7.11. POSTSCRIPT

7.11.1 INTRODUCTION TO POSTSCRIPT

The PostScript page description language is useful for representing the printed page in a device independent manner. It is the language that the majority of the laser printers in the Department of Computer Science understand. It is also used by some of the new window managers as their graphical language.

Some features of the PostScript language are:

- PostScript is particularly suited to describe graphical information

- about 1/3 of PostScript is devoted to graphics -- the remainder is a general purpose programming language which closely resembles FORTH and other stack-oriented languages

- it is often called a "page generation language"

- it is interpreted

7.11.2 VIEWING POSTSCRIPT FILES

PostScript files can be viewed by either using some kind of viewing program or by sending them to a PostScript printer. To view a PostScript file on the workstation monitor, use the *GhostScript* program. To print your file, use the *lpr* command.

7.11.2.1 USING GHOSTSCRIPT

GhostScript is a PostScript interpreter that runs on many different computers under a variety of different environments. It can be used as a stand alone program or as a front/back end for other programs such as *ghostview*, for example. To use GhostScript, log into an terminal with X display capabilities, such as one of the DEC Stations and enter the command `'gs'`. GhostScript uses a command line window for command entry and displays results in another X-Window. You can enter a PostScript program interactively by directly typing it into GhostScript, line by line, or you can load and execute a program using the following command: `'(filename.ps) run'`. To get out of GhostScript, type `'quit'`.

7.11.2.2 USING LPR

In most applications, a PostScript program (or PostScript file) is created and saved automatically or sent to some output device, such as a printer. To print out a PostScript file, the printer used must support the PostScript language, as is the case for all laser printers in the CS department. To print out your PostScript source code you should use the following command:

```
lpr -Pc135ps filename.ps
```

To print out the output from your PostScript source code you should make sure that the first line of your program is "`%!`" and then issue the same lpr command as before. If the first line of your program is not "`%!`", then the source code will be printed and not the image generated by the source code.

7.11.3 POSTSCRIPT INTERNALS

7.11.3.1 POSTSCRIPT'S STACK

PostScript reserves a piece of memory called a *stack* (FILO, LIFO). All PostScript functions (known as *macros*) which require parameters take them from the stack. When passing values to macros, they are placed on the stack in the proper order required. PostScript uses *postfix notation*. The operands that it requires are placed on the stack in the order required and the results of the operation are then placed on the stack after the operands have been removed.

7.11.3.2 POSTSCRIPT GRAPHICAL OBJECTS

PostScript supports 3 types of graphical objects:

- **Text**:

- any typeface in any position, orientation, or scale

- **Geometric Figures**:

- describe locations of lines of any size, orientation, & width as well as spaces of any size, shape, or colour

- **Sampled Images**:

- in any scale or orientation

7.11.3.3 POSTSCRIPT IMAGING MODELS

An *imaging model* is a set of rules which are very similar to those used by graphical artists. A page is built up piece by piece and then displayed on an output device. In PostScript, all "paint" is opaque.

- **current page**:

- this is the "ideal page" that the image is drawn on and is independent of any output device used

- when a program begins, the current page is blank

- **current path**:

- a set of connected and disconnected points, lines, and curves that, together, describe shapes and positions

- a path is not actually a mark on a page (yet) but must be stroked, clipped, or filled

- **clipping path**:

- the boundary of the area that may be drawn upon -- originally the standard paper size but can be any size or shape

7.11.3.4 COORDINATE SYSTEM

PostScript uses a standard X,Y coordinate system with location (0,0) in the bottom left hand corner. This is known as *user space*. Different output devices have different, idiosyncratic coordinate systems, which is called *device space*. Coordinates in a PostScript program are automatically transformed from user space to device space before actual printing of the current page. User space may be altered by *translating*, *rotating*, or *scaling* PostScript's default units are 1/72 inch, also known as a *point*. Therefore, 72 points is equal to one inch.

7.11.4 EXAMPLES

7.11.4.1 SIMPLE EXAMPLES

The following are short examples that demonstrate the use of the PostScript stack.

7.11.4.2 MORE COMPLEX EXAMPLES

The following are some PostScript example programs selected to show the features of PostScript in action. As with any other language, the only way to learn it is to try it. In the examples below, as new PostScript keywords are encountered, they will be highlighted.

7.11.5 POSTSCRIPT IMAGES

PostScript contains numerous operators that allow users to create graphics. PostScript also contains features that allow for the easy manipulation of *bitmaps*. The *image* operator is used to perform these functions

7.11.5.1 THE IMAGE OPERATOR

The image operator interprets the bits passed to it as a description of the gray values of a stream of pixels of from one to eight bits each. The results are printed in a *one-unit square* in which the lower left corner is at the origin. To view the image you must *scale* it to a viewable size since the default is 1/72 inch -- far to small to be really usable.

The image operator requires 5 arguments:

- **scan length**

- the number of samples per scan line

- **scan lines**

- the number of scan lines in the image

- **bits per sample**

- the number of bits making up the sample (1, 2, 4, or 8)

- an image with 1 bit sampling will print in black or white while an image with 8 bit sampling will print using the values 0 (black) through 255 (white)

- **transform matrix**

- the image operators impose a coordinate system on the source image:

[IMAGE]

where **w** is the width and **h** is the height

- other conventions (ie. user space) can be mapped into the PostScript convention (image space) by coordinate transformations

- a six element array that determines the mapping of samples into the one unit square imaging region

- for an image with scan length of n and scan lines of m, the following matrix is used: [n 0 0 m 0 0]

- for an image whose data begins in the upper left corner as opposed to the lower left (PostScript's default), the following matrix may be used: [n 0 0 -m 0 m]

- for a more general definition of the transform matrix, please refer to Section 4.4 of the PostScript Language Reference Manual

- **procedure**

- this is the procedure that produces the data strings needed by *image*

- if the string does not describe the complete image, the image operator will call this procedure again and again until the number of samples implied by the first 3 arguments have been processed

- any unused data left in the string at the end of the image is ignored as are any bits left in the current character of data at the end of the scan line

7.11.5.2 POSITION, ORIENTATION, AND SIZING

An image that has been mapped into the unit square may then be placed on the output page in the desired position, orientation, and size by invoking operations such as translate, rotate, and scale

ex: To place an image into a rectangle where the lower left corner is at (100,200), is rotated 45 degrees counterclockwise, and is 150 units wide and 80 high, one would execute the following prior to invoking the image operator:

```
100 200 translate 45 rotate 150 80 scale
```

7.11.5.3 A BINARY IMAGE:

Consider the following 16 bit string: `1100100100110110`

We can use the preceeding string in a PostScript program, such as in the following segment:

```
400 400 translate % position of lower left corner

72 72 scale % make the image 1 inch by 1 inch square

8 8 1 [8 0 0 8 0 0] {<c936>} image

showpage
```

- The third line prints an 8 pixel by 8 line image, each pixel being 1 bit deep.

- The transform matrix will fill the unit square with the image.

- The procedure argument encloses a hexadecimal string, denoted by the angle brackets:

- each pair of characters represents an 8 bit code, therefore `c936` is interpreted as the 16 bit string listed above

- it is important to note that, by default, PostScript defines images with the origin in the lower left corner and where the higher numbers are lighter colours and the lower numbers are darker colours -- in this case, 1 is white and 0 is black

The above program produces a simple bitmap where, since each line of the image is eight one-bit samples wide, each call of the procedure will supply data for two lines producing an image something like:

```
00110110

11001001

00110110

11001001

00110110

11001001

00110110

11001001
```

By changing the bits per sample a different pattern is created -- recall that 1 bit per sample results in a black and white image, 2 bits produces shades of gray from 0 to 3, 4 bits produces shades from 0 to 15, and 8 produces from 0 to 255

7.11.5.4 CREATING AN IMAGE

PostScript images can be created in a number of different ways:

- **generated by another program**

- many programs generate output files in PostScript format -- PMDraw, Corel Draw, and Super Paint are some examples

- **scanned in**

- a user may decide to use a scanner to digitize an existing picture and store it as a PostScript bitmap

- **created entirely by hand**

- a user may also decide to generate a bitmap entirely by hand, for example, by using a pencil and some graph paper

ex: Suppose a raster image of 256 by 256, 8 bits per pixel is available -- one way to display it is:

7.11.6 MISCELLANEOUS POSTSCRIPT OPERATORS

In addition to the image manipulation described above, there are some others that may be of use:

- **gsave**

- save the current graphics state on a *graphics state stack*, which can hold up to 32 states including the current one

- **grestore**

- restores the most recently saved graphics state -- all the characteristics of the state including current path, gray level, line width, and user coordinate system are returned to what they were when gsave was executed

- **X Y scale**

- set the appropriate scaling factor for all subsequent operations

- **currentpoint**

- put the coordinates of the current location on the stack

- **n { ... } repeat**

- the repeat loop causes a procedure to be executed n times

- **l m n { ... } for**

- the for loop iterates a procedure from l to n by an increment (or decrement) value of m

- **(...) stringwidth**

- stringwidth returns the length of the string (...) and (usually) the value 0.0

ex: the following code will determine the width of a string and center it on some background that is 200 units wide

```
(PostScript) stringwidth pop

200 exch sub 2 div

0 rmoveto

(PostScript) show
```

7.11.7 OTHER FILES OR BOOKS OF INTEREST

A few sample images are stored on the U of R **ftp** site in the directory **pub/405/PS**. These images are contained in actual PostScript programs -- to incorporate them into your own programs, you will need to edit the images out. Some of the sample PostScript programs use the commands described above as well as other commands not described in this manual, so it would be beneficial to give them a view.

Other sample files are available in **/net/share/lib/ghostscript/examples**. To determine which fonts are available for use in GhostScript, you may want to look in the file **/net/share/lib/ghostscript/Fontmap** for a list of the correct font names. The directory **/net/share/lib/ghostscript/doc** also contains a number of very useful documentation files for GhostScript as well as for PostScript in general.

The two books "PostScript Language Tutorial and Cookbook" and "PostScript Language Reference Manual" are very good PostScript references published by Adobe Systems Incorporated.