V I R T U A L G R A P H I C S L I B R A R Y

V G L

R E F E R E N C E M A N U A L

FEBRUARY, 1982 REV. A

TABLE OF CONTENTS

1.0 INTRODUCTION

The Virtual Graphics Library (VGL) is an enhancement Library for the MTU BASIC Language, which provides a powerful 2-dimensional graphics capability. The VGL is exceptionally easy to use because the programmer does not need to be concerned with the details of the physical operation of the bit-mapped pixel display. Instead the programmer can concentrate on the data in his own terms (often called "world" or "virtual" coordinates). All the transformations needed for the physical plotting are done automatically by the VGL Library.

An important concept in using the VGL Library is that of the window. This is illustrated conceptually in FIGURE 1. This figure shows the area of displayable coordinate whose limits are determined by the floating point range in MTU BASIC. It is assumed that the data or image you wish to display falls somewhere within this area. The window defines the portion of the area of displayable coordinates you wish to display on the screen. The window may be defined so that all, or only a selected portion of your data or image is displayed. The desired window is set using the WINDOW command.

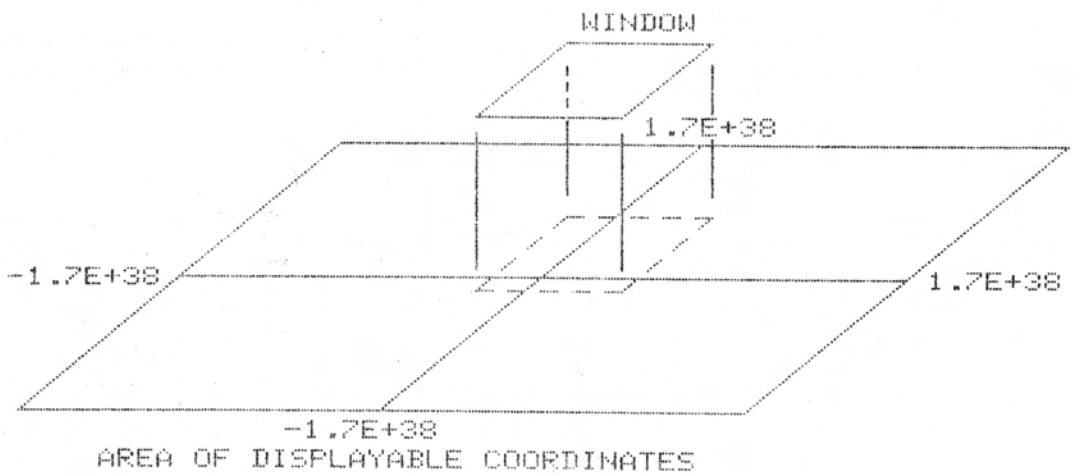

Figure 1. Illustration of Window Concept

Another command which affects the screen display is the VIEWPORT command. The viewport is the portion of the display screen which is to display the data bordered by the window. By setting the viewport you can control the size and position of the window on the screen.

The WINDOW and VIEWPORT commands allow you to set up the most ideal display for each particular application. Once done, the remainder of the VGL commands work with the coordinate values suited to the application. The purpose of all this is to make displaying your data or image as simple as possible.

There is one other concept which is important in using the VGL Library. This is the concept of the graphics cursor position. The VGL Library has the ability to "remember" the last point drawn or moved to. Many of the commands will use this point as a starting point for a new line or other operation. For purposes of discussion this "remembered" point is called the graphics cursor.

2. LOADING INSTRUCTIONS

To load the VGL Library, simply include the name "VGL" as a file name in a LIB command. Since the VGL is designed to work in conjunction with the IGL you will normally load both of them. When you do, you should place the "VGL" name prior to the "IGL" name in the LIB command, such as in this example:

```
LIB "VGL","IGL"
```

This order is necessary if you wish to use all commands in both libraries. If they are loaded the reverse order, some of the commands in the IGL Library will give SYNTAX ERRORs. These IGL commands will be the ones which have a matching command in the VGL, only without the "S" prefix. An example would be the SMOVE command which has a matching MOVE command in the VGL Library.

If the VGL and IGL Libraries are not on the default drive (usually 0), then you may specify a drive number as shown in the following example:

```
LIB "VGL:1","IGL:1"
```

which loads the both Libraries from drive 1.

If you wish, you may load the VGL Library by itself. In this case, you should make sure you will not be needing any of the commands in the IGL Library (such as PENMODE, LABEL, etc.).

3. GETTING STARTED WITH THE VGL LIBRARY

To help get you started using the VGL Library, here is a short "hands on" demo which shows how a sine function might be plotted. Naturally the first step is to get MTU BASIC started, once CODOS is booted up. To start MTU BASIC, simply execute the command:

```
BASIC
```

Once the "READY." message appears on the screen, execute the BASIC command:

```
LIB "VGL","IGL"
```

This will make all of the VGL and IGL commands available to BASIC. Now we can start writing the demo program. First we must decide how much of the sine function to plot and how many line segments we will be using in plotting it. In this demo we will plot two cycles using 50 line segments. Using this information, and knowing that the value of the sine function will vary from -1 to +1, we can enter the following statements into the program:

```
10 PI=3.1416: NS=50 :REM NS = NUMBER OF SEGMENTS
20 WINDOW 0,4*PI,-1,1
```

We will use the default viewport which occupies all of the screen execpt the function key legend boxes. Next we need a statement to clear the screen. Enter the next statement to clear the screen:

```
30 CLEARVP
```

To plot the function we will use a FOR/NEXT loop. Enter the following statements which plot the sine function:

```
60 MOVE 0,0: REM MOVE TO STARTING POINT
70 FOR X=0 TO 4*PI STEP 4*PI/NS
80 DRAW X,SIN(X)
90 NEXT X
```

Note that the plotting is simple because we are able to plot the problem data directly without having to scale to the physical plotting units of the screen. Now execute the following command to run the program:

```
RUN
```

Assuming the statements above were entered correctly, we should now have two cycles of a sine wave plotted in the default viewport as shown in Figure 2. To make the program a little more complete, we can draw horizontal and vertical axes as well. To do this, add the following statements to the program:

```
40 XAXIS 0,4*PI/NS
50 YAXIS 0,.1
```

In these commands, the first argument specifies the location of the axis. The second argument specifies the interval between tic marks. Now RUN the program again to see the result of the axes commands as shown in Figure 3.

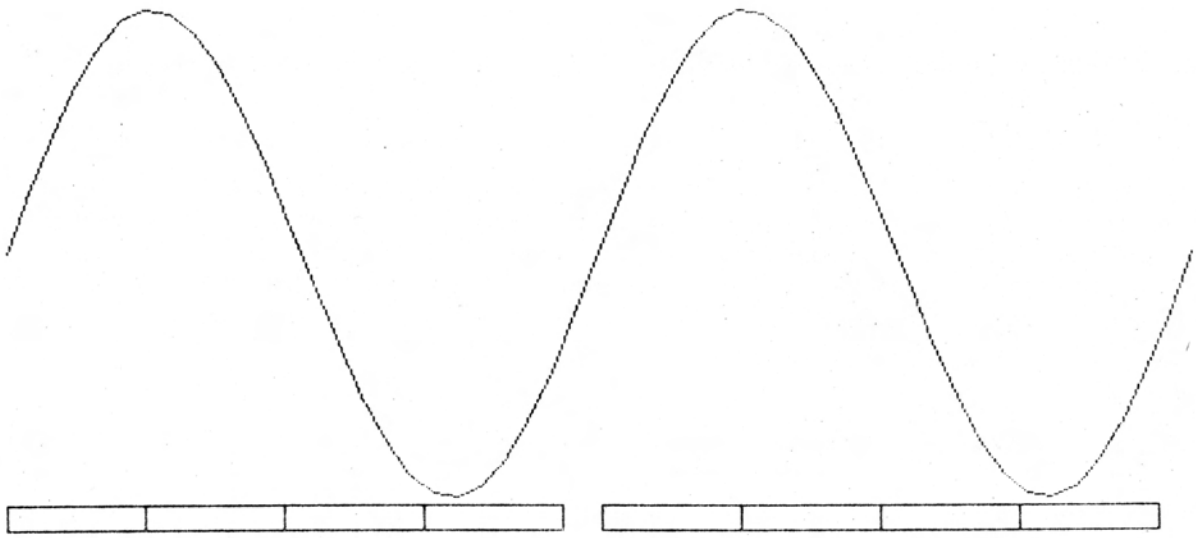

Figure 2. Sine Wave Plotted with VGL and MTU BASIC

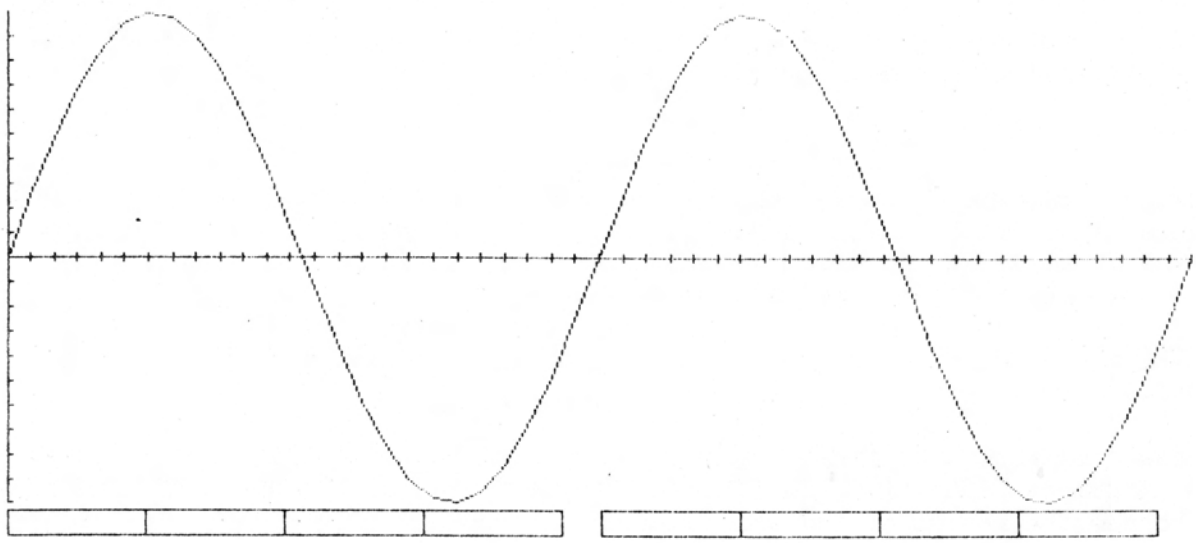

Figure 3. Sine Wave with Added Axis Statements

Now we can use this program to observe the effects of the VIEWPORT and window commands. To observe the effect of the VIEWPORT command, execute the following command:

```
VIEWPORT 100,300,50,200
```

This reduces the viewport boundaries from the default viewport size to the boundaries specified. RUN the program again and observe how its size is automatically reduced to fit within the new viewport as shown in Figure 4. The image is still the same, just reduced in size. You will note that some of the previous image is still on the screen. Since we reduced the size of the viewport, this portion is no longer cleared by the CLEARVP command in line 30.

Now clear the screen by entering a CTL-L.

To observe the effect of the window, suppose we wish to display only the first cycle of the sine function. To do this we don't need to change the statements which do the plotting (60-90). We only need to change the WINDOW command in line 20. To change line 20, enter the following statement to display only one cycle of the sine function.

```
20 WINDOW 0,2*PI,-1,1
```

Now RUN the program again and observe that only one cycle of the sine wave appears as shown in Figure 5. The other cycle is still being plotted, but it does not appear on the screen because its image falls outside the window. This process is called clipping. Another example of clipping is shown by executing:

```
MOVE 0,0: DRAW PI,2: DRAW 2*PI,0
```

Note that the two lines drawn do not lie entirely within the window. Both were clipped at the window boundary. Even though the lines were clipped, both proceed to the non-visible point (PI,2) outside the window. The portion of the lines that you see would not be any different had the lines been entirely within the window.

This tutorial demo has shown you a little about the function of the VIEWPORT and WINDOW commands, and how they can simplify the task of plotting data. It is beyond the scope of this reference manual to describe how various applications may be implemented. Hopefully with commands of the VGL Library at your disposal, you won't find it difficult to implement the applications you require. Don't forget, you also have the commands in the IGL Library available.

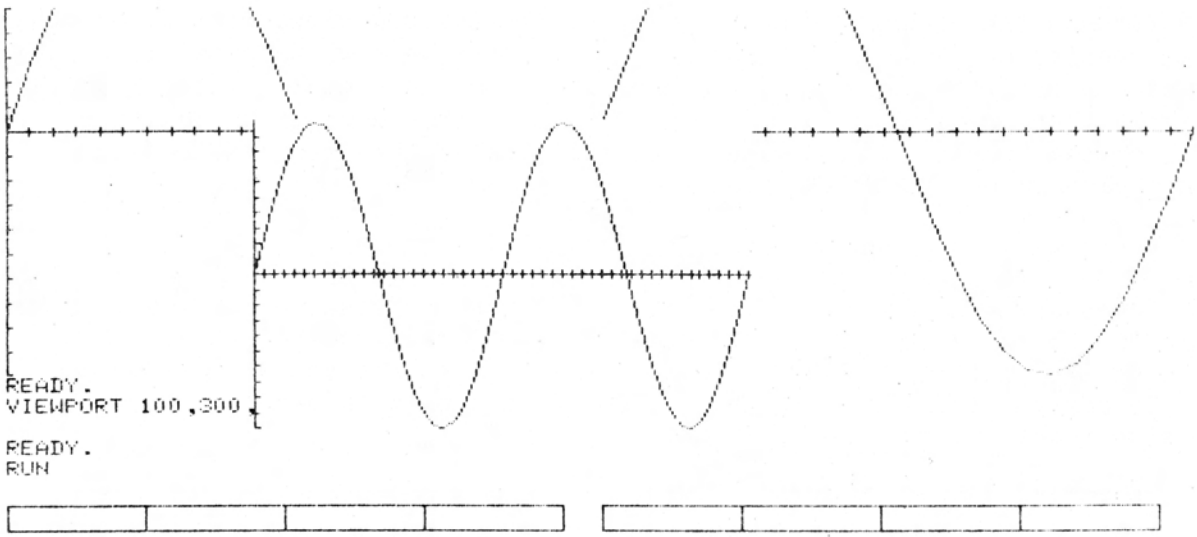

Figure 4. Effect of Reduced Viewport

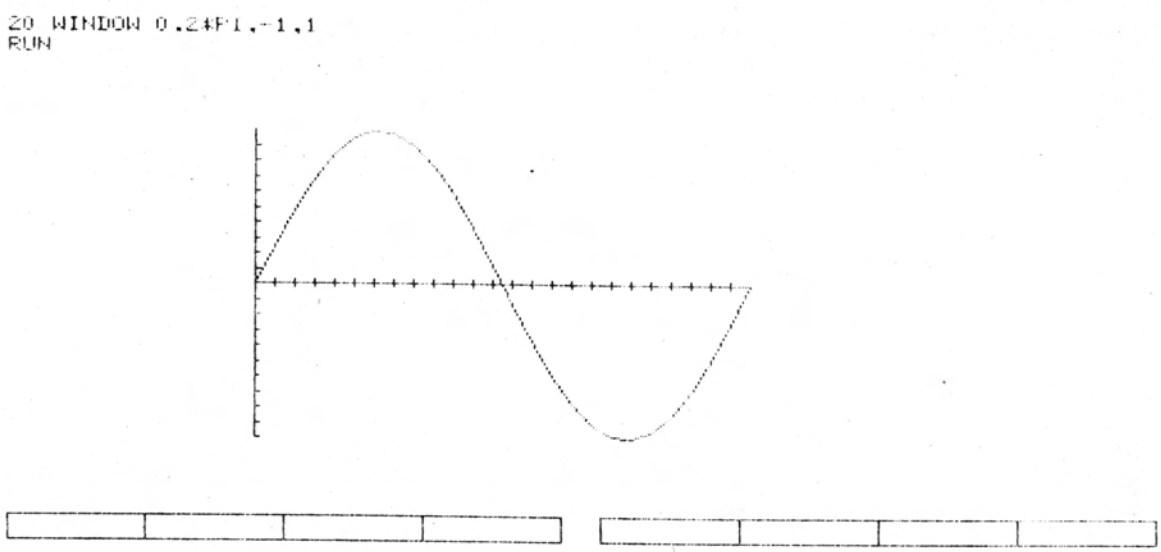

Figure 5. Effect of Reduced Window

4. VGL COMMAND DESCRIPTIONS

This section contains descriptions of each of the VGL commands, given in alphabetical order. Some of the examples given use IGL commands. So some make sure that both the VGL and IGL Libraries are loaded when trying these examples.

The first example given with each command may be executed as a direct command. If the default window and viewport are in operation, you will be able to observe the results of the command. For many of the commands, the results will vary depending on the position of the graphics cursor.

4.1 THE BOXVP COMMAND

PURPOSE: To draw lines around the perimeter of the current viewport.

SYNTAX: BOXVP

ARGUMENTS: none

DISCUSSION:

Often it is desirable to outline the current viewport with a box, thus revealing its size. The BOXVP command allows you to do this with a single command. When the command is executed, the current boundaries of the viewport are used to draw a rectangle around the viewport. The lines draw are part of the viewport itself, not one coordinate unit outside it.

EXAMPLES:

```
BOXVP
```

will draw a rectangle around the current viewport.

4.2 THE CLEARVP COMMAND

PURPOSE: Clear the current viewport.

SYNTAX: CLEARVP

ARGUMENTS: none

DISCUSSION:

Executing this command will clear the current viewport. This will also erase the lines drawn by a BOXVP command.

EXAMPLE:

```
CLEARVP
```

will clear the current viewport.

NOTES:

1. The CLEARVP command clears by erasing horizontal lines starting at the bottom of the viewport and moving up. If you wish to clear the entire screen, it would be faster to use the SCLEAR command of the IGL Library.

4.3 THE DASH COMMAND

PURPOSE: To draw a dashed line from the current graphics cursor to a point.

SYNTAX: DASH ⟨x-coord⟩ , ⟨y-coord⟩

ARGUMENTS:

⟨x-coord⟩ = an expression which gives the desired X coordinate.
⟨y-coord⟩ = an expression which gives the desired Y coordinate.

DISCUSSION:

The DASH command draws a dashed line from the current graphics cursor coordinates to the specified point. This specified point then becomes the new graphics cursor coordinates.

If either the current graphics cursor or the point specified in the DASH command falls outside the current window, automatic clipping of the line will occur. The clipping will be such that only the portion of the line passing through the window will be drawn. If no portion of the line passes though the window, the display will be unchanged.

The dashed pattern of the line is achieved by drawing the line in dashed line mode. This means that the dash pattern set by the DASHPAT command is in effect. For information on the various patterns that may be set, see the DASHPAT command in the IGL Reference Manual. For convenience, a few of these patterns are described in Note 3 below.

EXAMPLES:

```
DASH 75.5,75.5
```

will draw a dashed line from the current graphics cursor position to the coordinates 75.5,75.5. The default dash pattern will draw the line with 4 dots on followed by skipping 4 dots, etc.

```
DASHPAT 170,170: DASH I*DX,DT(I)
```

will draw a dashed line from the current graphics cursor position to the point with X coordinate of I*DX, and Y coordinate of DT(I). The dash pattern will be that of plotting every other dot (see Note 3).

NOTES:

1. The action of clipping a line will have no effect on the graphics cursor coordinates. However, it will have an effect on the current screen coordinates used by the IGL Library. For more information on how clipping affects the current screen coordinates, see Appendix C.

2. Executing the DASH command will have no effect on the drawing mode as set by the PENMODE command in the IGL Library.

3. Here are a few DASHPAT commands for some simple dotted line patterns.

```
DASHPAT 170,170   - plots every other dot
DASHPAT 204,204   - plots 2 dots and skips 2 dots
DASHPAT 240,240   - plots 4 dots and skips 4 dots (the default)
DASHPAT 255,0     - plots 8 dots and skips 8 dots
```

4.4 THE DRAW COMMAND

PURPOSE: To draw a solid line from current graphics cursor location to a point.

SYNTAX: DRAW ⟨x-coord⟩ , ⟨y-coord⟩

ARGUMENTS:

⟨x-coord⟩ = an expression which gives the desired X coordinate.
⟨y-coord⟩ = an expression which gives the desired Y coordinate.

DISCUSSION:

The DRAW command draws a solid line from the current graphics cursor location to the specified point. This specified point then becomes the new graphics cursor location.

If either the current graphics cursor location or the point specified in the DRAW command falls outside the current window, automatic clipping of the line will occur. The clipping will be such that only the portion of the line passing through the window will be drawn. If no portion of the line passes though the window, the display will be unchanged.

EXAMPLES:

```
DRAW 50.0,25.3
```

will draw a solid line from the current graphics cursor location to the coordinates 50.0,25.3.

```
DRAW I,COS(TH)
```

will draw a solid line from the current graphics cursor location to the point with X coordinate of I, and Y coordinate of COS(TH).

NOTES:

1. The action of clipping a line will have no effect on the graphics cursor location. However, it will have an effect on the current screen coordinates used by the IGL Library. For more information on how clipping affects the current screen coordinates, see Appendix C.

2. Executing the DRAW command will have no effect on the drawing mode as set by the PENMODE command in the IGL Library.

4.5 THE ERASE COMMAND

PURPOSE: To erase a line from the current graphics cursor location to a point.

SYNTAX: ERASE ⟨x-coord⟩ , ⟨y-coord⟩

ARGUMENTS:

⟨x-coord⟩ = an expression which gives the desired X coordinate.
⟨y-coord⟩ = an expression which gives the desired Y coordinate.

DISCUSSION:

The ERASE command erases a line from the current graphics cursor location to the specified point. This specified point then becomes the new graphics cursor location.

If either the current graphics cursor or the point specified in the ERASE command falls outside the current window, automatic clipping of the line will occur. The clipping will be such that only the portion of the line passing through the window will be erased. If no portion of the line passes though the window, the display will be unchanged.

The erasing is accomplished by drawing the line in erase mode. This mode calls for dots along the line to be set to the "off" state.

EXAMPLES:

```
MOVE 25.5,10.2: DRAW 100,50: ERASE 25.5,10.2
```

will first draw, then erase a line from 25.5,10.2 to 100,50.

```
ERASE DX(I),DY(I)
```

will erase a line from the current graphics cursor position to the point with X coordinate of DX(I), and Y coordinate of DY(I).

NOTES:

1. The action of clipping a line will have no effect on the graphics cursor location. However, it will have an effect on the current screen coordinates used by the IGL Library. For more information on how clipping affects the current screen coordinates, see Appendix C.

2. Executing the ERASE command will have no effect on the drawing mode as set by the PENMODE command in the IGL Library.

3. Since the ERASE command turns off all dots along the line, it may be used to draw black lines on a white background (reverse video).

4.6 THE FILL COMMAND

PURPOSE: To fill a rectangular area according to the current drawing mode.

SYNTAX: FILL ⟨xmin⟩ , ⟨xmax⟩ , ⟨ymin⟩ , ⟨ymax⟩

ARGUMENTS:

⟨xmin⟩ = an expression for the X coordinate of the left edge of the rectangle.
⟨xmax⟩ = an expression for the X coordinate of the right edge of the rectangle.
⟨ymin⟩ = an expression for the Y coordinate of the bottom edge of the rectangle.
⟨ymax⟩ = an expression for the Y coordinate of the top edge of the rectangle.

DISCUSSION:

The FILL command "fills" a specified rectangle. The "filling" occurs by drawing horizontal lines starting at the bottom of the rectangle and moving up one pixel as each line is drawn. This continues until the top of the rectangle is reached.

Each of the horizontal lines is drawn using the current drawing mode. Depending on the drawing mode, the FILL command can have different effects. If the draw solid line mode is set (PENMODE 1), all the dots in the rectangle will be set to the "on" state. If the erase line mode is set (PENMODE 2), the rectangle will be cleared. If the dashed line mode is set (PENMODE 5), the pattern in the rectangle will vary depending on the width of the rectangle and the dash pattern.

The arguments specified in the FILL command may be any legal floating point values. However, if any portion of the specified rectangle lies outside the window, only the area within the window will be "filled".

EXAMPLES:

```
FILL 0,10.8,0,25.2
```

will "fill" the rectangle with corners at 0,0; 10.8,0; 10.8,25.2; and 0,25.2.

```
FILL X,X+DX,Y,Y+DY
```

will "fill" the rectangle with a corner at coordinate at X,Y; with width and heigth of DX and DY, respectively.

NOTES:

1. The FILL command permits one exception to the syntax given above. It permits the ymin and ymax arguments to be switched. If the first argument is found to be greater than the second argument, the horizontal lines will be drawn starting at the top of the recangle and will proceed downward.

2. Using the FILL command will have no effect on the current graphics cursor location, or the current screen coordinates used by the IGL Library.

4.7 THE FLIP COMMAND

PURPOSE: To draw a line from the current graphics cursor location to a point using the flip line mode.

SYNTAX: FLIP <x-coord> , <y-coord>

ARGUMENTS:

<x-coord> = an expression which gives the desired X coordinate.
<y-coord> = an expression which gives the desired Y coordinate.

DISCUSSION:

The FLIP command draws a line in flip mode from the current graphics cursor location to the specified point. This specified point then becomes the new graphics cursor location.

If either the current graphics cursor location or the point specified in the FLIP command falls outside the current window, automatic clipping of the line will occur. The clipping will be such that only the portion of the line passing through the window will be drawn. If no portion of the line passes though the window, the display will be unchanged.

Drawing the line in flip mode causes each dot along the line to be flipped to the opposite of its current state. Dots which are currently "off" will be turned "on", and vice versa. The advantage of drawing in flip mode is that flipping the same line twice leaves the display unchanged. In the case where lines must added and later removed from the display, using flip mode can keep the fixed part of the display from being changed in the process.

EXAMPLES:

```
FLIP 125.8,10.5
```

will draw a line in flip mode from the current graphics cursor location to the coordinates 125.8,10.5.

```
FLIP DB(I),DB(J)
```

will draw a line in flip mode from the current graphics cursor position to the point with X coordinate of DB(I), and Y coordinate of DB(I).

NOTES:

1. The action of clipping a line will have no effect on the graphics cursor location. However, it will have an effect on the current screen coordinates used by the IGL Library. For more information on how clipping affects the current screen coordinates, see Appendix C.

2. Executing the FLIP command will have no effect on the drawing mode as set by the PENMODE command in the IGL Library.

4.8 THE FLIPVP COMMAND

PURPOSE: To flip the state of each point in the current viewport.

SYNTAX: FLIPVP

ARGUMENTS: none

DISCUSSION:

The FLIPVP command will flip each dot in the current viewport to the opposite of its current state.

EXAMPLE:

FLIPVP

will flip all the dots in the current viewport.

NOTES:

1. Executing the FLIPVP command will have no effect on the current graphics cursor position, or the position of the current cursor coordinates used in the IGL Library. It also will have no effect on the current drawing mode.

4.9 THE GRIN COMMAND

PURPOSE: To input an X,Y coordinate pair using the GRIN cursor routine.

SYNTAX: GRIN ⟨string variable⟩ , ⟨x-variable⟩ , ⟨y-variable⟩

ARGUMENTS:

⟨string variable⟩ = any string variable. This variable will receive the character used to terminate the input.
⟨x-variable⟩ = any numeric variable which is to receive the X-coordinate.
⟨y-variable⟩ = any numeric variable which is to receive the Y-coordinate.

DISCUSSION:

When a GRIN command is executed, a pulsating crosshair the full width and height of the display will appear on the screen. The intersection of the crosshair identifies the point being selected. The crosshair intersection may be "steered" to any point on the screen by using the four cursor controls. Holding down the "left-arrow" key moves the intersection to the left, right-arrow key moves it to the right, etc. The speed of movement can be increased by holding down the SHIFT key in conjunction with the cursor key. Once the intersection has been positioned over the desired point, depressing any non-cursor key will make the crosshair disappear. This non-cursor key will be returned to the specified string variable as a one character string. The X and Y coordinates of the selected point will be stored in their respective variables. It is important to note that the returned coordinates will be in the users virtual coordinate space, not screen coordinates.

The GRIN command allows you to select any point on the display screen, <u>including points which lie outside the current viewport</u>. If the point is outside the viewport, the proper coordinates will still be returned. However, this point will necessarily be outside the current window, and clipping will occur if you try to draw to the point.

EXAMPLES:

```
GRIN T$,X,Y: PRINT T$,X,Y
```

will execute the GRIN cursor routine. Positioning the crosshair and hitting the "D" key will exit the GRIN cursor routine. A "D" followed by the coordinates of the selected point will be printed. If the current window and viewport are at their defaults, positioning the crosshair on the upper right corner of the left legend box will return the coordinates 46.7640919,-.836820084. The Y coordinate is minus because the point is just below the viewport.

NOTES:

1. The initial position of the crosshair intersection will be at the current screen coordinates used by the IGL Library. Normally this will correspond to the graphic cursor position. However in some cases this will not be the case, such as when the graphics cursor is not within the current window.

2. The vertical line of the pulsating crosshair may appear "broken" when moving, especially at the higher speed. This "break" appears because the sweep of the CRT catches the display with part of the "new" line and part of the "old" line drawn.

4.10 THE LTPEN COMMAND

PURPOSE: To input an X,Y coordinate pair using the light pen.

SYNTAX: LTPEN <flag-variable> , <x-variable> , <y-variable>

ARGUMENTS:

<flag-variable> = a numeric variable which will receive the "hit flag".
<x-variable> = a numeric variable which is to receive the X coordinate.
<y-variable> = a numeric variable which is to receive the Y coordinate.

DISCUSSION:

The LTPEN command inputs graphics coordinates using the light pen. When executed, the command will wait up to 1/30 of a second for the light pen to see light. If light is seen, then the flag variable is set to 1, and the coordinates of the point where the light was seen are returned in the X and Y variables. If light is not seen, then the flag variable is set to 0, and the X and Y variables are left unchanged.

The LTPEN command allows you to select any point on the display screen, including points which lie outside the current viewport. If the point is outside the viewport, the proper coordinates will still be returned. However, this point will necessarily be outside the current window, and clipping will occur if you try to draw to the point.

Because of the field of view of the light pen, the coordinates returned for a selected point have a ± 2 pixel uncertainty. The amount of uncertainty this induces in the virtual coordinates returned will depend on the current viewport and window. This uncertainty may be computed by the following equations:

X-UNCERTAINTY = ± (XWNDMAX - XWNDMIN)/(XVPTMAX - XVPTMIN)*2

Y-UNCERTAINTY = ± (YWNDMAX - YWNDMIN)/(YVPTMAX - YVPTMIN)*2

where the variables refer to the minimum and maximum values specified for the window and viewport.

EXAMPLES:

```
LTPEN F,X,Y: IF F=1 THEN PRINT X,Y
```

will try to input a pair of coordinates using the light pen. If the light pen saw light, the coordinates of the selected point will be printed. If the veiwport and window are at their defaults, pointing the light pen at the upper left corner of the left legend box should return coordinates of approximately 46,-.84. Since the LTPEN command waits only a short period to see light, you should position the light pen before executing the command.

NOTES:

1. The actual time spent executing this command varies from a minimum of less than a millisecond to a maximum of 33 milliseconds. The exact time required depends on whether the pen sees light, and when the command was executed relative to the 16.6 millisecond display refresh cycle.

2. Clean face of CRT daily prior to first use of the light pen.

4.11 THE MOVE COMMAND

PURPOSE: To move the graphics cursor to a new location.

SYNTAX: MOVE <x-coord> , <y-coord>

ARGUMENTS:

<x-coord> = an expression which gives the desired X coordinate.
<y-coord> = an expression which gives the desired Y coordinate.

DISCUSSION:

The MOVE command sets VGL's graphics cursor location to the coordinates specified in the command. The coordinates specified may be any legal integer or floating point values. They need not refer to a point within the current window. If the specified point lies within the current window, the current screen coordinates used by the IGL Library will be updated to match location of the VGL's graphics cursor. If the point is not within the window the current screen coordinates are left unchanged.

EXAMPLES:

```
MOVE 100.8,50.3
```

will set the graphics cursor location to 100.8,50.3.

```
MOVE I,SIN(TH)
```

will set the graphics cursor X coordinate to the value of I, and the Y coordinate to the value of SIN(TH)

NOTES:

1. The MOVE command has no effect on the current drawing mode as specified by the PENMODE command of the IGL Library.

4.12 THE PEN COMMAND

PURPOSE: To draw from the current graphics cursor location to a point using the current drawing mode.

SYNTAX: PEN ⟨x-coord⟩ , ⟨y-coord⟩

ARGUMENTS:

⟨x-coord⟩ = an expression which gives the desired X coordinate.
⟨y-coord⟩ = an expression which gives the desired Y coordinate.

DISCUSSION:

The PEN command draws a line from the current graphics cursor location to the specified point using the current drawing mode. This specified point then becomes the new graphics cursor location.

If either the current graphics cursor location or the point specified in the PEN command falls outside the current window, automatic clipping of the line will occur. The clipping will be such that only the portion of the line passing through the window will be drawn. If no portion of the line passes though the window, the display will be unchanged.

The current drawing mode is the mode set by the PENMODE command in the IGL Library. For convenience, a list of these modes is given in Note 2 below.

EXAMPLES:

```
PEN 80.1,95.9
```

will draw a line using the current drawing mode, from the current graphics cursor position to the coordinates 80.1,95.9.

```
PEN I,SQR(HY)
```

will draw a line using the current drawing mode, from the current graphics cursor position to the point with X coordinate of I, and Y coordinate of SQR(HY).

NOTES:

1. The action of clipping a line will have no effect on the graphics cursor location. However, it will have an effect on the current screen coordinates used by the IGL Library. For more information on how clipping affects the current screen coordinates, see Appendix C.

2. Here is a list of the various drawing modes set by the PENMODE command in the IGL Library.

```
PENMODE 0  - move mode, no drawing takes place
PENMODE 1  - draw solid line mode
PENMODE 2  - erase line mode
PENMODE 3  - draw flip line mode
PENMODE 4  - move dashed mode, not a useful mode
PENMODE 5  - draw dashed line mode
PENMODE 6  - erase dashed line mode
PENMODE 7  - flip dashed line mode
```

4.13 THE POINT COMMAND

PURPOSE: To plot a point.

SYNTAX: POINT <x-coord> , <y-coord>

ARGUMENTS:

<x-coord> = an expression which gives the desired X coordinate.
<y-coord> = an expression which gives the desired Y coordinate.

DISCUSSION:

The POINT command plots the point at the specified coordinates. This specified point then becomes the new graphics cursor location.

If the specified point is within the current window, then the point is plotted by turning "on" the appropriate dot in the display. If the point is outside the current window, then the point is not plotted and the display remains unchanged.

EXAMPLES:

```
POINT 50.3,25.7
```

will plot the point at coordinates 50.3,25.7.

```
POINT I,COS(TH)
```

will plot the point with X coordinate of I, and Y coordinate of COS(TH).

NOTES:

1. If the specified point is plotted, then the current screen coordinates used in the IGL Library will reflect the location of this point on the screen. If the point is outside the current window, and thus is not plotted, then the current screen coordinates are left unchanged.

2. Executing the POINT command will have no effect on the drawing mode as set by the PENMODE command in the IGL Library.

4.14 THE QDOT FUNCTION

PURPOSE: To return the status of a particular point.

SYNTAX: QDOT(<x-coord> , <y-coord>)

ARGUMENTS:

<x-coord> = an expression which gives the desired X coordinate.
<y-coord> = an expression which gives the desired Y coordinate.

DISCUSSION:

It is important to note that QDOT() is a function, and must be used as part of an expression. When the QDOT() function is encountered in an expression, it returns a value of 0 or 1 depending on the current state of the specified point. If the point is "on", the returned value is 1. If it is "off", the returned value is 0.

EXAMPLES:

```
A=QDOT(89.6,22.1)
```

will test the state of the pixel at coordinates 89.6,22.1. If 89.6,22.1 lies outside the current window, then A will be set to -1. If the pixel is within the current window and is "on", then A will be set to 1. If the pixel is "off", A will be set to 0.

NOTES:

1. The QDOT function will have no effect on the current graphics cursor position, nor the current cursor location used by the IGL Library.

2. If QDOT is used to read back previously plotted points, the coordinates specified should be computed in exactly the same way they were when originally plotted. If they are not, slight differences in round-off error may cause the actual screen coordinates addressed to be different.

4.15 THE RDRAW COMMAND

PURPOSE: To draw a solid line from the current graphics cursor location using relative coordinates.

SYNTAX: RDRAW ⟨x-distance⟩ , ⟨y-distance⟩

ARGUMENTS:

⟨x-distance⟩ = an expression specifying the relative X distance.
⟨y-distance⟩ = an expression specifying the relative Y distance.

DISCUSSION:

The RDRAW command draws a solid line from the current graphics cursor location to a point specified by relative coordinates. The coordinates for this point are formed by adding the X and Y distances to the X and Y coordinates of the current graphics cursor, respectively. After drawing, this specified point then becomes the new graphics cursor location.

The specified X and Y distances may be any legal floating point values so long as the resulting graphics cursor coordinates are also legal floating point values. If either the current graphics cursor location or the point specified in the RDRAW command falls outside the current window, automatic clipping of the line will occur. The clipping will be such that only the portion of the line passing through the window will be drawn. If no portion of the line passes though the window, the display will be unchanged.

EXAMPLES:

Assuming the current graphics cursor coordinates are 5.25,10.0, then

```
RDRAW 0.5,3
```

will draw a solid line from the current graphics cursor position to the coordinates 5.75,13.0.

Assuming the current graphics cursor coordinates are 10000,1500, then

```
RDRAW DX(I),DY(I)
```

will draw a solid line from the current graphics cursor position to the X coordinate position of 10000+DX(I), and Y coordinate position of 1500+DY(I).

NOTES:

1. The action of clipping a line will have no effect on the graphics cursor coordinates. However, it will have an effect on the current screen coordinates used by the IGL Library. For more information on how clipping affects the current screen coordinates, see Appendix C.

2. Executing the RDRAW command will have no effect on the drawing mode as set by the PENMODE command in the IGL Library.

4.16 THE RMOVE COMMAND

PURPOSE: To move the graphics cursor to a new location using relative coordinates.

SYNTAX: RMOVE ⟨x-distance⟩ , ⟨y-distance⟩

ARGUMENTS:

⟨x-distance⟩ = an expression specifying the relative X distance.
⟨y-distance⟩ = an expression specifying the relative Y distance.

DISCUSSION:

The RMOVE command moves the current graphics cursor location to a point specified by relative coordinates. The coordinates for this point are formed by adding the X and Y distances to the X and Y coordinates of the current graphics cursor location, respectively.

The specified X and Y distances may be any legal floating point values so long as the resulting graphics cursor coordinates are also legal floating point values. The resulting graphics cursor position need not be within the current window.

EXAMPLES:

Assuming the current graphics cursor coordinates are 50,50, then

```
RMOVE 10.5,-10.3
```

will set the graphics cursor position to the coordinates 60.5,39.7.

Assuming the current graphics cursor coordinates are 75.9,10.4, then

```
RMOVE 0,DY
```

will set the graphics cursor X coordinate to the value of 75.9, and the Y coordinate to the value of 10.4+DY.

NOTES:

1. The RMOVE command has no effect on the current drawing mode as specified by the PENMODE command of the IGL Library.

4.17 THE RPEN COMMAND

PURPOSE: To draw a line from the current graphics location to a point using relative coordinates and drawing with the current drawing mode.

SYNTAX: RPEN ⟨x-distance⟩ , ⟨y-distance⟩

ARGUMENTS:

⟨x-distance⟩ = an expression specifying the relative X distance.
⟨y-distance⟩ = an expression specifying the relative Y distance.

DISCUSSION:

The RPEN command draws a line from the current graphics cursor location to a point specified by relative coordinates using the current drawing mode. The coordinates for this point are formed by adding the X and Y distances to the X and Y coordinates of the current graphics cursor, respectively. After drawing, this specified point then becomes the new graphics cursor location.

The specified X and Y distances may be any legal floating point values so long as the resulting graphics cursor coordinates are also legal floating point values. If either the current graphics cursor location or the point specified in the RPEN command falls outside the current window, automatic clipping of the line will occur. The clipping will be such that only the portion of the line passing through the window will be drawn. If no portion of the line passes though the window, the display will be unchanged.

The current drawing mode is the mode set by the PENMODE command in the IGL Library. For convenience, a list of these modes is given in Note 2 below.

EXAMPLES:

Assuming the current graphics cursor coordinates are 35.3,47.8, then

```
RPEN 100.5,-21.0
```

will draw a line using the current drawing mode, from point at 35.3,47.8 to the point at 135.8,26.8.

NOTES:

1. The action of clipping a line will have no effect on the graphics cursor coordinates. However, it will have an effect on the current screen coordinates used by the IGL Library. For more information on how clipping affects the current screen coordinates, see Appendix C.

2. Here is a list of the various drawing modes set by the PENMODE command in the IGL Library:

```
PENMODE 0  - move mode, no drawing takes place
PENMODE 1  - draw solid line mode
PENMODE 2  - erase line mode
PENMODE 3  - draw flip line mode
PENMODE 4  - move dashed mode, not a useful mode
PENMODE 5  - draw dashed line mode
PENMODE 6  - erase dashed line mode
PENMODE 7  - flip dashed line mode
```

4.18 THE RPOINT COMMAND

PURPOSE: To plot a point using relative coordinates.

SYNTAX: RPOINT ⟨x-distance⟩ , ⟨y-distance⟩

ARGUMENTS:

⟨x-distance⟩ = an expression specifying the relative X distance.
⟨y-distance⟩ = an expression specifying the relative Y distance.

DISCUSSION:

The RPOINT command plots a point specified by relative coordinates. The coordinates for this point are formed by adding the X and Y distances to the X and Y coordinates of the current graphics cursor location, respectively. This specified point then becomes the new graphics cursor location.

The specified X and Y distances may be any legal floating point values so long as the resulting graphics cursor coordinates are also legal floating point values. If the resulting point is within the current window, then the point is plotted by turning "on" the appropriate dot in the display. If the point is outside the current window, then the point is not plotted and the display remains unchanged.

EXAMPLES:

Assuming the current graphics cursor coordinates are 5.25,10.0, then

```
RPOINT 0.5,3
```

will plot the point at coordinates 5.75,13.0.

Assuming the current graphics cursor coordinates are 10000,1500, then

```
RDRAW DX(I),DY(I)
```

will plot the point with X coordinate of 10000+DX(I), and Y coordinate of 1500+DY(I).

NOTES:

1. If the specified point is plotted, then the current screen coordinates used in the IGL Library will reflect the location of this point on the screen. If the point outside the current window, and thus is not plotted, then the current screen coordinates are left unchanged.

2. Executing the RPOINT command will have no effect on the drawing mode as set by the PENMODE command in the IGL Library.

4.19 THE VIEWPORT COMMAND

PURPOSE: To define the boundaries of the viewport.

SYNTAX: VIEWPORT ⟨xmin⟩ , ⟨xmax⟩ , ⟨ymin⟩ , ⟨ymax⟩

ARGUMENTS:

⟨xmin⟩ = an expression for the X position of the left edge of the desired viewport.
⟨xmax⟩ = an expression for the Y position of the right edge of the desired viewport.
⟨ymin⟩ = an expression for the Y position of the bottom edge of the desired viewport.
⟨ymax⟩ = an expression for the Y position of the top edge of the desired viewport.

DISCUSSION:

The VIEWPORT command is used to set the viewport. This viewport is the physical rectangle of the screen which will be occupied by the user's window.

Since the arguments used in this command refer to actual screen coordinates, the range for xmin and xmax is 0 to 479. For ymin and ymax, the range is 0 to 255. If an argument is outside its range, it will be forced to the nearer limit without error indication. The default viewport is shown in the first example.

EXAMPLES:

```
VIEWPORT 0,479,16,255
```

will set the viewport to the full screen, minus the legends. This is the default viewport.

```
VIEWPORT X,X+100,Y,Y+50
```

will set a viewport with lower left corner at coordinates X,Y with width of 100 pixels and height of 50 pixels.

NOTES:

1. There are two exceptions allowed in the syntax for the VIEWPORT command given above. The xmin and xmax arguments may be swapped, and/or the ymin and ymax values may be swapped. Swapping xmin and xmax will flip the window from left to right. This means the window's minimum X coordinate will appear at the right edge of the viewport, and the maximum X at the left edge. Swapping ymin and ymax will flip the window upside down. This means the window's minimum Y coordinate will appear at the top of the viewport, and the maximum Y at the bottom. The effect on an image subsequently drawn into the viewport is to mirror or invert it.

For ymin ≥ legends (=15 pix) + n text lines (n*10 pix)
ymin = 15 + n*10 + 1

4.20 THE WINDOW COMMAND

PURPOSE: To set the bounds for the window.

SYNTAX: WINDOW ⟨xmin⟩ , ⟨xmax⟩ , ⟨ymin⟩ , ⟨ymax⟩

ARGUMENTS:

⟨xmin⟩ = an expression for the minimum X coordinate for the window.
⟨xmax⟩ = an expression for the maximum X coordinate for the window.
⟨ymin⟩ = an expression for the minimum Y coordinate for the window.
⟨ymax⟩ = an expression for the maximum Y coordinate for the window.

DISCUSSION:

The WINDOW command is used to set the bounds for the current window. The window defines the range of coordinate values which occupy the current viewport. When any graphics image is drawn, only the portion of that image which lies within the window will appear on the screen.

The values of the default window are given in the first example.

EXAMPLES:

```
WINDOW 0,100,0,50
```

will set the window bounds to 0 for xmin, 100 for xmax, 0 for ymin, and 50 for ymax. This is the default window.

```
WINDOW X0,X0+DX,Y0,Y0+DY
```

will set the window bounds to X0 for xmin, X0+DX for xmax, Y0 for ymin, and Y0+DY for ymax.

NOTES:

1. The window command requires that xmin and ymin arguments be less than the xmax and ymax arguments, respectively. If this requirement is not met, an ILLEGAL WINDOW ERROR is given.

4.21 THE XAXIS COMMAND

PURPOSE: To draw a horizontal axis with optional tic marks.

SYNTAX: XAXIS ⟨y-intercept⟩ [, ⟨tic-spacing⟩ [, ⟨xmin⟩ [, ⟨xmax⟩ [, ⟨tic-size⟩]]]]

ARGUMENTS:

⟨y-intercept⟩ = an expression for the Y intercept for the X axis.
⟨tic-spacing⟩ = an expression for the spacing between each tic mark.
⟨xmin⟩ = an expression which gives the starting point of the X axis.
⟨xmax⟩ = an expression which gives the ending point of the X axis.
⟨tic-size⟩ = an expression which gives the tic mark size, in number of dots.

DISCUSSION:

The XAXIS command provides a simple means of obtaining an X axis with optional tic marks. The XAXIS command accepts up to 5 arguments. Only the first is required, the rest are optional. All of the arguments except the tic-size are specified in virtual coordinate values.

If only the first argument (Y intercept) is specified, a horizontal line is drawn at the Y intercept the full width of the viewport. Because no other arguments are present, no tic marks will be drawn. If at least the first two arguments (Y intercept and tic-spacing) are specified, the remaining arguments will have the following default values if not specified.

⟨xmin⟩ = value of current window X minimum
⟨xmax⟩ = value of current window X maximum
⟨tic-size⟩ = 2

An XAXIS command with multiple arguments specified will draw a horizontal line at the Y intercept from xmin to xmax. Tic marks will be drawn at uniform intervals as specified by the tic-spacing. If the tic-spacing is positive, the tic marks will be drawn starting at xmin, and proceed toward xmax. If the tic-spacing is negative, they will be drawn starting at xmax, and proceed toward xmin. The actual tic marks will be drawn from tic-size dots above the axis to tic-size dots below the axis.

If any of the horizontal axis to be drawn is outside the current window, only the portion that falls within the window will be drawn.

EXAMPLES:

```
XAXIS 0,5
```

will draw a horizontal axis at Y intercept = 0, with a tic spacing of 5.

```
XAXIS YI,TS,X0,X0+100
```

will draw a horizontal axis from X0 to X0+100 at Y intercept = YI, with a tic spacing of TS.

NOTES:

1. If xmin is greater than xmax, an ILLEGAL AXIS SPECIFICATION ERROR is given.

2. If the specified tic-spacing translates to an interval of less than one pixel, a TIC SPACING TOO SMALL ERROR is given.

4.22 THE YAXIS COMMAND

PURPOSE: To draw a vertical axis with optional tic marks.

SYNTAX: XAXIS <x-intercept> [, <tic-spacing> [, <ymin> [, <ymax> [, <tic-size>]]]]

ARGUMENTS:

<x-intercept> = an expression for the X intercept for the Y axis.
<tic-spacing> = an expression for the spacing between each tic mark.
<ymin> = an expression which gives the starting point of the Y axis.
<ymax> = an expression which gives the end point of the Y axis.
<tic-size> = an expression which gives the tic mark size, in number of dots.

DISCUSSION:

The YAXIS command provides a simple means of obtaining a vertical axis with optional tic marks. The YAXIS command accepts up to 5 arguments. Only the first is required, the rest are optional. All arguments except the tic-size are specified in virtual coordinate values.

If only the first argument (X intercept) is specified, a vertical line is drawn at the X intercept the full height of the viewport. Because no other arguments are present, no tic marks will be drawn. If at least the first two arguments (X intercept and tic-spacing) are specified, the remaining arguments will have the following default values if not specified.

<ymin> = value of current window Y minimum
<ymax> = value of current window Y maximum
<tic-size> = 2

A YAXIS command with multiple arguments specified will draw a vertical line at the X intercept from ymin to ymax. Tic marks will be drawn at uniform intervals as specified by the tic-spacing. If the tic-spacing is positive, the tic marks will be drawn starting at ymin, and proceed toward ymax. If the tic-spacing is negative, they will be drawn starting at ymax, and proceed toward ymin. The actual tic marks will be drawn from tic-size dots to the right of the axis to tic-size dots to the left of the axis.

If any of the vertical axis to be drawn is outside the current window, only the portion that falls within the window will be drawn.

EXAMPLES:

```
YAXIS 100,10
```

will draw a vertical axis at X intercept = 100, with a tic spacing of 10.

```
YAXIS XI,DX,Y0,Y0+DX*20,4
```

will draw a vertical axis from Y0 to Y0+DX*20 at X intercept = XI, with a tic spacing of DX. The tic marks will be extend 4 pixels on either side of the axis.

NOTES:

1. If ymin is greater than ymax, an ILLEGAL AXIS SPECIFICATION ERROR is given.

2. If the specified tic-spacing translates to an interval of less than one pixel, a TIC SPACING TOO SMALL ERROR is given.

APPENDIX A.

COMMAND SUMMARY

SYNTAX	FUNCTION
BOXVP	- Draw box around current viewport.
CLEARVP	- Clear current viewport.
DASH x,y	- Draw dashed line.
DRAW x,y	- Draw solid line.
ERASE x,y	- Erase line.
FILL xmin,xmax,ymin,ymax	- Fill rectangular area, using current penmode.
FLIP x,y	- Flip line.
FLIPVP	- Flip state of all points in viewport.
GRIN s$,x,y	- Input graphics coordinates using GRIN cursor.
LTPEN f,x,y	- Input graphics coordinates using light pen.
MOVE x,y	- Move to coordinates.
PEN x,y	- Draw using current penmode.
POINT x,y	- Plot point.
QDOT(x,y)	- Get status of point.
RDRAW x,y	- Draw solid line using relative coordinates.
RMOVE x,y	- Move to point relative to current point.
RPEN x,y	- Draw line using current penmode and relative coordinates.
RPOINT x,y	- Plot point using relative coordinates.
VIEWPORT xmin,xmax,ymin,ymax	- Set viewport.
WINDOW xmin,xmax,ymin,ymax	- Set window.
XAXIS y-intercept,tic spacing,xmin,xmax,tic size	- Draw X axis.
YAXIS x-intercept,tic spacing,ymin,ymax,tic size	- Draw Y axis.

APPENDIX B.

THE MAXIMUM VALUES HANDLED BY THE VGL LIBRARY

The VGL Library makes use of the floating point routines found in MTU BASIC to perform all necessary calculations. These routines can handle numbers which are very large, but there is a limit. The largest value a floating point number can have is 1.70141183E+38. If at any point this value is exceeded during the calculations an OVERFLOW ERROR will occur.

There are two places where you might run into the OVERFLOW ERROR should you wish to use very large numbers. The first place is in the WINDOW command. When transforming from virtual to screen coordinates, an X scale factor and Y scale factor are used. The scale factor is computed according to the equation:

SCALE FACTOR = (VIEWPORT MAX - VIEWPORT MIN) / (WINDOW MAX - WINDOW MIN)

where the values are for either X or Y. This equations imposes a limit on the size of the window since WINDOW MAX - WINDOW MIN must not exceed 1.70141183E+38. If this is exceeded for either the X or Y scale factor, an OVERFLOW ERROR will occur when the WINDOW command is executed. Here is an example of a WINDOW command which will cause such an error.

```
WINDOW -3.6E+35,1.7E+38,0,1000
```

An OVERFLOW ERROR may also occur when you attempt to draw a diagonal line involving a point outside the current window. In this case the equation which clips the line must compute the difference between the X and Y coordinates of the two points. If one of the points is outside the window, the X or Y difference may exceed the 1.70141183E+38 limit. If this is the case, an OVERFLOW ERROR will occur when you attempt to draw the line. This won't occur on strictly horizontal or vertical lines, because clipping can be done without an equation.

APPENDIX C.

INTERMIXING VGL AND IGL LIBRARY COMMANDS

Many commands in both the VGL and IGL libraries use as a starting point the point last drawn or moved to by a previous command. In the VGL documentation, this point is called the graphics cursor location. In the IGL documentation this point is called the current screen coordinates. The IGL commands may only affect the current screen coordinates. However, the VGL commands may affect both of them. If you are intermixing IGL and VGL drawing commands this fact may be important.

There are basically two facts which should be remembered when mixing commands in the VGL and IGL Libraries which affect the cursors. The first is that the VGL commands will always set the current screen coordinates to what it needs to do its drawing. Therefore, using IGL drawing commands in between VGL drawing commands will not affect what the VGL commands do. Entering and running the following short program will illustrate this:

```
10 FRELIB:LIB "VGL","IGL"
20 CLEARVP
30 MOVE 0,0:DRAW 50,25
40 SRMOVE 0,6:SRDRAW 6,-6:SRDRAW -6,-6:SRDRAW -6,6:SRDRAW 6,6
50 DRAW 100,50
```

You will note that the drawing of the diamond with IGL commands doesn't affect the line drawn by VGL commands.

The second fact deals with the relationship between the two cursors when using just VGL drawing commands. The point referenced by the two cursors will always be the same as long as the commands use coordinates which are within the current window. However, when you draw to a point outside the current window, they cease to be the same point. VGL's graphics cursor will naturally contain the coordinates specified in the command. IGL's screen coordinates will contain the coordinates of the point where the line intersects the edge of the window. Entering and running the following short program will illustrate this:

```
10 FRELIB:LIB "VGL","IGL"
20 CLEARVP
30 WINDOW 0,479,16,255:REM MAKE WINDOW THE SAME AS THE VIEWPORT
40 BOXVP:MOVE 0,100
50 DRAW 240,300
60 SDRAW 400,100
70 DRAW 400,100
```

The line coming from the edge of the window to the point 400,100 was drawn by the SDRAW command. The line coming from the non-visible point 240,300 to the point 400,100 was drawn by the DRAW command in line 70.

APPENDIX D.

LOCATION OF THE WINDOW AND VIEWPORT VALUES IN THE VGL PROGRAM

This appendix describes how to obtain the current boundary values of the window and viewport in the VGL program. This would be necessary if you were developing a library of your own commands which involve the window or viewport.

Since the location of where the VGL program is running can vary, the location of the viewport and window values will also vary. These values must therefore be found through pointers which don't move.

The window boundaries are found using a pointer at 781 hex. This pointer points to a location where the four window boundary values are stored. Each is a five byte floating point number. The order in which the values are stored is as follows:

```
($781)     - X window minimum
($781)+5   - X window maximum
($781)+10  - Y window minimum
($781)+15  - Y window maximum
```

The viewport boundaries are found using a pointer at 783 hex. This pointer points to the location where the four viewport boundary values are stored. Each value is a two byte integer stored low byte followed by high byte. The order in which the values are stored is as follows:

```
($783)     - X viewport minimum
($783)+2   - Y viewport minimum
($783)+4   - X viewport maximum
($783)+6   - Y viewport maximum
```

The pointers at 781 and 783 hex are set during initialization of the VGL Library. This means they are not valid unless VGL is LIBed in.