Kenneth A. Reek, Professor
Department of Computer Science
102 Lomb Memorial Drive
Rochester Institute of Technology
Rochester, NY 14623
kar@cs.rit.edu
Hello World

%!
/ZapfChancery-MediumItalic findfont
100 scalefont setfont
80 400 moveto
(Hello world!) show
showpage

Hello world!
What Is PostScript?

A page description language
- Describes how dots are produced on raster devices

A programming language
- Variables, loops, conditionals, procedures

Not intended for use by people
- More likely produced by a graphics or word processing program
Characteristics

PostScript is
• interpreted, stack-based
• extensible

Data types provided:
• integers, reals, strings
• arrays, dictionaries
• names
Stacks

The PostScript interpreter uses four stacks:

- operand (``the stack``)
- execution
- dictionary
- graphics state
Dictionaries

A dictionary is a collection of key/value pairs

• Present a dictionary with a key, and the value is looked up and executed
• `Executing` something like a number means to push it on the stack

Dictionary lookups

• start with the topmost dictionary on the dictionary stack
• look in lower ones only if key not found

The bottom dictionary on the stack is `systemdict` which defines all the operators

• New operators can be defined
• Existing operators can be redefined
Imaging Model

Lay ink down on paper
- The ink can be any shade of gray or any color
- The ink is opaque (things drawn later obscure things drawn earlier)

User coordinate system:
- Abstract, initially measured in points (1/72 inch)
- Lower left corner is (0,0)
- Upper right corner depends on physical medium

Device coordinate system:
- Depends on the nature of the device

Best to work in user coordinate space
Imaging Model

Marks made on the paper by defining a path and then painting it

• Path can be a non-contiguous sequence of lines, arcs, Bézier curves

To paint a path:

• Stroke: follow the path with a brush
• Fill: paint the inside of the path
Execution Model

In batch mode, the interpreter reads and executes input until EOF is sensed.

- A "job" is a contiguous, all-inclusive sequence of bytes that describes the page(s) to be printed
- The job is executed as it is read in (not stored first)
- Cannot use anything that has not yet been defined
- Data is put on the stack as it is read—most efficient to use it there
The properties that affect graphics operations include:

- current point
- current path
- current font
- current clipping region
- etc.
Sample Drawing Program

%! 
100 100 moveto % set the current point
200 300 lineto % draw a line to this point

500 200 moveto % move to center
400 200 100 0 270 arc % draw arc around center

stroke % paint (then destroy) the path

100 400 moveto % start a new path
150 0 rlineto % draw relative to current point
0 150 rlineto 
-150 0 rlineto

closepath % add a line back to beginning of path

.5 setgray % change the ink color

fill % fill the shape

showpage % display the page
Creating and Using Procedures

Tools needed:

- `/`: indicates that a literal name follows
- `{ }`: enclose a procedure
- `bind`: replace operator names with their operations
- `def`: define the procedure

Example:

```
/inch { 72 mul } bind def
```
Using Procedures

%! 
/box { % stack: x y
moveto % use x & y from stack
200 0 r lineto % draw the box
0 200 r lineto
-200 0 r lineto
closepath fill % and fill it
} bind def % define the procedure

100 100 box % draw a box
.5 setgray % change the ink and
200 200 box % draw another one
.75 setgray
300 300 box % display the page
showpage
A ```font``` is a dictionary

- The ```keys``` are the characters in the font
- Each ```value``` is a procedure that describes how to draw the character

The default character size is one point

- Must scale the font to desired size before using

**Example:**

```
/Courier findfont 16 scalefont setfont
```

If you’re changing fonts often, do this:

```
/F { findfont exch scalefont setfont } bind def
16 /Courier F
```
Drawing Text

To draw text:

- move to where you want it to go
- put a string on the stack
- paint it with the `show` operator
Now is the time for all good men to come to the aid of their country.
Variations

Text is drawn starting at current point: left justification

- To get other justifications: just change the point

Tools needed:

- stringwidth: pushes width and height of string on the stack
Justified Text

%!
/inch { 72 mul } bind def
/right_justify { % stack: (string)
  dup stringwidth pop % (string) swid
  line_width % (string) swid line_width
  exch sub % (string) line_width–swid
  0 rmoveto % (string)
  show
} def

center { % stack: (string)
  dup stringwidth pop line_width exch sub
  2 div 0 rmoveto show
} def

/line_width 7.5 inch def

/Helvetica findfont 32 scalefont setfont
.5 inch 8 inch moveto
(Left justified) show
.5 inch 7.5 inch moveto
(Centered across the line) center
.5 inch 7 inch moveto
(Right justified) right_justify
showpage
Modifying the Coordinate System

Types of modifications:

- Translation
- Scaling
- Rotation
Translation

Moves the origin of the coordinate system

- Example:

  72 72 translate

  moves origin one inch to the right and up
%!
/NewCenturySchlbk-Bold findfont
30 scalefont setfont
/box { % stack: (string)
    0 0 moveto 144 0 rlineto % draw box
    0 144 rlineto −144 0 rlineto % and
    closepath fill % fill it
    dup stringwidth pop % center text
    2 div 72 exch sub % on top of
    150 moveto show % the box
} bind def

(Box 1) box % draw a box at the origin
200 0 translate % now move the origin
(Box 2) box % and draw another one
100 300 translate
(Box 3) box
−200 200 translate
(Box 4) box
showpage
Scaling changes the size of the units in the coordinate system

- Example:

  2 2 scale

  doubles the units

- There will be only 36 units per inch; objects will be drawn twice as large
Scaling

%! 
/NewCenturySchlbk-Bold findfont 
30 scalefont setfont 
/box { % stack: (string) 
  0 0 moveto 144 0 rlineto % draw box 
  0 144 rlineto −144 0 rlineto % and 
  closepath stroke % fill it 
  dup stringwidth pop % center text 
  2 div 72 exch sub % on top of 
  150 moveto show % the box 
} bind def 

(Box 1) box % draw a box at the origin
2 2 scale % change the scaling
(Box 2) box % and draw another one
2 .75 scale
(Box 3) box
.33 3 scale
(Box 4) box
showpage
Rotation

Rotation is performed around the origin

- Example:

  60 rotate

  rotates the coordinate system 60 degrees counterclockwise
Rotation

%! 
/Courier findfont 30 scalefont setfont 
/box { % stack: (string) 
  396 0 moveto 144 0 rlineto % draw box 
  0 144 rlineto -144 0 rlineto % and 
  closepath fill % fill it 
  dup stringwidth pop % center text 
  2 div 468 exch sub % on top of 
  150 moveto show % the box 
} bind def 

(Box 1) box % draw a box 
30 rotate % rotate the coord system 
(Box 2) box % draw another one 
30 rotate 
(Box 3) box 
30 rotate 
(Box 4) box 
showpage
Rotation

Where did box 4 go?

- It fell off of the edge of the paper
- Hard to debug if you can’t see the output
- The following program is useful for debugging:

```
2.833 in 3.667 in translate % move origin toward center of page
0.333 .333 scale % make everything 1/3 normal size
0 0 moveto 8.5 in 0 rlineto % trace a line around
0 11 in rlineto -8.5 in 0 rlineto % our small page
closepath stroke % and stroke it
```

Arranges for the output to appear in a 1/3rd size box in the center of the page
%!
/in { 72 mul } def
2.833 in 3.667 in translate
.333 .333 scale
0 0 moveto 8.5 in 0 rlineto 0 11 in rlineto
–8.5 in 0 rlineto closepath stroke

/Courier findfont 30 scalefont setfont
/box { % stack: (string)
  396 0 moveto 144 0 rlineto % draw box
  0 144 rlineto –144 0 rlineto % and
  closepath fill % fill it
  dup stringwidth pop % center text
  2 div 468 exch sub % on top of
  150 moveto show % the box
} bind def

(Box 1) box
30 rotate (Box 2) box
30 rotate (Box 3) box
30 rotate (Box 4) box
showpage
Saving and Restoring the Graphics State

``gsave`` saves the graphics state on the graphics state stack
``grestore`` pops a state and restores it

- Useful to isolate local changes to the state
Saving and Restoring the Graphics State

%! /star { % stack: x y radius
/radius exch def
moveto
0 radius rmoveto
gsave
162 rotate 0 radius 2 mul rlineto
144 rotate 0 radius 2 mul rlineto
144 rotate 0 radius 2 mul rlineto
closepath
gsave .75 setgray fill grestore
stroke
} bind def

306 500 150 star
153 250 75 star
459 250 75 star
showpage
**Conditionals**

Conditional:

```plaintext
boolean { proc } if
boolean { trueproc } { falseproc } ifelse
```

Boolean and logical operators:

```plaintext
eq, ne, gt, ge, lt, le
and, or, not, xor
```

Example:

```plaintext
y 12 gt { stroke } { fill } ifelse
```
The mind is not a vessel to be filled, but a fire to be ignited.
—Plutarch
Loops

For loop:

\[
\text{start increment end } \{ \text{proc} \} \text{ for}
\]

- For each value in the range, pushes that value on the stack and executes proc
- Can count up or down
- Counting can be either integer or real
- Have the usual "compare for equality" problem with reals
Loop Example

%! /Bookman-Demi findfont 80 scalefont setfont
/fancyprint { % stack: (string)
  gsave
currentpoint translate % desired loc=>origin
  1 −.05 −0.01 {
    dup 20 mul dup 2 mul moveto setgray
dup show % print gray, varying loc
  }
} for
2 2 moveto 1 setgray show % white on top
  grestore
} bind def

170 325 moveto
(rocks!) fancyprint
80 400 moveto
(PostScript) fancyprint
showpage
Recursion

%! /depth 0 def /maxdepth 1 def
/down { /depth depth 1 add def } def
/up { /depth depth 1 sub def } def

/DoLine {
    0 500 rlineto currentpoint stroke
    translate 0 0 moveto
} def

/Arrow {
    gsave .65 .65 scale 10 setlinewidth
    down DoLine
    depth maxdepth le {
        135 rotate Arrow –270 rotate Arrow
    } if
    up grestore
} def

300 300 moveto
Arrow stroke
showpage
Recursion

\[
%!
/depth 0 def /maxdepth 2 def
/down { /depth depth 1 add def } def
/up { /depth depth 1 sub def } def

/DoLine {
  0 500 rlineto currentpoint stroke
  translate 0 0 moveto
}
def

/Arrow {
  gsave .65 .65 scale 10 setlinewidth
down DoLine
depth maxdepth le {
  135 rotate Arrow –270 rotate Arrow
}
if
  up grestore
}
def

300 300 moveto
Arrow
stroke
showpage
Recursion

%! /depth 0 def /maxdepth 3 def
/down { /depth depth 1 add def } def
/up { /depth depth 1 sub def } def

/DoLine {
  0 500 rlineto currentpoint stroke
  translate 0 0 moveto
} def

/Arrow {
  gsave .65 .65 scale 10 setlinewidth
  down DoLine
  depth maxdepth le {
    135 rotate Arrow –270 rotate Arrow
  } if
  up grestore
} def

300 300 moveto
Arrow
stroke
showpage
%!
/depth 0 def /maxdepth 10 def
/down { /depth depth 1 add def } def
/up { /depth depth 1 sub def } def

/DoLine {
  0 500 rlineto currentpoint stroke
  translate 0 0 moveto
} def

/Arrow {
  gsave .65 .65 scale 10 setlinewidth
  down DoLine
  depth maxdepth le {
    135 rotate Arrow –270 rotate Arrow
  } if
  up grestore
} def

300 300 moveto
Arrow
stroke
showpage
Character Outlines

Font descriptions can be stored as:

1) outlines that are filled (most fonts use this)
2) lines to be stroked
3) bit maps

The "charpath" operator can be used with outlined and stroked fonts

- Takes a string argument
- Adds the outlines of the characters to the current path
Character Outlines

%! 
/ZapfChancery–MediumItalic findfont
100 scalefont setfont 
/outline { % stack: (string) 
    true charpath stroke 
} def 

100 400 moveto (PostScript) outline showpage
Clipping

The interpreter maintains a `clipping` path

- No ink is put on the page outside of the clipping path
- Initially set to the page boundaries
- Can be set smaller by intersecting it with another path
- Cannot be made larger (use gsave/grestore)
Images

Image operator reads binary image data from a string on the stack.

Takes five arguments:

1) Number of pixels per scan line
2) Number of scan lines in the image
3) Bits per pixel (1, 2, 4, 8)
4) Transform matrix that maps the samples onto a one-unit-square imaging region
5) A procedure that produces the data strings; called repeatedly if necessary

Usually use translate/scale/rotate rather than the matrix to position and size the image.
Images

%!
234 400 translate
144 144 scale
8 8 1 [8 0 0 8 0 0] { <c936> } image
0 –1.5 translate
8 8 1 [8 0 0 8 0 0] { <c936ab54> } image
showpage

% Note: 1=white, 0=black
Images

Variable density text
1) Get character outlines of text
2) Set clipping path
3) Paint variable density image
4) Stroke text outline path
%!
/AvantGarde--Book findfont 100 scalefont setfont

/str 256 string def  % create a string
0 1 255 { str exch dup put } for % fill with 0 1 2 ...
75 500 translate  % start image here
0 25 moveto  % move up for descenders
(PostScript) dup true charpath
gsave clip  % define clip path
stringwidth pop dup scale % scale to size of word
256 256 8 [256 0 0 256 0 0] {str} image
grestore stroke  % restore path and stroke
showpage

PostScript

Images
Making a Poster

Paint the page over and over

- Scale the image to make it bigger
- Translate it to paint different portions each time
Making a Poster

Big Sale Today!
50% off everything in the store!
TODAY ONLY!

%!
/poster { % stack: {proc}
  /proc exch def
  gsave
  .5 i .65 i translate
  gsave
  frame
  .824 .824 scale
  0 0 moveto
  proc
  grestore
  4 i 0 translate
  gsave
  frame
  .824 .824 scale
  -4.25 i 0 moveto
  proc
  grestore
  % etc.

Rochester Institute of Technology
Department of Computer Science

Copyright © 2004 K. Reek
Languages for Lunch 2/6/2004
Redefining Showpage

Change the way a document is printed
- Overprint something (e.g. DRAFT) on each page
- Another way to make a poster

Method:
1) Save old showpage with a new name
2) Redefine showpage to make the desired change
Redefining Showpage

%!
/oldshowpage /showpage load def
/showpage {
    gsave
    /Helvetica findfont 200 scalefont setfont
    52 rotate % rotate to diagonal
    (DRAFT) dup
    stringwidth pop % compute length
    1000 exch sub % get remaining len
    2 div -80 moveto % center it
    true charpath stroke % outline it
    restore
    oldshowpage
} bind def

% PostScript for the original document
%!
/ZapfChancery–MediumItalic findfont
100 scalefont setfont
80 400 moveto
(Hello world!) show
showpage
% End of original document
Nested Documents

A PostScript document can include another PostScript document in its entirety

- Set translation and scaling to put included document where you want it
- Redefine showpage to prevent the included document from rendering the page prematurely
Nested Documents

Included Page:

R·I·T

Included Documents

%!
% defs not shown: font, center, i

/included_page {
  % PostScript to paint the page
  showpage
}
def

4.25 i 8 i moveto (Included Page:) center
/state save def
userdict maxlength dict begin
/showpage {} def
0 8 i translate
-90 rotate
8.5 i 11 i div dup scale

included_page

end
state restore
showpage
Fonts Available

%! 
/v 20 def 
/y 750 def 
/Courier findfont v scalefont setfont 
/printname { % stack: fontname font 
    pop % pop the font itself 
    /y y v sub def 
    y 30 lt { showpage /y 750 def } if 
    /Courier findfont v scalefont setfont 
    dup 100 string cvs 
    72 y moveto show ( ) show 
    dup findfont v scalefont setfont 
    100 string cvs show 
} bind def 
FontDirectory { printname } forall 
showpage 

Rochester Institute of Technology 
Department of Computer Science
Nonlinear Text

PostScript is fun!

%!  
/Palatino-Roman findfont 60 scalefont setfont
100 500 moveto
32 rotate
(PostScript is fun!) {
  (X) dup 0 % stack: int (X) (X) 0
  4 -1 roll % stack is now: (X) (X) 0 int
  put % copy int (our char) into string
  show % print the string
  -4 rotate
} forall
showpage
There is More

1) Kerning, expanding to right adjust
2) Modifying/defining fonts, downloadable fonts
3) Image mask
4) Color (HSB, RGB)
5) Halftone screens, color separations
6) etc. etc. etc.
Thank you!