Terak/UCSD Pascal Version II.0
Monochrome Graphics Computer System
Graphics Release Guide

This document describes the graphics capabilities provided by Terak, including SIGGRAPH graphics and the low-level graphics intrinsics for all major languages used on the Terak 8510.

Terak Corporation

Scottsdale, Arizona

60-0060-001B

Terak/UCSD Pascal Version II.0
Monochrome Graphics Computer System
Graphics Release Guide

Document No. 60-0060-001B

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

FIGURES

1. INTRODUCTION

1.1 Computer Graphics

Computer graphics represents a medium for communications between man and computer, that allows the expression of large quantities of information in a format that is easily assimilated with the "mind's eye." For the user who has waded through volumes of line printer output searching for answers, a few appropriate pictures are a welcome relief. For both the novice and experienced users, today's computer graphics resources are providing new and challenging techniques for harnessing the power of the computer and making it available to all.

1.2 Terak Graphics Software

The graphics capability provided by Terak is a general purpose package that is compatible with the ACM SIGGRAPH GSPC Document 1979. (See Appendix A for additional details.) The SIGGRAPH document proposes some guidelines that give the user the ability to write graphics applications and then transport them between systems that also support the SIGGRAPH definitions. The SIGGRAPH definitions provide a bridge between the user's application and the specialized hardware characteristics of the device being used, giving him a sense of hardware independence for his applications. The Terak SIGGRAPH graphics provides the capability of drawing lines and printing characters, while a variety of "projection routines" allow him to rapidly develop any form of graphics application with a minimum of effort.

The Terak 8510 Graphics Processor provides a number of useful intrinsic procedures in addition to the SIGGRAPH capabilities, (See Section 7 for more hardware details.) These routines present the display processor as an array of dots, allowing the user to develop his own specialized algorithms without utilizing assembly language programming.

Reserved

2. SOME GRAPHICS FUNDAMENTALS

2.1 The Graphics Space

2.1.1 General Concepts

A graph is used when a person wishes to represent relationships of data pictorially. For example, one such relationship would be SALES vs. TIME for a particular product history. The graph of such a relationship would typically consist of two axes and special markers connected by lines. Such a graph is presented in Figure 2-1.

To produce this type of graph, the points are described in terms of positions in a Cartesian coordinate system. A two-dimensional Cartesian coordinate system consists of two straight line axes such that any point on the plane may be described in relation to its distance from each axis. In Figure 2-1, SALES are measured in the vertical direction, in thousands of dollars, and TIME is measured in the horizontal direction, in months.

Appendix D contains two figures that represent an analogy of the graphics process. It may be useful to reference these figures while reading the following sections.

2.1.2 World Coordinates

Parameters are passed to the SIGGRAPH graphics procedures using coordinates such as those described above. In other words, the user program defines an area by means of ranges of coordinate values in the vertical and horizontal directions. Lines and points within (or even outside) this area are then described by using these coordinates.

2.1.2.1 The Window

The region defined by the ranges of World Coordinates is known as a Window. When objects (such as lines or markers) are defined either partially or entirely outside of the edges of the Window, clipping automatically takes place at the Window edge (i.e., anything outside the Window "disappears" from the picture).

If a line is drawn such that one endpoint is inside the Window and the other endpoint is outside the Window, the line will be drawn with a slope natural to the connecting points with only that portion of the line inside the Window being visible.

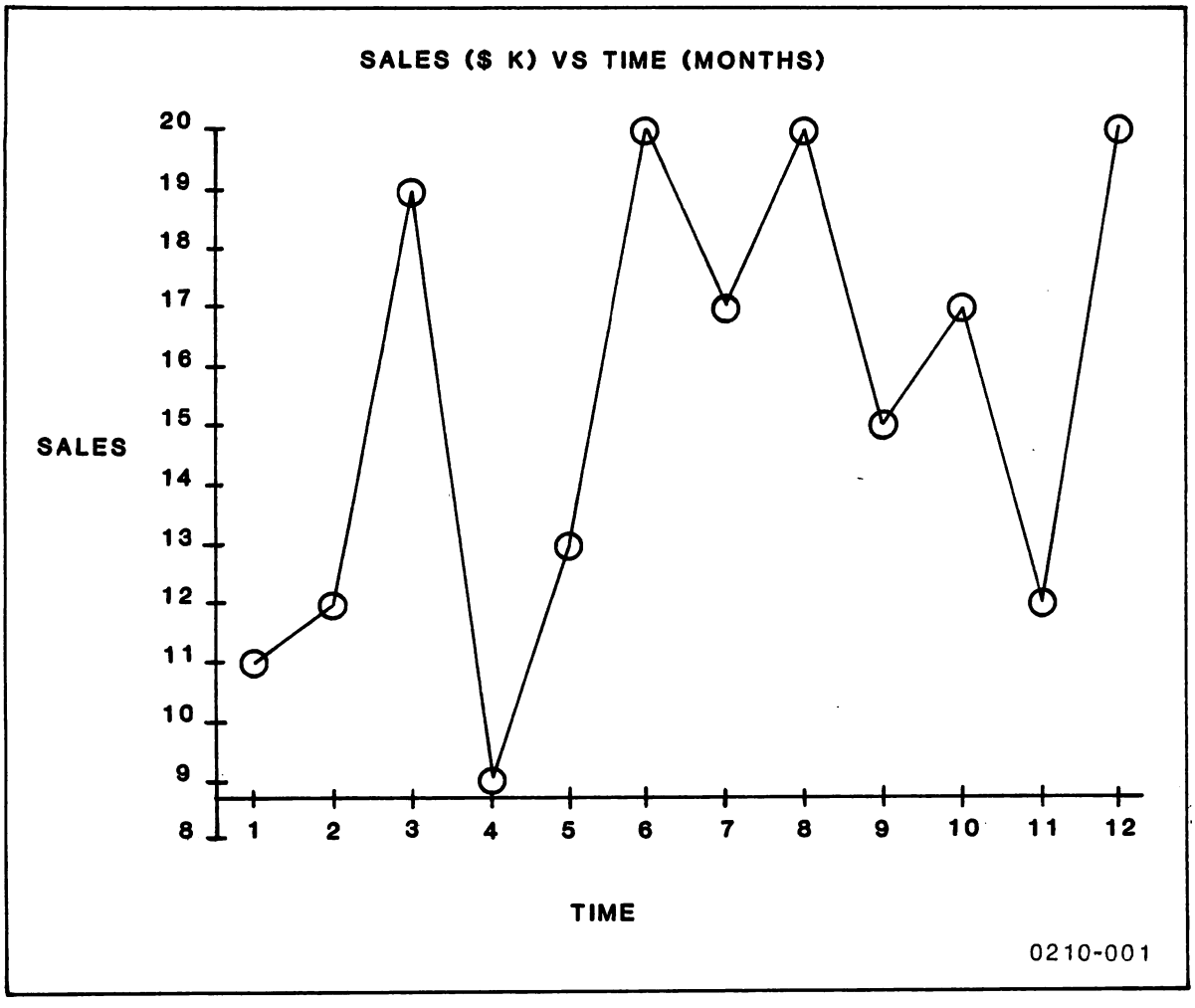


Figure 2.1 Sales vs. Time Graphics Example

2.1.3 The View Surface

To produce a display, the graphics procedure must convert the World Coordinate specifications to the absolute physical screen coordinates of the Terak 8510. To this end, the Window is mapped onto a Viewport, which is a rectangular portion of the View Surface. First, let us define the View Surface.

The physical display of the Terak 8510 is divided into three "zones", each of which is the width of the screen and one third of its height (see Figure 2-2).

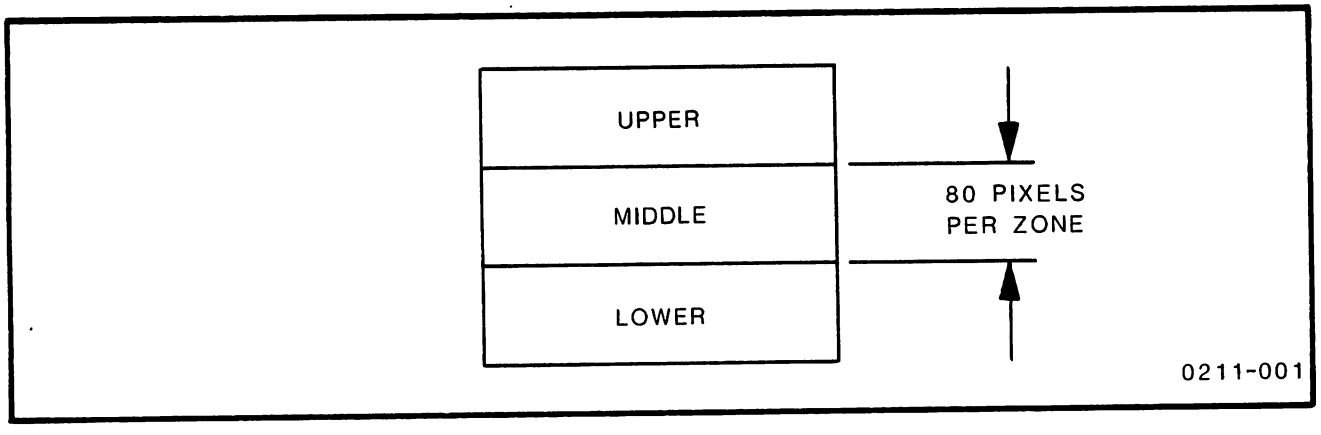


Figure 2.2 Zones

The View Surface is defined by the user program in relation to these zones. The width of the View Surface is the width of a zone (and therefore, the width of the screen). The height of the View Surface may be any multiple of the height of a zone. It may therefore be less than, equal to, or even greater than the height of the screen.

2.1.3.1 Normalized Device Coordinates

Normalized Device Coordinates are used to describe the View Surface. These coordinates range from 0 to 1 in both the vertical and horizontal directions, with (0,0) at the lower left of the View Surface, and (1,1) at the upper right, as shown in Figure 2-3.

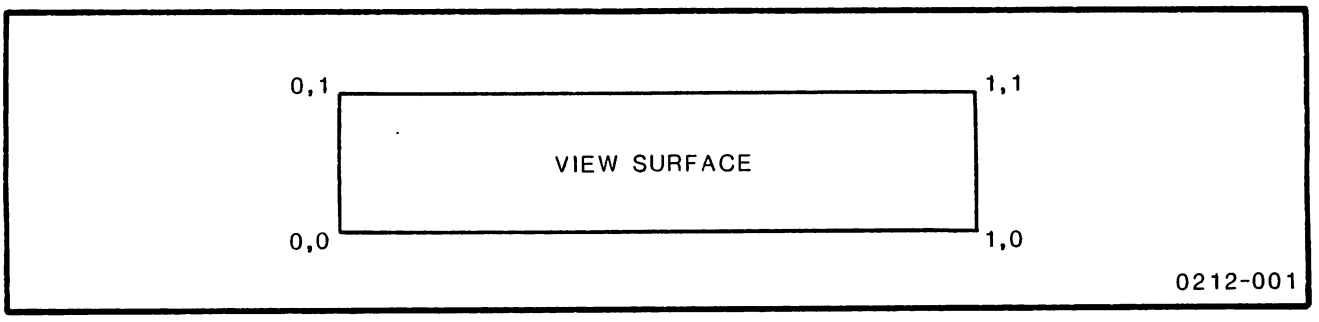


Figure 2.3 Normalized Device Coordinate Orientation

2.1.3.2 Viewports

As we said, the Viewport is a rectangular portion of the View Surface onto which the Window is mapped. Most often, the Viewport will occupy the entire View Surface. However, this is not always desired. For example, consider dividing the View Surface into four quadrants and mapping a different display (Window) onto each quadrant. To do this, it would be necessary to locate Viewports at different locations within the View Surface.

The area of the View Surface in which a particular Viewport is located is defined by the user program with reference to ranges of Normalized Device Coordinates. For example, the ranges .5 to 1.0 in the horizontal directionand 0 to .5 in the vertical direction would map the Window onto a Viewport occupying the lower right quarter of the View Surface.

2.2 Graphics Display Concepts

22.1 The Pen or Cursor

Complete displays on the Terak 8510 are generated by output primitives, which create lines, markers, and text on the display. Intrinsic to this graphics support is the concept of a cursor, or pen, which is always present at some position in the World Coordinate Space. An analogy to this concept is the position of the tip of a pen on a piece of paper.

2.2.2 Moving the Cursor

Using the graphics procedures, it is possible to move the cursor from its current position to a new position in the World Coordinate Space. The user may choose to move the cursor without drawing a line (as if the tip of the pen were lifted off the paper and set down again in a different place), or a straight line may be drawn as the cursor is moved, connecting its starting point with its ending point (just as if a line were drawn by the pen on the paper).

2.2.2.1 Absolute Moves

When moving the cursor as described above, its new position is passed to the graphics procedures in terms of World Coordinates. The cursor may be moved to an absolute position in the World Coordinate Space by specifying the particular vertical and horizontal coordinates desired. For example, suppose that a World Coordinate Space is defined with a range of 0 to 10 in the horizontal direction, and -10 to 10 in the vertical direction. Then moving the cursor to the absolute position of 5,0 would place it in the center of the World Coordinate Space.

2.2.2.2 Relative Moves

The cursor may also be moved to a new position relative to its current position, again using the World Coordinate units. For example, given the World Coordinate Space just described, and given that the cursor's current position is in the center of that space (at the absolute coordinates 5,0), then specifying a relative move of the cursor with the coordinates -5,10 will move it 5 units to the left and 10 units upward, placing it in the top left corner of the World Coordinate Space.

2.2.3 Markers

Procedures are provided to draw predefined markers centered at a point. The size of any marker will be a maximum of 7 dots high by 7 dots wide. Markers are useful to distinguish certain points in the display (they are used in Figure 2-1). The marker procedures move the current position (without drawing a line) to the point where the marker is to be centered and then draw the current marker symbol there.

The position at which the marker is to be centered can be specified in either absolute or relative terms, just as can be done when moving the cursor or drawing a line. The marker symbol to be used is set with a separate procedure. A table of available marker patterns is provided in Section 3.6.1 of this document.

2.2.4 Line Style

Lines, markers and text can be drawn into the display as white, black, or (for lines only) as the complement of the display being overlayed.

2.2.5 Character Precision

A procedure is provided to draw character text into the display. Two variations of text may be displayed: either low precision or medium precision. Low precision text is placed into the Character Page buffer overlaying the graphics display, while medium precision text is placed directly into the graphics display.

When low precision text is chosen, it will be placed into the character display beginning at the row and column closest to the current position of the cursor. Medium precision text is output into the graphics display, with the lower left corner of the first character at the current cursor position. Medium precision text also has the attributes of size and spacing. Size is specified in multiples of the standard 10 dot height and 8 dot width. Spacing is specified in terms of relative World Coordinates. For example, a spacing of .1,-.05 would cause each character to be placed .1 unit to the right and .05 units lower than the previous character.

3. SIGGRAPH GRAPHICS ROUTINES

Terak supports SIGGRAPH-compatible graphics routines for three languages: BASIC, FORTRAN, and Pascal. This section lists all the SIGGRAPH graphics routines supported by Terak in any of those three languages. Each procedure is identified by its complete SIGGRAPH-compatible name, and is described in general terms, common to all three of the languages. Language-specific names and argument lists for the Terak UCSD Pascal Version II.0 Installation are provided in Section 4 of this document. See Appendix A for a discussion of SIGGRAPH compatibility, and Appendix C for a language cross-reference.

3.1 Initialization Commands

The four routines described in this Section on initialization and control of the graphics display space are not a part of the SIGGRAPH Core-79 standard. Implementation using the Terak 8510 graphics hardware necessitated an approach to these control procedures different from SIGGRAPH's, however, the functional intent of the related SIGGRAPH procedures has been maintained. It should be noted that all of the procedures in the Terak implementation, other than the four in this Section, are a part of the SIGGRAPH Core-79 standard.

3.1.1 INIT_GRAPHICS

This procedure initializes the graphics control environment, including marker symbol, Viewport, Window, cursor position, linestyle, and character precision, spacing, and size. For the initial default values of these attributes, see Section 4.5. INIT_GRAPHICS must be called once before calling any of the other graphics procedures, and may be called again if a return to the initial values is desired.

3.1.2 USE_SURFACE

This procedure provides the facility to the user of explicitly defining memory to be used as the graphics display page. This is useful when the user intends to perform I/O to and from memory allocated to graphics (i.e., reading and writing FOTO files), or if he requires dynamic display context switching between different view surfaces. See Section 6 for a discussion of FOTO files.

3.1.3 DRAW_ON_VIEW_SURFACE

This procedure defines the size of the View Surface, where the size allocated is the number of zones specified multiplied by 1600. This is due to the fact that each graphics blanking zone is 1600 16-bit words in size, and the View Surface can be any number of zones in height.

3.1.4 DISPLAY_VIEW_SURFACE

This procedure defines which area of the View Surface is to be mapped into the screen display. Since the View Surface may be any number of zones in height, and only three zones may be displayed on the screen at any one time, the offset or displacement (specified as some number of zones) on the View Surface at which the actual display is to start is included as an argument passed to the DISPLAY_VIEW_SURFACE procedure. A displacement of zero will map to the screen display beginning with the first zone of the View Surface. A displacement of -1 begins with the second zone of the View Surface, and so on. The displacement may be positive, but in the normal case it will be zero or negative.

The three zones on the physical display may each independently be blanked for either graphics or characters. Character blanking and graphics blanking parameters are also specified as arguments to the DISPLAY_VIEW_SURFACE procedure. Each parameter is computed by assigning the values 4, 2, and 1 respectively, to the upper, middle, and lower physical display zones, and then adding up the values of the zones to be displayed. The following table summarizes the eight combinations.

PARAMETER VALUE	EFFECT ON GRAPHICS OR CHARACTER BLANKING
0	Blank all three zones
1	Display lower, blank middle and upper zones
2	Display middle, blank lower and upper zones
3	Display lower and middle, blank upper zone
4	Display upper, blank lower and middle zone
5	Display upper and lower, blank middle zone
6	Display upper and middle, blank lower zone
7	Display all zones

Note that a positive displacement may cause the display of memory outside the View Surface. Any such superfluous memory in the physical display may be blanked by setting the graphics blanking zone pararmeter appropriately.

Also note that once a call to DISPLAY_VIEW_SURFACE has been made, the areas specified will be displayed on the screen at that time, and subsequent lines, markers and text drawn into those areas will appear on the screen as they are drawn. DISPLAY_VIEW_SURFACE may be called at any time, any number of times, and need not be called at all until the entire display is complete.

3.2 Commands to Set and Map the World Coordinate Space

3.2.1 SET_WINDOW

This procedure defines the World Coordinate Space, specified in terms of a minimum and maximum in the horizontal (x) direction, and a minimum and maximum in the vertical (y) direction. Any lines, markers, or text lying outside the Window defined by these parameters are clipped at the Window edge and will not be displayed.

3.2.2 SET_VIEWPORT

This procedure defines which portion of the View Surface (defined by DRAW_VS) is to receive the Window to Viewport mapping. A left, a right, a lower, and an upper bound is specified, all in terms of Normalized Device Coordinates (defined in Section 2.1.5 of this document). The left bound must be less than the right, the lower bound must be less than upper, and all must be between (or may include) the values of 0 and 1.

3.3 Commands to Move and Locate the Cursor

3.3.1 MOVE_ABS

This procedure moves the cursor, without drawing a line, to the absolute World Coordinates specified.

3.3.2 MOVE_REL

This procedure moves the cursor, without drawing a line, from the current position to a new position specified in relative World Coordinates. In other words, the new position is determined by adding the relative World Coordinates specified to the current absolute position.

3.3.3 INQUIRE_CURRENT_POSITION

On calling this procedure, the current position of the cursor is returned, in absolute World Coordinates.

3.4 Commands to Set Background/Line Contrast

3.4.1 SET_LINESTYLE

This procedure defines how lines, text and markers are to be drawn. The following linestyles are available:

PARAMETER VALUE	LINESTYLE
1	White
2	White (same as 1)
3	Black
4	Complement (valid only for lines)

3.4.2 NEW_FRAME

This procedure resets the View Surface defined by DRAW_ON_VIEW_SURFACE according to the current linestyle. If linestyle is equal to 1 or 2, the background will become black. If linestyle is equal to 3 or 4, the background will become white.

3.5 Commands to Draw Lines

3.5.1 LINE_ABS

This procedure draws a line, according to the current linestyle, from the current position to the absolute World Coordinates specified. These coordinates become the new current cursor position.

3.5.2 LINE_REL

This procedure draws a line, according to the current linestyle, from the current position to a new position detemined by adding the relative World Coordinates specified to the current position. This new position becomes the current cursor position.

3.5.3 POLYLINE_ABS

This procedure will draw a connected sequence of lines, according to the current linestyle. An array of x-coordinates, an array of y-coordinates (both in absolute World Coordinates), and a number n (less than or equal to the dimensions of the arrays) are specified. The first line is drawn from the current position to the position specified by the first elements of the x-coordinate and y-coordinate arrays. The second line (if n is greater than 1) is drawn from the ending position of the first line to the position specified by the second elements of the coordinate arrays, and so on, until n lines have been drawn.

3.5.4 POLYLINE_REL

This procedure is identical to POLYLINE_ABS, except that the two coordinate arrays contain relative (rather than absolute) World Coordinates, so that the first line is drawn from the current position to the position determined by adding the coordinates from the first element of the x-coordinate and the first element of the y-coordinate arrays to the coordinates of the current position. The second line (if n is greater than 1) is drawn from the ending position of the first line to the position determined by adding the coordinates from the second elements of the coordinate arrays to the coordinates of the ending position of the first line, and so on, until n lines have been drawn.

3.6 Marker Commands

3.6.1 SET_MARKER_SYMBOL

This procedure sets the current marker symbol to one of the patterns provided. This marker symbol will be used on subsequent calls to the marker procedures. The following marker symbols are available:

PARAMETER VALUE	MARKER SYMBOL
1	single dot (.)
2	plus sign (+), 7 by 7 dots
3	asterisk (*), 7 by 7 dots
4	circle 7 dots in diameter
5	cross (X), 7 by 7 dots
6	vertical bar (|), 5 dots high
7	horizontal bar (-), 5 dots wide
8	diamond, 7 by 7 dots
9	square, 7 by 7 dots
10	square, 5 by 5 dots
11	block (filled square), 7 by 7 dots

3.6.2 MARKER_ABS

This procedure moves the cursor from the current position to the absolute World Coordinates specified (without drawing a line), and the current marker symbol is drawn centered at that position (which then becomes the new current cursor position).

3.6.3 MARKER_REL

This procedure is identical to MARKER_ABS except that the position at which the marker is to be drawn is determined by adding the relative World Coordinates specified to the coordinates of the current position.

3.6.4 POLYMARKER_ABS

Given an x array and a y array of absolute World Coordinates, this procedure will draw a specified number of markers (n), in the following manner: The first marker is drawn centered at the position determined by the first element of the x-coordinate array and the first element of the y-coordinate array. The second element (if n is greater than 1) is drawn at the coordinates specified by the second elements of the arrays, and so on, until n markers have been drawn.

3.6.5 POLYMARKER_REL

This procedure is identical to POLYMARKER_ABS, except that the values of the x and y coordinate arrays are in terms of relative (rather than absolute) World Coordinates. The position at which to center any marker (n) is determined by adding the coordinates from the nth elements of the x and y coordinate arrays to the coordinates of the position of the previous marker.

3.7 Text Commands

3.7.1 SET_CHARACTER_PRECISION

This procedure determines the type of text that will be output by the TEXT procedure. Two types of text are available: Low precision and medium precision. Medium precision text may be output at two different speeds, depending on the visual impact desired by the user. The following table summarizes the three varieties:

PARAMETER VALUE	TYPE OF TEXT
1	Low precision
2	Medium precision, fast
3	Medium precision, slow

3.7.2 SET_CHARACTER_SIZE

This procedure is applicable only when medium character precision (a value of 2 or 3) is specified using the SET_CHARACTER_PRECISION routine. The standard character size is 10 dots high by 8 dots wide. Width and height are specified to SET_CHARCTER_SIZE in terms of multiples of these standard values. For example, specifying a width of 2 and a height of 3 would produce characters that were each 16 dots wide and 30 dots high, or twice the width and three times the height of the standard character.

3.7.3 SET_CHARACTER_SPACE

This procedure is applicable only when medium character precision (a value of 2 or 3) is specified using the SET_CHARACTER_PRECISION routine. SET_CHARACTER_SPACE defines the spacing between each character produced by TEXT, in terms of relative World Coordinates. In other words, after each character is drawn, the current position is updated by adding in the coordinates specified.

3.7.4 TEXT

This procedure outputs the specified string of characters according to the current values of Character Precision, and if applicable, Character Size and Spacing.

Reserved

4. TERAK UCSD PASCAL VERSION II.0 IMPLEMENTATION OF SIGGRAPH GRAPHICS

4.1 Procedure Names

For the UCSD Pascal Version II.0 implementation of SIGGRAPH graphics, the procedure names were chosen to not exceed eight characters, because uniqueness is guaranteed only for the first eight characters of UCSD Pascal identifers.

4.2 Arguments

Arguments may be passed to the procedures either as constants or as variables that have been previously defined as the appropriate type. In Section 4.4, the procedures available in the Pascal implementation with their shortened names, argument lists, and variable types are summarized.

4.3 Using SIGGRAPH Graphics on the Terak/UCSD Pascal Version II.0 System

The SIGGRAPH graphics procedures are included as a unit in the system library (SYSTEM.LIBRARY). They are reached by including the statement "USES GRAPHICS;" directly after the PROGRAM <identifier> statement in the user program. When the program is compiled, the Compiler indicates to the system that linking is required before execution. The linker is automatically invoked when the program is R(un. It will search SYSTEM.LIBRARY for the GRAPHICS Unit, and will link it into the workfile. The Linker may also be explicitly invoked, and in some cases must be. See Section 1.6 of the UCSD Pascal Operating System Manual for directions on using the Linker.

4.4 Terak UCSD Pascal Version II.0 SIGGRAPH Graphics Routines

NAME	ARGUMENT LIST	DESCRIPTION
INIT_GRF		INIT_GRAPHICS initializes the graphics area with the default values given in Section 4.5.
USE_SURF	(surface) Type:PSCR	USE_SURFACE explicitly defines memory, with the name surface, to be used as the graphics display page. See Example 4 in Section 5.4 for a description of type PSCR.
DRAW_VS	(size) Type:INTEGER	DRAW_ON_VIEW_SURFACE defines the size of the view surface, where size equals the number of zones (a zone consists of 1600 16-bit words).

DISP_VS	(disp,graph_zones,char_zones) Type:INTEGER	DISPLAY_VIEW_SURFACE defines which space is to be mapped into the screen display. Disp defines mapping; graph_zones and char_zones define displacement. See Section 3.1.4.
SET_WNDW	(xmin,xmax,ymin,ymax) Type:REAL	SET_WINDOW defines the x,y World Coordinate Space mapped onto the Viewport.
SET_VPRT	(left,right,bottom,top) Type:REAL	SET_VIEWPORT defines which portion of the View Surface is to receive the Window to Viewport mapping. All arguments are in Normalized Device Coordinates.
MOVE_ABS	(x,y) Type:REAL	MOVE_ABS moves the current position to Absolute World Coordinates x,y, without drawing a line.
MOVE_REL	(dx,dy) Type:REAL	MOVE_REL moves the current position to relative World Coordinates dx,dy, without drawing a line. The new current position is computed from the old current position plus the displacements dx and dy.
INQ_CPOS	(xpos,ypos) Type:REAL	INQUIRE_CURRENT_POSITION returns the current x and y positions in absolute World Coordinates into the variables xpos and ypos.
SET_LNST	(linestyle) Type:INTEGER	SET_LINESTYLE defines how lines, text and markers are to be drawn. For values oflinestyle, see Section 3.4.1.
NEW_FRAM		NEW_FRAME resets the View Surface according to the current linestyle. See Section 3.4.2.
LINE_ABS	(x,y) Type:REAL	LINE_ABS draws a line, according to the current linestyle, from the current position to absolute World Coordinates x,y. The current position becomes x,y.

LINE_REL	(x,y) Type:REAL	LINE_REL draws a line, according to the current linestyle, from the current position to the relative World Coordinates x,y. That is, the new current position is computed from the old current position plus displacement dx and dy.
SET_MKSM	(n) Type:INTEGER	SET_MARKER_SYMBOL sets the current marker symbol to pattern n. See Section 3.6.1 for values of n.
MARK_ABS	(x,y) Type:REAL	MARKER_ABS moves the current position to absolute World Coordinates x,y, and draws the current marker symbol centered there.
MARK_REL	(dx,dy) Type:REAL	MARKER_REL moves the current position to relative World Coordinates dx,dy, and draws the current marker symbol centered there. That is, the new current position is computed from the old current position plus the displacements dx and dy.
SET_CHPR	(charprecision) Type:INTEGER	SET_CHARACTER_PRECISION specifies the type of text to be output by the TEXT procedure. See Section 3.7.1 for values of charprecision.
SET_CHSZ	(width,height) Type:INTEGER	SET_CHARACTER_SIZE specifies the size of characters to be output by the TEXT procedure when precision is Medium. Character width will be width*8 dots, and the height will be height*10 dots.
SET_CHSP	(dx,dy) Type:REAL	SET_CHARACTER_SPACE defines the movement of the current position after each character is drawn by TEXT when precision is Medium. The spacing is in relative World Coordinates; that is, after each character, the new current position is computed from the old current position plus the displacements dx and dy.
TEXT	(s) Type:STRING	TEXT outputs the string specified by s according to the current values of character precision, and if applicable, size and spacing.

4.5 Initial Values

When the procedure INIT_GRF is called, the default values of the control attributes are set as if the following calls had been made:

EQUIVALENT CALL	EFFECT
SET_MKSM(1);	Marker is single dot
SET_VPRT(0.,1.,0.,1.);	Viewport = entire View Surface
SET_WNDW(0.,1.,0.,1.);	World Coordinates = Normalized Device Coordinates
MOVE_ABS(0.,0.);	Cursor at lower left corner
SET_LNST(1);	White lines
SET_CHPR(1);	Medium precision characters
SET_CHSP(0.,0.);	No spacing between characters
SET_CHSZ(1,1);	Standard size characters (8 dots x 10 dots)

5. EXAMPLES

5.1 Example 1

The following program fragment shows three zones for graphics.

```
PROGRAM EXAMPLE1;
USES GRAPHICS;
BEGIN
  INIT_GRF;           {Must call first, for initialization}
  DRAW_VS(3);         {Three zones, one full screen}
  DISP_VS(0,7,7)      {Normal displacement, no blanking}
  .
  .
  {Calls to graphics procedures. Elements will be displayed
  as they are drawn.}
  .
  .
END.
```

5.2 Example 2

The following program fragment uses one graphic zone.

```
PROGRAM EXAMPLE2;
USES GRAPHICS;
BEGIN
  INIT_GRF;
  DRAW_VS(1);         {one zone}
  DISP_VS(1,2,5)      {Display graphics in middle zone,
                      characters in upper and lower zones}
  .
  .
  {Calls to graphics procedures.}
  .
  .
END.
```

5.3 Example 3

The following program fragment uses four zones for graphics.

```
PROGRAM EXAMPLE3;
USES GRAPHICS;
BEGIN
  INIT_GRF;
  DRAW_VS(4);          {four zones: full screen + 1}
  DISP_VS(1,3,4)       {Display first two zones of view surface
                       in lower two zones of physical display,
                       with graphics blanked in upper display
                       zone, and characters blanked in lower two.}
  .
  .
  DISP_VS(-1,7,0);     {Now display second, third and fourth zones.}
END.
```

5.4 Example 4

The following program fragment shows how to employ the USE_SURF procedure to explicitly define a View Surface. The argument passed to the USE_SURF procedure must be a pointer to a packed array [0..319,0..79] of BOOLEAN (one zone). Therefore, unless a one-zone View surface is desired, TYPE statements similar to those shown must be included. The type PSCR allows a three zone array to be allocated (with the NEW statement) that same array to be allocated reference as if it were a one-zone array (for the USE_SURF procedure). The fact that the array type passed to USE_SURF is only one zone does not effect the size of the View Surface. That size is specified to the graphics procedure in the call to DRAW_VS, which should follow directly after the USE_SURF call.

```
PROGRAM EXAMPLE4;
USES GRAPHICS;
TYPE SCR3 = PACKED ARRAY[0..319,0..239] OF BOOLEAN
     SCR1 = PACKED ARRAY[0.319,0..79] OF BOOLEAN
     PSCR = RECORD CASE OF BOOLEAN
            TRUE:  (ONE_ZONE.^SCR1);
            FALSE: (THREE_ZONE.^SCR3);
            END;
VAR SURFACE:PSCR;

BEGIN
  INIT_GRF;
  {explicitly allocate memory for three zones}
  NEW(SURFACE. THREE_ZONE);
  {tell graphics to use this array; but this time reference it as
  SURFACE.ONE_ZONE so that the argument is passed to
  USE_SURF is to the correct type}
  USE_SURF(SURFACE.ONE_ZONE);
  {Now tell graphics to make the View Surface three zones (the actual
  size of the array}
  DRAW_VS(3):
  DISP_VS(0,7,7);
  .
  .
  {calls to graphics procedures}
  .
  .
END.
```

5.5 Example 5

The following program fragment shows how to employ the USE_SURF procedure to define more than one View surface.

```
PROGRAM EXAMPLE5;
USES GRAPHICS;
TYPE SCR3 = PACKED ARRAY[0..319,0..239] OF BOOLEAN
     SCR1 = PACKED ARRAY[0.319,0..79] OF BOOLEAN
     PSCR = RECORD CASE OF BOOLEAN
            TRUE:  (ONE_ZONE.^SCR1);
            FALSE: (THREE_ZONE.^SCR3);
            END;
VAR SURFACE1,SURFACE2:PSCR;

BEGIN
  INIT_GRF;
  NEW(SURFACE1.THREE_ZONE);
  USE_SURF(SURFACE1.ONE_ZONE)
  DRAW_VS(3):
  DISP_VS(0,7,7);       {begin displaying Surface 1.}
  .
  .
  {graphics calls to draw on Surface 1.}
  .
  .
  {now allocate memory for Surface 2.}
  NEW(SURFACE2.THREE_ZONE);
  {tell graphics to use Surface 2 as View Surface}
  NEW(SURFACE2.ONE_ZONE);
  {and make it three zones also}
  DRAW_VS(3);
  .
  .
  {now calls made to the graphics procedure will draw on Surface 2,
  but Surface 1 continues to be displayed unchanged}
  .
  .
  DISP_VS(0,7,7);
  {now Surface 2 is being displayed, showing the graphics drawn on it
  while Surface 1 is being displayed.  More calls to the graphics
  procedures now will continue to draw graphics elements on the
  Surface 2 and they will be displayed as they are drawn.  The
  following three statements will switch the display back to Surface
  1 again.:}
     USE_SURF(SURFACE1. ONE_ZONE);
     DRAW_VS(3);
     DISP_VS(0,7,7);
   END.
```

5.6 Example 6

The following program shows how to employ the USE_SURF procedure so that file I/O can be done. The file I/O procedures expect to read and write character data, so the type SCR3 is expanded so that the three-zone array can also be referenced as a packed array [0.4863] of CHAR, which is equivalent to 19 blocks of disk space (at 256 words per block). The actual three-zone array consists of only 4800 words or eighteen and three quarters blocks, but since the I/O procedures BLOCKREAD and BLOCKWRITE operate in increments of one block, the amount of data read or written must be increased to the nearest block boundary (in this case, 19 blocks).

```
PROGRAM EXAMPLE6;
USES GRAPHICS;
TYPE SCR3 = RECORD CASE BOOLEAN OF
            TRUE: (A: PACKED ARRAY[0..319,0..239] OF BOOLEAN)
            FALSE: (B: PACKED ARRAY[0.4863] OF CHAR
            END;
     SCR1 = PACKED ARRAY[0..319,0..79] OF BOOLEAN;
     PSCR = RECORD CASE OF BOOLEAN
            TRUE: (ONE_ZONE.^SCR1);
            FALSE: (THREE_ZONE.^SCR3);
            END;
VAR SCREEN; PSCR;
     PHYLE: FILE;
     S      STRING;

PROCEDURE INPUT;        {load input file}
BEGIN
  RESET(PHYLE, 'FILE1.FOTO');
  IF BLOCKREAD(PHYLE,SCREEN.THREE_ZONE^.B,19)<>19 THEN
     WRITELN('*FILE READ ERROR*');
  CLOSE(PHYLE):
END;

PROCEDURE OUTPUT;       {write out screen to file}
BEGIN
  REWRITE(PHYLE, 'FILE2.FOTO');
  IF BLOCKWRITE(PHYLE,SCREEN.THREE_ZONE^.B,19)=19 THEN
     CLOSE(PHYLE):
  ELSE BEGIN WRITELN ('*FILE WRITE ERROR*');
     CLOSE(PHYLE,PURGE);
  END;
END;
```

The following program uses the above procedures and the graphics procures to read FILE1.FOTO, add a frame to the data read in, and write the result to FILE2.FOTO.

```
BEGIN
  INIT GRF;
  NEW(SCREEN.THREE_ZONE);
  USE_SURF(SCREEN.ONE_ZONE);
  DRAW.VS(3);
  DISP_VS(0,7,7);
  INPUT;
  LINE_ABS(0,1);
  LINE_ABS(1,1);
  LINE_ABS(1,0);
  LINE_ABS(0,0);
  READLN;
  OUTPUT;
END;
```

5.7 Complete Examples

Included in the Pascal Software Release Kit are four demonstration programs that use SIGGRAPH graphics. These are OIL and SALES, STARS and STR.DISP. FIG1D2 produces the graph shown in Figure 2-1 of this document. STARS will draw a start with a specified number of points and will write is to a file STARS.FOTO. STR.DISP will display the file STARS.FOTO once it has been created with STARS.

6. FOTO FILES

Example 6 in Section 5.6 demonstrates the use of file I/O. The filenames in that example have the extension ".FOTO". This does not imply compatability with the FOTO files produced by the Terak utility "GREDIT" although there is some correspondence between the two.

FOTO files created by GREDIT are always nineteen blocks in length, with the first eighteen and three quarters blocks containing the graphics image, and the last quarter block containing information about the state of GREDIT at the time the file was created (cursor location, heading, penstate, etc.).

A file created by writing out an array being used by SIGGRAPH graphics routines: 1) need not have the .FOTO extension; 2) need not be nineteen blocks in length(if more or fewer that 3 zones were written), and 3) does not have any information on the state of the SIGGRAPH attributes at the time the file was created.

A file created by GREDIT can be read and used by the SIGGRAPH graphcis routines provided that nineteen blocks are read into a 4864-word character array. For example, the program in Section 5.6 would work equally well if FILE1.FOTO had been created using GREDIT as it would if the file had been created by writing out a SIGGRAPH graphics array.

The reverse, however, is not true. Although the grphics image contained in the first 18 and 3/4 blocks of the file created by either method is identical in format, GREDIT expects to find information in the final 1/4 block that will not be present in a file created through SIGGRAPH graphics. For this reason, the attempted use of such a file as input to GREDIT is not recommended. The results are at best unpredictable.

Reserved

7. TERAK EXTENSIONS

The information that follows concerns the direct driven pixel graphics intrinsics provided by Terak/UCSD Pascal. The intrinsics provided by UCSD are described in Section 2.1.4 of the UCSD Pascal Operating System Manual, and are described in more detail in the following paragraphs. Also following are descriptions of intrinsics provided only by Terak. All of these graphics intrinsics can be used in conjunction with, or separately from the SIGGRAPH graphics procedures, and are especially useful to support animation, direct image manipulation and user-defined graphics.

7.1 Introductory Information

The Terak 8510 supports bit mapped, raster scan graphics, refreshed directly from main memory. The display presented is a composite of the 240 by 320 dot graphics display with the 24 by 80 character display. Two 8510 registers, in the I/O memory page, control the display of graphics: the Graphics Address Register (GAR) contains the starting address of the memory to be displayed as graphics. The Video Control Register (VCR) controls the blanking/unblanking of the graphics and characters on the video monitor. Detailed descriptions of the operatons of these registers are contained in the 8510 System Installation & Users Guide, Terak Document Number 50-0010-001.

The GAR and VCR may be set from high level Pascal code by the UNITWRITE intrinsic operating on Unit #3 (GRAPHIC:). Before issuing a call to UNITWRITE, the Pascal program should have allocated memory for graphics by declaring a variable. For example, any one of the cases in the following Pascal fragment will allocate one picture space:

```
 TYPE
 TERAKSCREEN = RECORD
   CASE INTEGER OF
   1:( BITS:PACKED ARRAY[0..239] OF PACKED ARRAY[0..319]
       OF BOOLEAN);
   2:( CHRS:PACKED ARRAY[0..9599] OF CHAR);
   3:( INTS:ARRAY[0..4799] OF INTEGER;);
   4:( SETS:ARRAY[0..4799] OF SET OF [0..15]);
   5:( BLKS:ARRAY[0..18] OF ARRAY[0..255] OF INTEGER)
 END;(*CASE*)

VAR SCREEN :TERAKSCREEN;
```

These allocate one picture-full of memory to the variable SCREEN. The screen contents can be manipulated either by direct assignment:

```
SCREEN.BITS[10,100]:= TRUE
```

(which lights the dot at row 10, column 100), by I/O intrinsics:

```
RESET(PIX,'PIX.FOTO'); PIXOK:=BLOCKREAD(PIX,SCREEN,19)=19;
```

(which loads a binary file into the picture), by high level operations:

```
FOR I:=0 TO 4799 DO SCREEN.SETS[I]:=[0..15] - SCREEN.SETS[I];
```

(which reverses the entire picture), or by intrinsic graphic procedures.

The graphic procedures supplied with the Terak release of the UCSD Pascal system are documented here. Note that a picture memory space need not be a full screen, and need not necessarily be displayed while being manipulated. Typically, the picture memory space must be initialized to all blanks or all dots lit. This can be accomplished, respectively, by either of these two statements:

```
FILLCHAR(SCREEN,9600,CHR(0))          for blanks, or
FILLCHAR(SCREEN,9600,CHR(255))        for all dots lit.
```

The generic call of UNITWRITE to volume #3, connects the graphics display hardware of the 8510 with the allocated picture memory:

```
UNITWRITE(3,GARVAL,VCRVAL);
```

where GARVAL = <starting address of picture memory>,
and VCRVAL = <integer zone blanking variable>.

The GARVAL parameter locates the graphics display memory, and the VCRVAL parameter directs which of the character and graphics zones of the 8510 are to be visible. When using UNITWRITE to volume #3 the address of the second parameter is loaded into the GAR, and the third parameter is loaded directly into the VCR. Thus, any of the bits in the VCR can be changed by placing the decimal representation of the bits into the third parameter of a UNITWRITE call to volume #3. VCR values from 0 to 63 cover all combinations of graphics and character zone blanking. Addresses loaded into the GAR must always be on even (integer) boundaries, and may be indexed from the array base. The following illustrate different effects of the UNITWRITE parameters:

```
UNITWRITE(3,SCREEN,63);
```

Display 3 (all) zones of graphics from picture memory in SCREEN, and display 3 (all) zones of the character display.

```
UNITWRITE(3,SCREEN,56);
```

Display 3 (all) zones of graphics from picture memory in SCREEN, and blank all zones of the character display.

```
UNITWRITE(3,SCREEN,49);
```

Display upper two zones of graphics from picture memory in SCREEN, and lower one zone of the character display.

```
UNITWRITE(3,SCREEN.INTS[1600],19);
```

Display middle one zone of graphics from picture memory in SCREEN, starting at SCREEN[3200] thru SCREEN[4799], and lower one zone of the character display. The upper display zone is blanked. Note that the GAR must be directed to the virtual starting address of the upper zone, although it and other zones may be blanked.

```
UNITWRITE(3,I,263); UNITWRITE(3,I,63);
```

Blank all graphic display zones, unblank all character display zones, and sound a 'click' at the display by toggling the state of the Audio bit in the VCR. In this case, 'I' is a dummy second parameter.

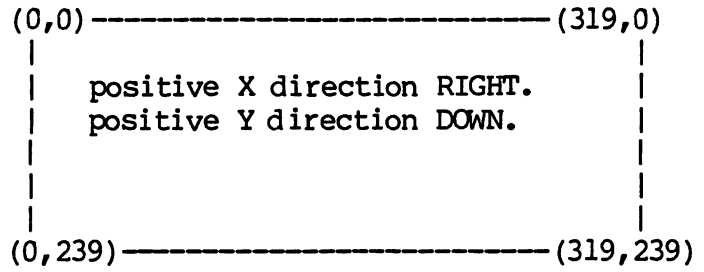
7.2 Graphics Procedure Calls

The Procedures DRAWLINE and DRAWBLOCK are provided by UCSD. The Procedures DRWBLK, GCHAR, GMARK, and THROTTLE are provide by Terak. All procedures are contained in SYSTEM.LIBRARY and must be declared EXTERNAL before use.

```
******************  WARNING  **************************
** These graphics procedures do no range checking on.  **
** parameters.  If parameters passed to the procedures  **
** are 'out of bounds' the procedures will produce      **
** unexpected results -- most likely, destruction of    **
** the user program, or operating system.               **
*********************************************************
```

7.3 DRAWLINE, DRAWBLOCK, and DRWBLK Conventions

The Coordinate System used by DRAWLINE, DRAWBLOCK, and DRWBLK fixes the point (0,0) in the upper left portion of the display. X and Y locations of the screen should be addressed using the following scheme.



7.4 DRAWLINE

This procedure draws lines in one of five modes, into memory. Note that the ROWWIDTH parameter indicates the width of the picture space, and is not necessarily restricted to the standard screen width. Picture space widths must be on integer boundaries; thus the parameter indicates the multiples of 16 bits of width required. Drawing into reduced width pictures is useful to prepare a subpicture for transfer by DRAWBLK, which also has a width parameter. In all DRAWLINE calls, the starting bit is not affected by the line. RADAR mode will return the number of steps from the starting point to the nearest obstacle (bits set) along the line, into RANGE.

```
PROCEDURE DRAWLINE(
      VAR RANGE : INTEGER; {returns result of radar scan
                            when PENSTATE=4}
      VAR SCREEN: TERAKSCREEN; {graphics memory}
          ROWWIDTH,          {# of 16 bit words per row,
                              typically 20 }

          XSTART,            {beginning X point of line}
          YSTART,            {beginning Y point of line}
          DELTAX,            {distance to move in X}
          DELTAY,            {distance to move in Y}
          PENSTATE:INTEGER
             ); EXTERNAL;
```

PENSTATE	ACTION	
0	PENUP	no change in picture
1	PENDOWN	force bits on
2	ERASE	force bits off
3	COMPLEMENT	reverse bits
4	RADAR	scan for obstacle, no change in picture

7.5 DRAWBLOCK

This procedure will do a two-dimension oriented transfer of bits, from a source block into a target block. The source and destination block must be of the same width and height, but may be located at any bit location within the same or different picture memory spaces. Different picture memory spaces are allowed to have different widths. The effect which the source block has upon the target block is controlled by the mode parameter. Complement mode is useful to overlay a picture with a block image, and then erase it while restoring the original picture contents. Graphics animation typically makes use of Complement mode. NOTE: DRAWBLK calls which overlap the source and target blocks should be approached with caution. Note also that row widths are given in bits, not words (as in DRAWLINE), and must be a multiple of 8.

```
CONST
   SRCXSIZE = {# of bits in source x direction.
               Use ((multiple of 8)-1)}
   SRCYSIZE = {number of bits in source y direction}
   TGTXSIZE = 319; {320 bits in x when target is TERAKSCREEN}
   TGTYSIZE = 239; {240 bits in y when target is TERAKSCREEN}

TYPE
   TERAKSCREEN = PACKED ARRAY[0..TGTYSIZE] OF
                      PACKED ARRAY[0..TGTXSIZE] OF BOOLEAN;
   SRC         = ARRAY[0..SRCYSIZE] OF
                      PACKED ARRAY[0..SRCXSIZE] OF BOOLEAN;

PROCEDURE DRAWBLOCK(VAR SOURCE : SRC;     {source block}
   SRCROW,          {#bits/row of block,multiple of 8}
   SRCX,            {x start location of source}
   SRCY  :INTEGER; {y start location of source}
   VAR DEST   :TERAKSCREEN; {Destination block}
   DSTROW,          {#bits/row of dst block,multple of 8}
   STX,             {x start location of destination}
   DSTY,            {y start location of destination}
   CNTX,            {number of bits to move x direction}
   CNTY,            {number of bits to move y direction}
   MODE  :INTEGER   {see below}
   ); EXTERNAL;
```

DRAWBLOCK MODE	ACTION
0	tgt := src {replace}
1	tgt := not (src){complement & overlay}
2	tgt := src XOR tgt{eraseable overlay}
3	tgt := src OR tgt{overlay}

NOTE1: The call interface and modes are different from the I.4 implementation of DRAWBLOCK. If you are converting programs from I.4 to II.0 either change mode parameters, or use the DRWBLK procedure provided below.

NOTE2: When using DRAWBLOCK or DRWBLK for animation the intrinsics UNITWAIT and UNITWRITE on volume #3 perform syncronization with vertical retrace of the video display (every 60th of a second). This is useful to pace the changes to the screen, maintaining uniform intensity of animated features.

7.6 DRWBLK

DRWBLK is provided for use in converting programs from I.4 to 1.5 or II.0. If you are beginning new development, use DRAWBLOCK above, as it performs the same function as DRWBLK in a more general fashion. In particular, note that DRWBLK requires that the source block be on an even (integer) boundary, while DRAWBLK is completely general. Also, the Mode parameter differs in values from the two procedures.

To convert I.4 programs to I.5 or II.0 include the following external declaration for DRWBLK then change every occurence of DRAWBLOCK to DRWBLK in the program.

```
PROCEDURE DRWBLK( VAR SOURCE:SRC;  {source block}
        VAR SCREEN:TERAKSCREEN; {target block}
            ROWSIZE,    {always 20}
            STARTX,     {start x for target}
            STARTY,     {start y for target}
            SIZEX,      {number of bits to move in x}
            SIZEY,      {number of bits to move in y}
            MODE : INTEGER {see below}
            ); EXTERNAL;
```

DRWBLK MODES	ACTION
0	tgt := tgt OR src
1	tgt := src
2	tgt := not (src)
3	tgt := tgt XOR src

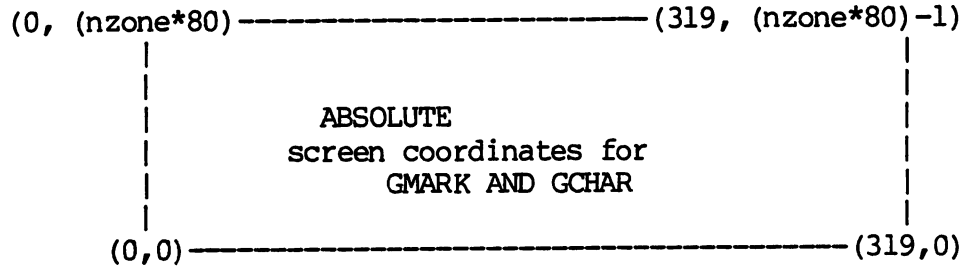
7.7 GCHAR & GMARK

The following routines GMARK & GCHAR support graphics on the 8510 by drawing characters and markers in the graphics space.

Both routines address the screen in absolute screen coordinates with (0,0) defined as the lower left corner of the screen. Note that this is a different addressing convention from that of DRAWLINE or DRAWBLK.

Both routines will support a picture memory height smaller, equal to, or larger than the actual display height (as controlled by the VCR zone blanking. The y dimension must, however, be a multiple of 80 (i.e 1/3 screen or the equivalent of a single screen zone. The parameter NZONE conveys the the picture memory height to the procedures.

The screen dimension in the x direction is always 0..319.



Linestyle for both routines is 0 for white (set bits on), 1 for black (clear bits out--erase). Neither routine supports XOR or COMPLEMENT mode.

Character patterns for GCHAR are derived from an 8 dot wide by 10 dot high template, which is fetched from the 8510 writeable character generator. The HEIGHT and WIDTH parameters to GCHAR define how many templates high and wide the target character block will be. Thus a call to GCHAR with the parmeter values h=3 and w=2 would create a character in the graphics space which is 30 dots high and 16 dots wide. The X, Y coordinates locate the lower left corner of the target block.

```
PROCEDURE GCHAR( VAR:
      SCREEN ARRAY:  POINTER TO ARRAY USED AS SCREEN,
       NZONE       :  INTEGER, {NUMBER OF ZONES TO DRAW ON}
       ORD(CHAR)   :  INTEGER, {Character to print}
       X           :  INTEGER,   {RANGE 0<=X<=319}
       Y           :  INTEGER,   {RANGE 0<=Y<=(NZONE*80-1)}
       HEIGHT      ;  INTEGER,
       WIDTH       ;  INTEGER,
       LINESTYLE   :  INTEGER);   EXTERNAL;
```

7.8 GMARK

This routine draws a 7 dot wide by 7 dot high marker, into the graphics picture memory. The pattern of the marker is controlled by the parameter MN. The marker will be centered on the screen location X,Y. If the marker would lie outside the clipping boundary defined by [XLEFT..XRIGHT] and [YBOT..YTOP] then the marker will be trimmed to fit the boundary.

The following conditions are expected to be true. Violation of these conditions will result in unpredictable results.

```
                    0<=X<=319
                  0<=Y<=NZONE*80-1
                XLEFT <= X <= XRIGHT
                 YBOT  <= Y <= YTOP
```

```
PROCEDURE GMARK(  SCREEN: ARRAY FOR SCREEN DISPLAY
       NZONE : INTEGER,  # OF 1/3 ZONES OF SCREEN
       X     : INTEGER,  X LOCATION OF MARKER
       Y     : INTEGER,  Y LOCATION OF MARKER
       MN    : INTEGER,  MARKER NUMBER 0<=MN<=7
       AXL   : INTEGER,  XLEFT OF WINDOW TO CLIP MARKER
       AXR   : INTEGER,  XRIGHT OF WINDOW TO CLIP MARKER
       AYB   : INTEGER,  YBOTTOM TO CLIP MARKER
       AYT   : INTEGER,  YTOP TO CLIP MARKER
       LSTY  : INTEGER,  LINESTYLE FOR PEN: 0 IS WHITE, 1 BLACK);
    EXTERNAL;
```

7.9 THROTTLE

This procedure provides rudimentary time control. Control will return to the calling program when the indicated time, in ticks of the line frequency clock, has passed.

```
PROCEDURE THROTTLE(TICKS:INTEGER); EXTERNAL;
```

APPENDIX A

A.1 SIGGRAPH CORE-79 COMPATIBILITY

The Graphics Extensions package provided by Terak for UCSD Pascal Version II.0 is a derivative of the basic output, no input, 2D proposed standard as developed by the ACM/SIGGRAPH Graphic Standard Planning Committee. Not all features of the SIGGRAPH CORE - 79 standard have been implemented. However, those features which have been implemented comply with the functional and semantic specifications for that feature. Even though all features of the standard have not been implemented, the user is still provided with a powerful, functional, integrated graphics library. The complete proposed graphics standard is contained in:

Computer Graphics
A Quarterly Report of SIGGRAPH-ACM
Vol 13, Number 3, August 1979
Status Report of the Graphics Standards
Planning Committee

Copies may be purchased by contacting:

ACM
P.O. Box 12105
Church Street Station
New York, New York 10249

The proposed standard has been specifically designed to avoid the limitations of computer language binding. Language binding considerations have been left to the implementor. The approach taken in the Terak implementation has been to apply generally accepted Pascal programming techniques. All capabilities have been implemented using the UCSD Pascal Version II.0 subroutine call with conventional subroutine argument sequences. The routine names have been compressed to meet UCSD Pascal Version II.0 limitations, while still maintaining a high degree of readability.

The Terak implementation of the CORE-79 standard is compatible with output level 1: Basic Output, input level 1: no Input and dimension level 1: 2D, as described below. It should be noted that the only error checking that exists is that which is provided by UCSD Pascal.

A.1.1 Output Primitives

The output primitives, as summarized in the CORE-79 standard, Appendix A, except for:

```
POLYLINE_ABS
POLYLINE_REL
POLYMARKER_ABS
POLYMARKER_REL
INQUIRE_TEXT_EXTENT_2
```

are supported. The "POLY" routines are not supported because of existing limitations in the UCSD Pascal compiler. They provide a functional implementation of the features defined for each procedure as defined in Section 2 of the standard.

A1.2 Picture Segmentation and Naming

No support is provided for explicit temporary picture segmentation and naming as described in Section 3 of the standard.

A1.3 Attributes

Only a subset of the attributes, as specified in Section 4 of the standard are supported. They are:

SET_LINESTYLE	(SET_LNST)
SET_CHARSIZE	(SET_CHSZ)
SET_CHARSPACE	(SET_CHSP)
SET_CHARPRECISION	(SET_CHPR)
SET_MARKER_SYMBOL	(SET_MKSM)

Some additional attribute values have been provided which are defined as implementation dependent and exceed the minimum standard requirements. The user is referred to the appropriate discussion in Section 4.4 of this document for additional details.

A1.4 Viewing Operations and Coordinate Transformations

Two of the viewing and coordinate procedures, as defined in Section 5 of the standard, are supported. These are:

SET_WINDOW	(SET_WNDW)
SET_VIEWPORT	(SET_VPRT)

The SET_VPRT procedure functions as if the default viewport specification were the entire Normalized Device Coordinate space, (as set by a nonsupported graphics procedure SET_NDC_SPACE 2). All procedures in the Terak implementation function as if SET_WINDOW_CLIPPING were ON.

A1.5 Control

Currently, none of the graphics control procedures, as defined in Section 7 of the standard, are supported by the Terak implementation. The functional intent of that section has, however, been provided in three procedures: INIT_GRF (graphics initialization), DRAW_VS and DISP_VS. These procedures provide control of the graphics display space for the Terak implementation. It should be noted that these procedures are not a part of the CORE-79 standard. They do however, provide access to the full power of the Terak 8510 graphics hardware. Support of these specialized hardware features is not part of the standard.

A1.6 Future Considerations

Future release of this graphics library will continue support of the CORE-79 standard and will move towards a greater degree of compliance and compatiblity.

Reserved

APPENDIX B

B.1 TERAK 8510 GRAPHICS DISPLAY PROCESSOR

The graphics display is presented on the 8532 Video Monitor as a 320 dot wide by 240 dot high matrix. The graphics display, when active, illuminates or blanks each dot according to the contents of a memory buffer. Sixteen hundred words (16 bits per word) are required for each third (zone) of the screen, or forty eight hundred for an entire 320 by 240 dot display. This can be reduced, with simultaneous reduction of the graphics area of the monitor displayed, by the zone blanking feature. Each zone of the graphics display may be blanked or displayed independently.

The character display is presented on the Video Monitor as a matrix 80 characters wide by 24 characters high. Like the graphics display, the character display is divided into three zones, which can be selectively blanked.

The display presented to the user is a Video overlay of the graphics and character displays. Except for the conventions established by this and other system software, the two displays are independent. Each graphics zone is 80 dots high by 320 dots wide, and each character zone is 8 characters high by 80 characters wide. Blanking of any one character display zone is mutually independent of the blanking of any one graphics display zone.

Reserved

APPENDIX C

C1. TERAK SIGGRAPH GRAPHICS LANGUAGE CROSS-REFERENCE

FUNCTION	BASIC	FORTRAN	Pascal
INIT_GRAPHICS	X	X	X
USE_SURFACE	X(DIM)	X	X
DRAW_ON_VIEW_SURFACE	X	X	X
DISPLAY_VIEW_SURFACE	X	X	X
SET_WINDOW	X	X	X
SET_VIEWPORT	X	X	X
SET_LINESTYLE	X	X	X
NEW_FRAME	X	X	X
INQUIRE_CURRENT_POSITION	X	X	X
MOVE_ABS	X	X	X
MOVE_REL	X	X	X
LINE_ABS	X	X	X
LINE_REL	X	X	X
POLYLINE_ABS	X	X	
POLYLINE_REL	X	X	
SET_MARKER_SYMBOL	X	X	X
MARKER_ABS	X	X	X
MARKER_REL	X	X	X
POLYMARKER_ABS	X	X	
POLYMARKER_REL	X	X	
SET_CHARACTER_PRECISION	X	X	X
SET_CHARACTER_SIZE	X	X	X
SET_CHARACTER_SPACE	X	X	X
TEXT	X	X	X

"BASIC" above refers to both MUBASIC and BASIC-11.

"Pascal" above refers to UCSD Pascal Version II.0.

Reserved

APPENDIX D

D1. ANALOGY OF THE GRAPHICS PROCESS

Figure D-1 illustrates each stage of the conversion process to a star drawn by a user program to the resulting image displayed on the screen. The window is set using World Coordinates. The top point of the star is lost here because it lies outside the Window. The Viewport is set in Normalized Device Coordinates. In this example, the Viewport maps the Window onto the entire View Surface, which has been defined to be three zones. The View Surface is then displayed on the entire screen.

Figure D-2 gives a more complex example of the same process. The top point of the star is again lost because it lies outside the Window. The Viewport maps the Window onto a portion of the View Surface, which has been defined to be five zones. In this example, the physical memory for the graphics display has been explicitly defined, and consists of seven zones. The View Surface is contained in the first five of these seven. An offset of -2 has been used in mapping the View Surface onto the physical display, which results in the use of zones 3 through 5 of the View Surface for this purpose. This process eliminates the entire top section of the star. All three character zones are displayed, but only the upper graphics zone, while the middle and lower zones are blanked. This results in the loss of the bottom two points of the star from the final display.

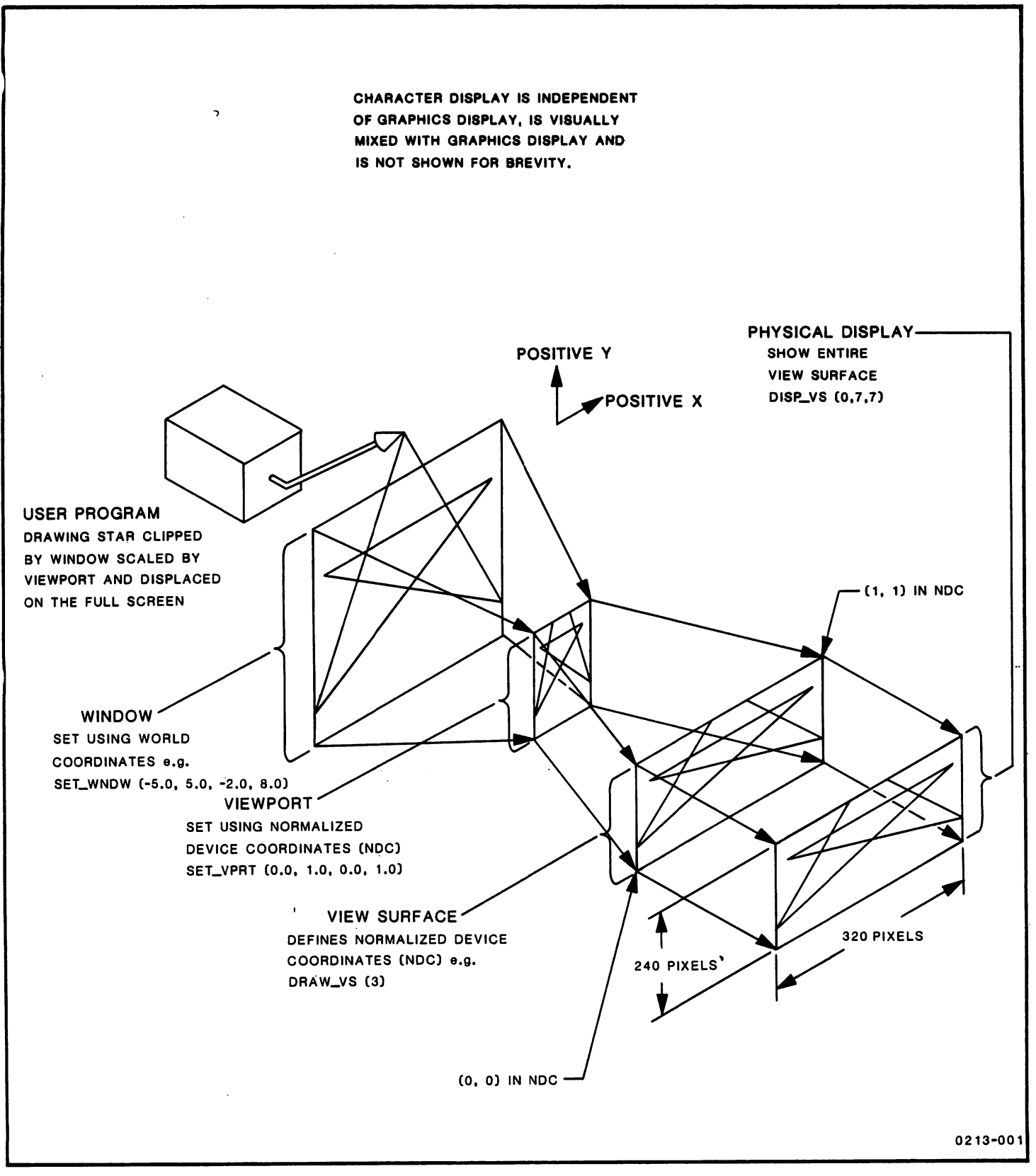


Figure D-1 Analogy of the Graphics Process

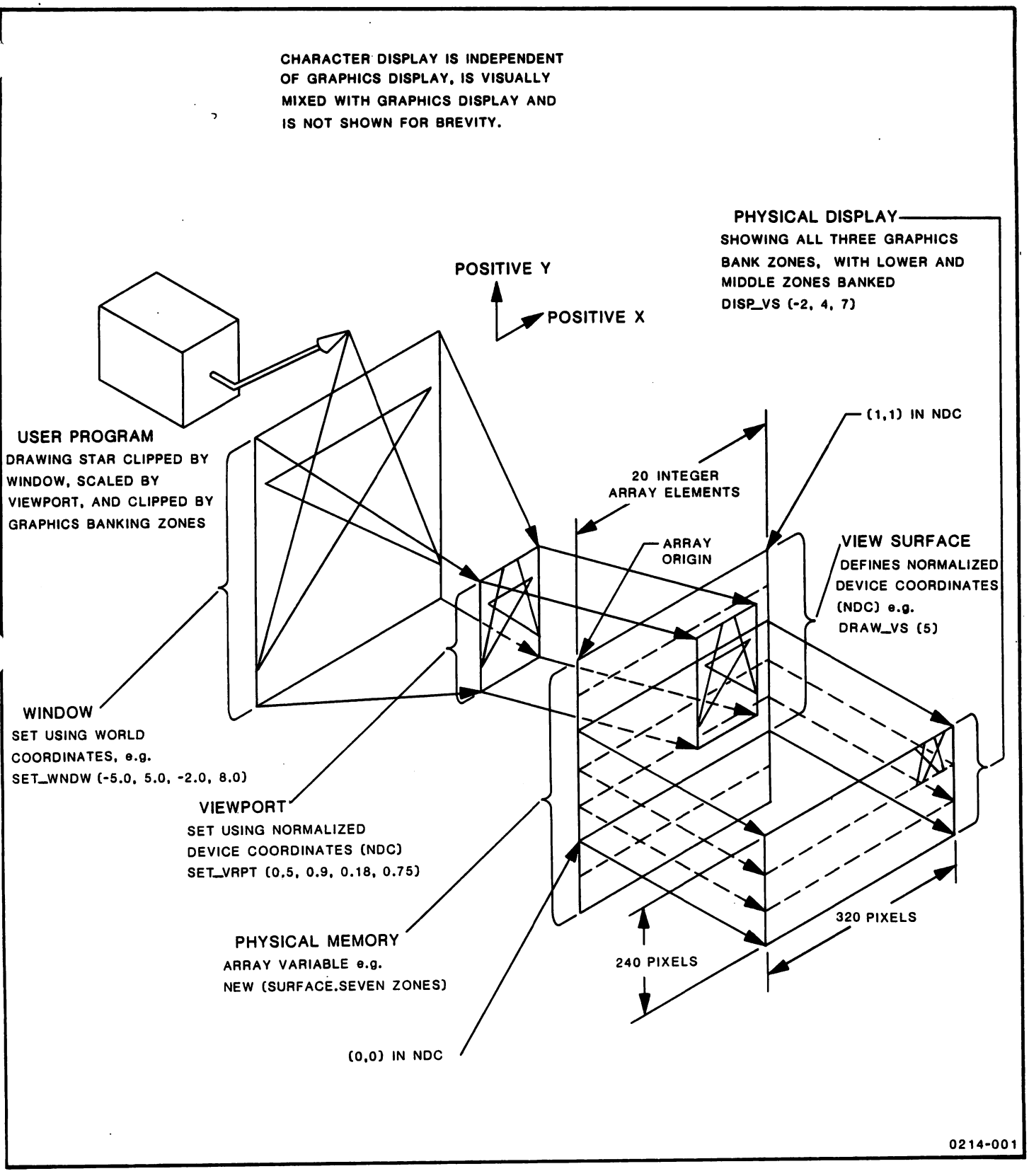


Figure D-2 Extended Analogy of the Graphics Process

Reserved