passed: None.

Action: Returns to normal print.

Esc 5 v - Set Reverse Video Mode.

Parameters passed: None.

Action: All characters printed from this point on are in reverse video.

Esc 5 w - Reset Reverse Video Mode.

Parameters passed: None.

Action: Returns to normal print.

Esc 5 x - Select Cursor Type.

Parameters passed: ASCII Number as follows.

0 - Block

1 - Cross - hair

2 - Arrow

3 - User defined

Action: Uses specified cursor. (Default - Arrow).

Esc 5 y - Set Underline Mode.

Parameters passed: None.

Action: All characters printed from this point on have an underline (the bottom row of dots within the character cell in turned on).

Esc 5 z - Reset Underline Mode.

Parameters passed: None.

Action: Returns to normal print.

Esc 5 0 - Save Graphics Cursor Position.

Parameters passed: None.

Action: Saves the current graphics cursor position for subsequent return.

Esc 5 1 - Return Graphics Cursor to Previously Saved Position.

Parameters passed: None.

Action: Returns graphics cursor to position saved when Esc 5 0 was executed.

COMBINATION RULES

The the portion of a screen (or a whole screen) to be moved is called the source and is labeled by S; the portion of the screen (or a whole screen) targeted as the location of the move is called the destination prior to the transformation and is labelled by D; the destination after applying the transformation is labelled by D'. With these definitions in mind, the following rules specify the possible transformations of the destination by the source:

0. D'=0 - the destination ram contains all zeros.

1. D'=S and D - the destination is formed by anding the corresponding bits in the original destination and the source.

2. D'=S and not D - the destination is formed by anding the complimented original destination with the source.

3. D'=S - the source is moved to the destination.

4. D'=not S and D - the destination is formed by anding the inverted source with the destination.

5. D'=D - no operation: no change in the destination results immaterial of the source.

6. D'=S xor D - the destination is formed by exclusive-oring the corresponding bits in the original destination and the source.

7. D'=S or D - the destination is formed by oring the corresponding bits in the original destination and the source.

8. D'=not S and not D - the destination is formed by anding the complimented the complimented destination and the complimented source.

9. D'=not S xor D - the destination is formed by exclusive-oring the corresponding bits in the original destination and the complimented source.

10. D'=not D - the destina;tion is formed by complimenting all the bits in the original destination.

11. D'=S or not D - the destination is formed by complimenting the original destination and oring the corresponding bits with the source.

12. D'=S xor D - the complimented source is moved to the destination.

13. D'=not S or D - the destination is formed by oring the corresponding bits in the original destination and the complimented source.

14. D'=not S or not D - the destination is formed by oring the complimented original destination and the complimented source.

15. D'=1 - the destination ram contains all ones.

Figure 4 shows the posibilities given a simple destination and source. Note that the power offered through the combination rules gives the ultimate flexibility in graphic presentation, including simulating motion on the screen.

Some simple examples of using the combination rules:

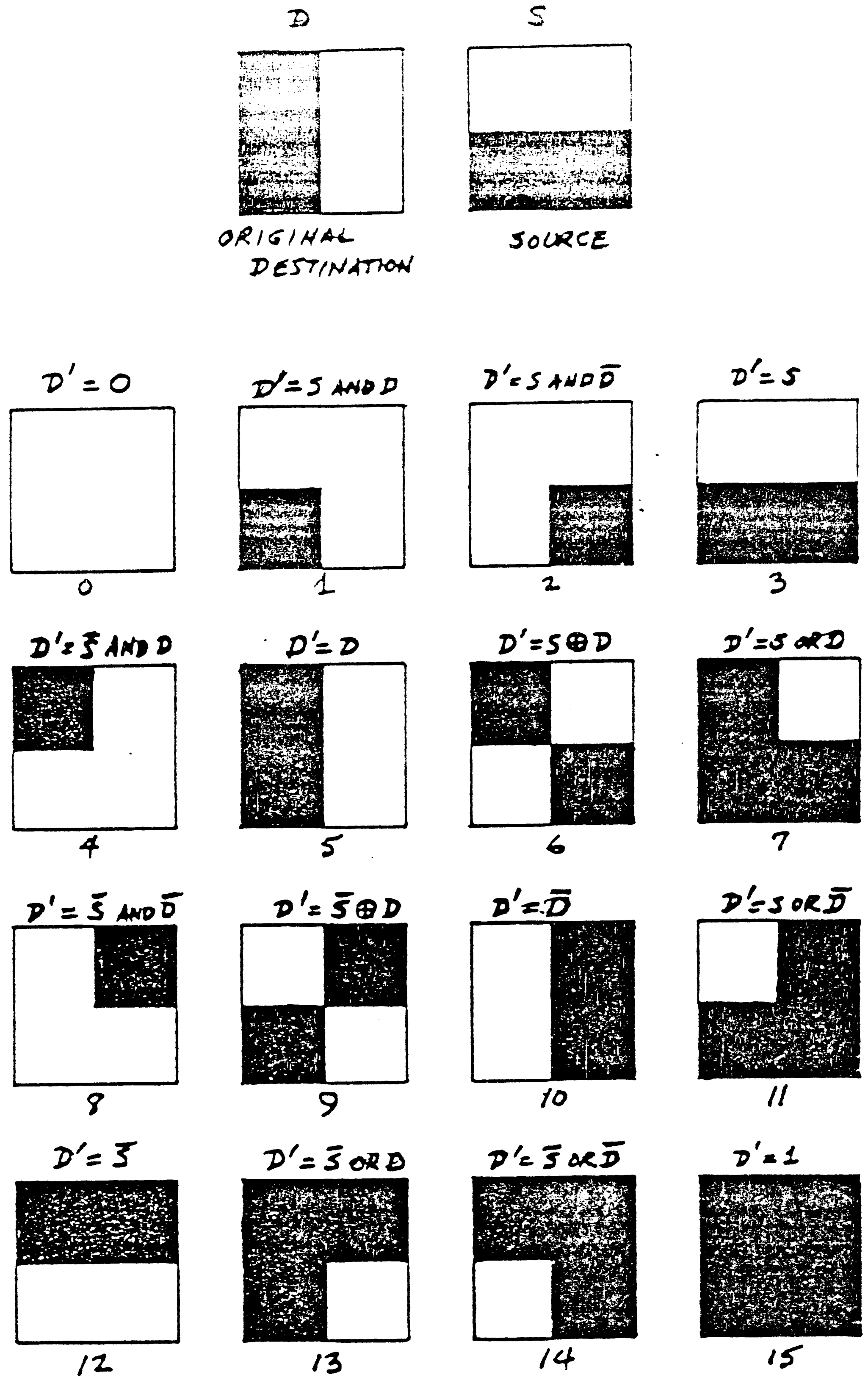
If combination rule D'=not D is selected and the move specified from Screen N back to Screen N the effect is to invert each dot on Screen N.

In addition, since in the current hardware only Screen 0 or 1 may be selected as the display screen, this function allows Screen N (if N>1) to be moved to Screen 0 or 1 whichever one is selected for display and thus allows Screen N to be displayed.

The move together with a combination rule allows any of the screens to be erased setting it to all ones or all zeros. In a word processing application if a screen is cleared to all ones and characters are EXORed into the screen a black on white output is acheived. Simply invoking D'=not D will create the normal white on black display.

A given screen may be selected as a scratch-pad where drawings, graphs, characters, may be formed and brought into the currently viewed screen by invoking a move through any of the combination rules.

FIGURE 4
COMBINATION RULES



Hi-Res Printing Routines

Besides the ability to print within the text window a special Hi-Res print function is available (Esc 5 p). To use, simply position the cursor where you desire the upper left-hand corner of the first character to be placed on the screen. Call the Hi-Res Print Function and pass the string of characters desired to be printed. The function is terminated by a carriage return line feed. In addition, prior to executing the print the type font desired may be specified using the Select Character Set (Esc 5 i) function. Printing may be accomplished in a number of styles: Shadow printed, reverse video, and underlined. Shadow printing is accomplished by printing the character twice; the second time ORing-in the same character moved over one dot to the right. The effect is to highlight that character. Reverse video and underline function in exactly the same manner as in a normal terminal. These three may be used one at a time or in combination.

Superscript and subscript modes work as follows: the cursor is moved one half the currently selected character height up or down as specified. The reset of subscript and superscript just undoes the move. Therefore, if a given character set is being used, going into subscript mode allows the printing of subsripts (multiply indented if desired) and then followed by a reset of subscript mode. Continuation of printing will line up with the original line. Note that if a change in character font size is required, the user must manage the exact positioning of the cursor to accomplish multiple subscript/superscript print.

Another print enhancement is the enabling of the double character size mode. In this mode whatever the character set selected, printing will ensue with the upper left-hand corner of the character located at the current cursor position. The only difference is that the character size is doubled (each dot within the character definition is reproduced as four adjacent dots).

Yet another function allows the inversion of the character as well as the print direction. This is the Set Invert Character and Print Direction Function (Esc 5 J). This function is discussed in more detail in the introductory sections of this manual.

In case a proportionally spaced character set is used, while the graphics package will automatically space the right amount to print a string proportionally spaced on the screen, the user must manage the location of the right margin by using the Get Character Width Function (Esc 5 a). Note that microspacing on the screen can be accomplished by using the relative cursor position function to move over one or more dots wherever required.

Error Messages

To help with software development, if an invalid command is given, the graphics package generates one of the following 8 error messages:

0. **Invalid Command** error is generated when executing Esc 5 followed by a symbol not interpreted by the Grafix Kernel.

1. **Parameter(S) Missing** error generated when invoking one of the escape sequences which expects parameters to be passed and the parameters have been omitted or an insufficient number of parameters have been passed.

2. **Invalid Parameter** error is generated when any invalid command is executed. For example, if only three screen have been enabled at installation and a move screen or window is executed to Screen 4. Another cause for generating this error is passing numeric parameters when alphic parameters are expected or vice versa.

3. **Cursor Out Of Range** error is generated whenever an attempt is made to move the cursor out of the cursor positioning range which is between -800 and 1599 in the x direction and -400 and 799 in the y direction.

4. **File Not Found** error is generated when trying to enable a character set whose name does not exist on the disk in the specified or default drive. Similarly if a load window from

disk function is accessed and the file name passed does not exist on the disk in the specified or default drive.

5. **Disk Full** error is generated if the save window on disk on command is executed and the specified or default disk does not have sufficient space to save the file.

6. **Invalid File** error is generated when executing a select character set function and the file name passed does not have extent ".CHR" or when executing a load window from disk function and the file name passed does not have extent ".SCR".

7. **Table Overflow** error is generated in using the fill function and indicates that the polygon specified for the fill has too complex a shape (too many discontinuities).

Questions and Answers in Using the Grafix Package

1. Where can I position the cursor?

All cursor positioning refers to the currently selected screen. The cursor may be positioned anywhere within the actual screen or in the space between the limits of -800 and 1599 for x and -400 and +799 for y. The total cursor space is shown in Figure 5. Note that the cursor may be selected for printing (cursor turned on) or non-printing (cursor off) within the actual screen but defaults to non printing if positioned outside the actual screen. To move the cursor from one screen to another, simply enable the desired screen.

2. Since an extra screen uses up 40,000 bytes of RAM what use is there for an extra screen?

A screen, or a portion of the screen, can be saved on disk. In applications where numerous frames are to be displayed Screens 0 and 1 should be used in the following manner: Load the first frame from disk to Screen 1 while viewing Screen 0. Flip the viewing screen to 1 and then load the next frame from disk to Screen 0. At that point flip the viewing screen to 0. In this manner the speed with which data is retrieved from disk and loaded into the screen is not perceptible to the end user.

Another use for multiple screens is when the graphics to be displayed on screen is generated during the execution of the program; it may be useful to build complete screens or partial

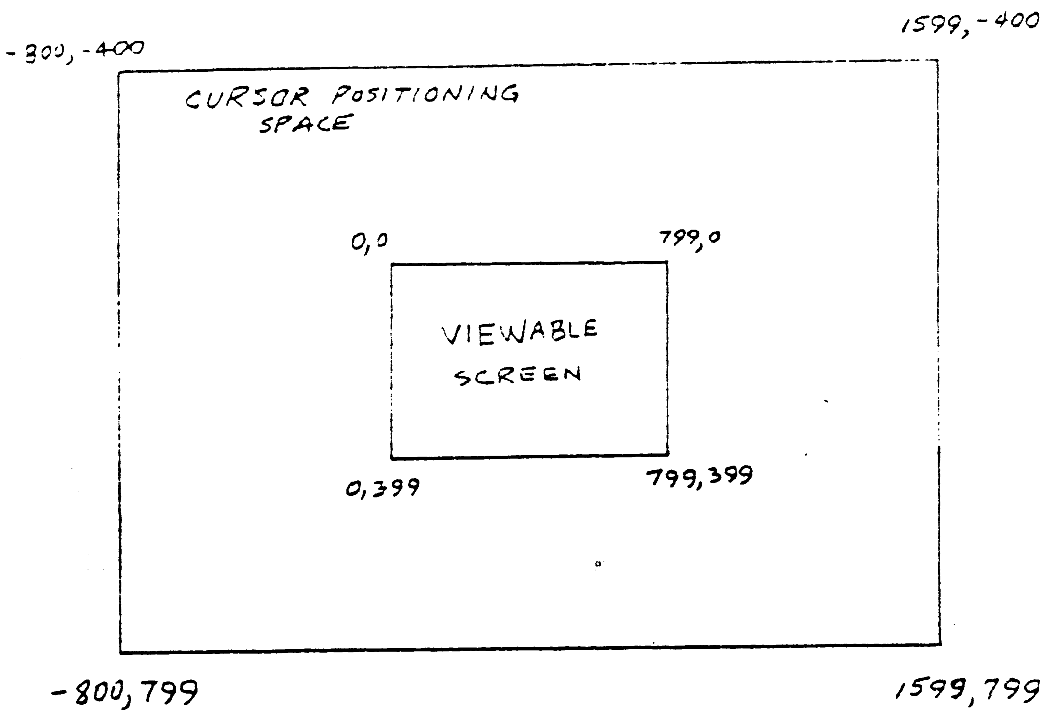


FIGURE 5

CURSOR SPACE

screens called Windows, in background that is on a screen not currently being viewed and bring those onto the viewed screen, so that the eventual end user does not actually see the building of the screen. Yet another use for multiple screens is to save time in applications where multiple graphics screens are necessary at the same time. Rather than rebuild or bring in from disk successive frames, they can be built or brought in from disk once and selected or moved to the viewable screen when desired.

3. How do I use the Window function?

The Window is a rectangular portion of the screen, of any size, up to and including the whole screen. A window is defined as a rectangular area on the currently enabled screen by positioning the cursor to the desired location of the upper left-hand corner of the window. The extent of the window is entered by using the define window function and specifying the width and height of the window in dots. A window can be saved on disk, loaded from disk; it can be moved from one screen to another, or within the same screen. To illustrate the use of windows let us go through the following example. Let us assume that the upper left-hand quarter of Screen 3 is to be moved to Screen 0 and become the lower right-hand quarter of that screen. The sequence of calls is as follows:

a. Enable Screen 3.

b. Position Cursor to x=0,y=0

c. Define Window Function x = 399, y = 199.

d. Move Cursor to x=399, y=199

e. Specify Combination Rule if it needs to be changed from whatever the current value is.

f. Invoke the Move Window Function and specify Screen 0.

This will accomplish moving the window from the upper left-hand corner of Screen 3 to the lower right-hand corner of Screen 0 and change the currently enabled screen from Screen 3 to the destination screen, Screen 0.

As another example let us assume that a window has been either defined on another screen or a window file is being loaded from disk. Further let us assume that we do not wish to transfer the whole window but just a portion of the window. To position that window on the selected screen the graphics cursor is positioned at the location where we desire the **upper left-hand corner** of the window. Figure 6 shows four examples. If the cursor is positioned at the point labeled A (above and to the left of the selected screen), the specified window is transferred to the selected screen and only the portion of the window that coincides with this selected screen may be viewed (if the selected screen is also the viewable screen, or if the selected screen is moved to the viewable screen). The hatched area of the window is in effect discarded. Examples of positioning the cursor at B, C, and D show some other possibilities. Obviously, if the cursor is within the selected screen and the window falls entirely within that screen all of the window is transferred.

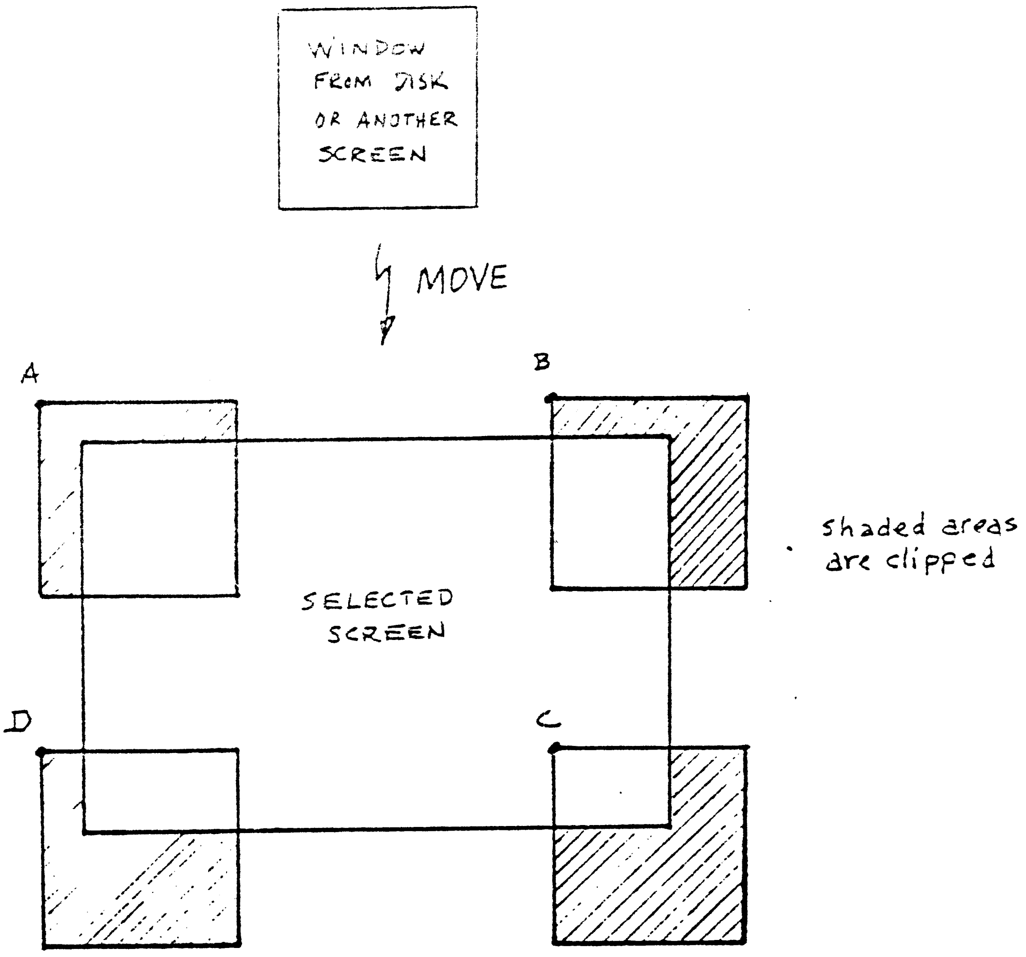


FIGURE 6

MOVING A WINDOW
(see text)

Note that the window function together with a Move need not actually perform a Move. For example, let us assume that we have positioned a cursor at the upper left-hand corner of a window and specified the x extent and y extent of that window. If we do not move the cursor but invoke a move and specify the screen on which the window has been defined, the move is actually not a move. The portion of the screen defined as the window is acted upon by the currently selected combination rule. In this case note that if the combination rule is to form the new destination by inverting the old (D' = not D), the effect is to turn the portion of the screen defined by the window in reverse video. Similarly, the portion of the screen defined by the window can be erased by invoking the appropriate combination rule (D'=0).

The true power of the window function becomes evident once the notion of moving a window, either from screen to screen or within the same screen, using one of 16 possible combination rules, is grasped.

4. **Exactly how do these combination rules work?**

The combination rules, 16 in all, work on a portion of one screen with a portion of the same screen or another screen. The first portion is called the "destination" (D), the second portion, called the "source" (S) is a currently defined window. The rules specify how the destination is transformed by the source. The transformed destination is labeled as D'. Figure 4 gives examples showing a possible destination and a possible source together with the 16 possible transformed destinations. Note that, using the transformation rules, it is possible to turn on all the dots within the destination (D'=1), turn off all the dots in the destination (D'=0), to invert all the dots in the destination (D=not D), to move the source into the destination no matter what the previous destination was (D=S), to move the inverted source into the destination no matter what the previous destination was (D=not S), and to combine the source and destination or either one inverted through the logical anding, oring, or xoring, of the two. Note that only the dimensions of the source need be specified. The destination automatically assumes the same dimensions. The source and destination may be on separate screens, may be different portions of the same screen, or may be identical, meaning the same portion of the same screen.

The combination rules allow for multiple means of achieving the same end. It is left to the users imagination as to all the possibilities of using the combination rules, we shall give two examples here.

One extremely useful function which can be implemented is an exchange of screens. Let us assume that screen 0 is the viewed screen and that we want to view screen 3, however we wish to preserve the contents of screen 0 for future use. If we simply invoke the screen move function and transfer screen 3 to screen 0, the contents of screen 0 are forever lost (the contents of screen 0 and 3 would be the same). To exchage the contents of the two screens without having to write one to a buffer (a 40,000 byte buffer would be needed), we simply invoke the following set of instructions (let the contents of screen 0 be labelled by S0, screen 3 by S3):

a. Set the combination rule to D'= S XOR D.

b. Set the enabled screen to 0 - this places the cursor on screen 0.

c. Move screen to screen 3 - this XORs the contents of screen 0 with screen 3 and leaves the result in screen 3.
S3= S0 XOR S3.

d. Move screen to screen 0 - this XORs the contents of screen 3 with screen 0 and leaves the results in screen 0. S0= S0 XOR S0 XOR S3 = S3. The original contents of screen 3 are now in screen 0!

e. Move screen to screen 3 - this XORs the contents of screen 0 (S3) with screen 3 and leaves the result in screen 3. S3= S0 XOR S3 XOR S3 = S0. The original contents of screen 0 are now in screen 3!

As another example, a "blend" or "fade" effect is possible. Let us assume that we want to fade in the contents of another window or screen (called PIC3) into the viewed screen. In case of a window, we would first define it. The sequence of instructions is as follows:

a. Enable the screen with PIC3 on it.

b. Set the combination rule to OR (D'=S OR D).

c. Move screen (or window) to viewed screen - this ORs in the desired graphics into the viewed screen.

d. Set the combination rule to D'= D.

e. Enable the screen with PIC3 on it.

f. Move screen (or window) to viewed screen - this places PIC3 on the viewed screen.

5. To which operations do the combination rules apply?

The combination rules apply to all window and screen moves, high resolution printing, the set dot function, draw line, draw circle, draw ellipse and draw arc function. All 16 rules apply to window and screen moves and the set dot functions. For the Hi-Res print function, only 3 of the 16 combination rules are supported. They are:

a. hard print the character D'=S

b. OR in the character D'=D OR S

c. XOR in the character D'=D XOR S.

In case a combination rule other than one of the three above is in effect when Hi-Res print is activated, the OR rule is automatically implemented as a default

For all the draw functions, only four of the combination rules apply since the source (line) is assumed to have all dots that make up the line on. The four are:

a. Hard write (or OR in) the line (D'=S).

b. Erase a line (D'= not S)

c. XOR in the line (D'= D XOR S)

d. No operation (D'= D)

6. Do I have to clip to the viewing screen in any of the drawing routines?

No. Referring back to Figure 3 any point within the cursor positioning space ($-800 \leq x \leq 1599$, $-400 \leq y \leq 799$) is fair game for any of the drawing routines. Graphics are automatically clipped with the viewable screen acting as a clipping window, that is if one were to draw a line from the origin (x=0: y=0), to x=1599, y=799 the visible line would be diagonally across the screen ending at x=799, y=399, and the cursor winding up at position 1599, 799. If one attempts to position the cursor outside the cursor space, an error will occur.

7. How can I keep from having to redraw the same graphic figures in different portions of my program?

A number of different ways are available to solve this problem. First, any number of graphics figures can be generated once and saved on disk. Once the program is running, wherever the graphics figure is required, the load window from disk function can be called up, preceded by a proper cursor positioning to the upper left hand corner of the region where we want the graphics figure. The second method involves building graphics dynamically within the program execution. The set of graphics figures needed is drawn on a background screen (a non-viewed screen); and then can be recalled (moved to a viewable screen) when needed.
*(Note that moving a window or a screen does not destroy the original image; that is, if we move a window, say, from Screen 3 to Screen 0, the contents of the window are still available on Screen 3 undisturbed.

8. With three types of cursors, as well as a user defined cursor,

how do I know at which specific dot the cursor is pointing?

In all cases, the graphics cursor points at a single dot on the screen. If a block cursor is enabled the upper left hand corner of the block is the actual absolute x and y position of the cursor. If an arrow cursor is enabled the dot corresponding to the tip of the arrow is the absolute x and y location of the cursor. If a cross-hair cursor is enabled the dot in the middle of the cross-hair specifies the absolute x and y position of the cursor. In the case of the user defined cursor, whatever the cursor character, the absolute location of the dot pointed to is specified with respect to the upper left hand corner of the cursor character (a 16X16 dot array) specified by the x and y cursor offsets.

9. Exactly how do I print on the Hi-Res Screen?

Before we deal with the actual printing on the Hi-Res screen let us briefly digress to examine how character sets are defined. The possible dot pattern for any character is a 16 dot wide by 16 dot high pattern. Each of the character sets, whether predefined or assembled by the user, has certain attributes. These are the character height (H), which is fixed for all 128 characters within a set and specifies the height in dots the field to be covered by the characters. The second parameter, which can be different for each of the characters, is the character width (W) which is the number of dots that the character requires in the horizontal direction. Thus, a given character within any set (fixed width or proportionally spaced) is defined by a window starting at the lower left of the 16x16 dot character array having a width specified for that character and a height specified for the whole character set. In essence one can think of any given character as being a small window which is being transferred to the selected screen.

Thus, let us first take the case of printing a line using only one character set. Since all characters within any given character set have the same height, the cursor proceeds along the top of the boundry defined for the characters as the characters are printed. Note that the Hi-Res print function accepts a string of characters to be printed, automatically spacing from left to right exactly the amount necessary for that character. If the character set is a proportionally spaced one, the printing on the screen is proportionally spaced. Executing a Hi-Res carriage return linefeed (ASCII code 160, 161) within the string causes the Hi-Res cursor to advance a specific number of dots down specified by the height parameter for the currently enabled character set and proceed to print the rest of the string (normal return linefeed terminates the Hi-Res print function). Note that the user must manage the line width. That is, no automatic end of line wrap-around is implemented. In order to properly format a line using proportionally spaced characters the user must first do a look up for the width (W) of the characters within the string. This is accomplished using the get character width function where one character at a time is passed and the width of that character is returned. A line thus formatted can then be printed using the Hi-Res print function together with cursor positioning (either absolute or relative). The crucial thing to realize here is that in Hi-Res print:

a- Multiple characters can be passed at one time and the cursor automatically advances to the next character position each time; and

b- As contrasted to printing on a terminal the cursor is not considered to be a block equal to the size of a character, rather the cursor points to a single dot; this dot is at the upper left hand corner of the character field - the field being H dots high and W dots wide.

The Hi-Res print routine prints any character between 32 and 127 when it receives the corresponding ASCII code. To access characters 0 through 31 within a character set, send ASCII codes of 128 through 159 respectively.