PostScript Tutorial

Michael D. Lore
University of Houston
Department of Electrical Engineering

I. Introduction

A. PostScript: A Page Description Langauge

1. Can write a variety of text in many different typestyles. Can draw geometric figures, and draw digitized images. Provides support for color too.

2. The imaging model allows one to build up where the ink goes on a page, and then display the page. PostScript has a current page, a current path, and a current clipping path. Operators exists for defining the current path and current clipping path. Other operators allow you to stroke or fill the path.

3. PostScript uses a standard mathematical coordinate system, and allows coordinate system operations like rotation and translation.

B. PostScript: A Programming Language

1. The portions of PostScript not devoted to graphics make up a general programming language based on postfix notation and a stack.

2. The PostScript stack is where most data and program code reside. Postfix notation is used because it most closely parallels the operation of a stack.

3. PostScript supports several data types, like integers, reals, and dictionaries.

II. Using the PostScript Langauge

A. The Stack

1. A stack is a type of last-in, first-out list. A good example is an auto change dispenser, in which coins may be inserted and removed from the top. The last coin placed in is always the first coin removed.
Putting an item on top of the stack is called "pushing it onto the stack."
Removing an item from the top of the stack is called "popping it off of the stack."

2. In PostScript, an in-line item is pushed onto the stack by writing it:

```
23 44 55.5
```

Places 23, then 44, and then 55.5 onto the top of the stack. When operators are placed on the stack, they are immediately thereafter executed. You might think of an execution for every item ever placed on the stack: the operation defined for numbers is to place them on the stack.

23 44 add ---> 67

B. Postfix Notation

1. Normal arithmetic operators follow a notation convention in which the operator is placed in between the two items being operated on:

3 + 4

With Postfix notation, the operator is placed after the two arguments:

3 4 +

2. Normal arithmetic suffers from an ill-defined way to determine precedence between operations. 4 + 5 * 7 must be evaluated carefully because * has priority over +.

With postfix notation, however, there is no such problem. 4 5 7 * + can be easily and almost carelessly evaluated because each operator always acts upon the previous two arguments. This method is very powerful for a stack-based system, because operators and arguments can be immediately evaluated as they are popped off the stack.

C. PostScript Examples

1. Arithmetic

5 3 add	-->	8	like 5 + 3
99 1 sub	-->	98	like 99 - 1
3 3 3 add mul	-->	18	like (3 + 3) * 3

2. Stack operators

3 44 55 clear	-->	
4 dup	-->	4 4
3 55 pop	-->	3
33 44 exch	-->	44 33

III. Drawing

A. Introduction

1. Drawing consists of two steps: defining a path to follow and then stroking or filling that path. My examples will be of drawing a box. PostScript features will be desribed along the way.

2. The YAP PostScript previewer is used for these examples, so that we can

see what is going on along the way.

B. A Box

1. Draw a box

```
newpath
        270 360 moveto
        0 144 rlineto               % relative line to.
        144 0 rlineto
        0 -144 rlineto
        -144 0 rlineto
        12 setlinewidth             % set width of drawn line
stroke                              % finally stroke (as if with a pen) the outline
showpage
```

2. Draw a better box

```
newpath
        270 360 moveto
        0 144 rlineto
        144 0 rlineto
        0 -144 rlineto
        closepath                   % closepath closes a path nicely
        12 setlinewidth
stroke
showpage
```

3. Fill the box

```
newpath
        270 360 moveto
        0 144 rlineto
        144 0 rlineto
        0 -144 rlineto
        closepath
fill                                % fill the outline instead
showpage
```

4. Use a different color

```
newpath
        270 360 moveto
        0 144 rlineto
        144 0 rlineto
        0 -144 rlineto
        closepath
0.5 setgray                         % use a different gray level
fill
showpage
```

5. Draw 3 overlapping boxes

```
newpath                         % black box
      252 324 moveto
      0 144 rlineto
      144 0 rlineto
      0 -144 rlineto
      closepath
0 setgray
fill

newpath                         % dark gray box
      270 360 moveto
      0 144 rlineto
      144 0 rlineto
      0 -144 rlineto
      closepath
0.4 setgray
fill

newpath                         % light gray box
      288 396 moveto
      0 144 rlineto
      144 0 rlineto
      0 -144 rlineto
      closepath
0.8 setgray
fill

showpage
```

6. Why write the code 3 times as above? We can define procedures.

```
/X 270 def            % variable stores x location
/Y 360 def            % variable stores y location

% --- procedure to draw a box.  Stack: -  -->  -
/box {
       144 0 rlineto
       0 144 rlineto
       -144 0 rlineto
       closepath
} def

% --- main program
newpath                             % black box
      X 18 sub Y 36 sub moveto box 0 setgray
fill
newpath                             % dark gray box
```

```
	X Y moveto box 0.4 setgray
fill
newpath					% light gray box
	X 18 add Y 36 add moveto box 0.8 setgray
fill

showpage
```

7. We can also pass parameters on the stack. The inch procedure converts from inches to the PostScript coordinates. We also redefine the box procedure to allow us to tell where to draw it, what color, and how big.

```
% --- procedure to convert from inches to 1/72 inch units. Stack: inches --> psunits
/inch {
	72 mul
} def

% --- procedure to draw a box. Stack: size color x y -->  -
/box {
	newpath
		moveto
		setgray
		/size exch def
		size 0 rlineto
		0 size rlineto
		size neg 0 rlineto
		closepath
} def

% --- main program
2 inch 0 1 inch 1 inch box fill			% black box
2.5 inch 0.4 1.5 inch 1.5 inch box fill% dark gray box
3 inch 0.8 2.0 inch 2.0 inch box fill		% light gray box
3 inch 0 2.0 inch 2.0 inch box
	8 setlinewidth stroke			% border light gray box

showpage
```

8. Generally, passing parameters on the stack can be done as follows:

If the call to the procedure is: parm1 parm2 parm3 procedureName

Then:

```
/procedureName {
	/parm3 exch def
	/parm2 exch def
	/parm1 exch def
```

```
        .... use variables parm1, parm2, and parm3
} def
```

is the procedure.

9. Writing text

To write text, we must perform the following operations:

- Find the font of the given name from the font dictionary.
- Scale the font to the appropriate size.
- Set the font by popping it off the stack and establishing it as the current font.

```
/inch {72 mul } def
/font1 /Times-Roman findfont 100 scalefont def
          % save the found, scaled font
/font2 /Times-BoldItalic findfont 80 scalefont def
          % save the found, scaled font
2 inch 2 inch moveto                % write 'HELLO'
0 setgray
font1 setfont (HELLO) show
3 inch 1.7 inch moveto              % write 'there'
0.3333 setgray
font2 setfont (there) show
showpage
```

10. Final Example: a NeXT Button

```
% draw button - xloc,yloc should be lower left corner of button.
%  stack: xloc yloc xsize ysize -> -
/white 1.0 def
/lightGray 0.66666 def
/darkGray 0.33333 def
/black 0.0 def
/drawButton {
        gsave
        /ysize exch def
        /xsize exch def
        /yloc exch def
        /xloc exch def
        1 setlinewidth
        2 setlinecap
                newpath
                        xloc yloc moveto
                        0 ysize rlineto
                        xsize 0 rlineto
                white setgray stroke
                newpath
                        xloc yloc moveto
```

```
            xsize 0 rlineto
            0 ysize rlineto
        black setgray stroke
        newpath
            1 xloc add 1 yloc add moveto
            xsize 2 sub 0 rlineto
            0 ysize 2 sub rlineto
        darkGray setgray stroke
        newpath
            1 xloc add 1 yloc add moveto
            xsize 2 sub 0 rlineto
            0 ysize 2 sub rlineto
            2 xsize sub 0 rlineto
        closepath lightGray setgray fill
    grestore
} def
/inch { 72 mul } def

% main program
/Times-Roman findfont 30 scalefont setfont

% draw background
newpath
    0 0 moveto
    0 4 inch rlineto
    4 inch 0 rlineto
    0 -4 inch rlineto
    closepath
    lightGray setgray
fill

% draw button
2 inch 2 inch 1.4 inch 0.5 inch drawButton
0 setgray
2.1 inch 2.1 inch moveto
(Button) show
showpage
```

IV. Conclusion

PostScript is versatile enough to provide all drawing needs, and fast enough on the NeXT for good application performance. It allows a true "what you see is what you get" system, and developers need not worry about supporting various different kinds of hardware.