

Postscript Intro

CSE 413, Autumn 2005
Programming Languages

<http://www.cs.washington.edu/education/courses/413/05au/>

References

- *Postscript Language Reference*, Adobe
- *Postscript Language Tutorial and Cookbook*, Adobe
- Adobe's Postscript web site
 - » <http://www.adobe.com/products/postscript/main.html>
- Postscript resources, Jim Land
 - » <http://www.geocities.com/SiliconValley/5682/postscript.html>
- Postscript resources, Doug Zongker (UW)
 - » <http://isotropic.org/uw/postscript/>
- First Guide to PostScript, Peter Weingartner
 - » <http://www.cs.indiana.edu/docproject/programming/postscript/postscript.html>

What is Postscript?

- Page description language
 - » device independent way of representing a 2D page
 - » emphasis on scalable text, graphics presentation
- Simple programming language
 - » stack oriented
 - » interpreted
- Fundamental tool for 2D display



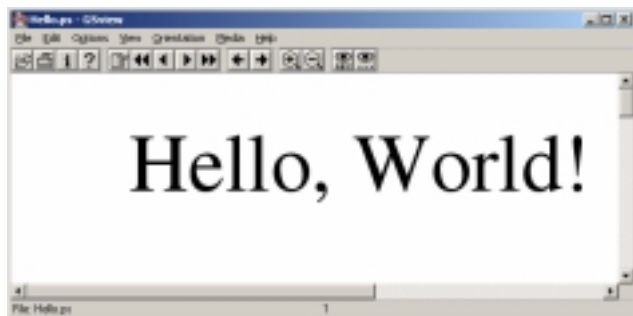
images from Doug Zongker papers

Hello World! in Ghostscript



Hello World! in GSView

```
%!PS
/Times-Roman findfont 50 scalefont setfont
72 720 moveto
(Hello, World!) show
showpage
```



Page Description Language

- Graphics operators
- Text
 - » any position, orientation, scale
- Geometric figures
 - » straight and curved lines
 - » filled spaces
- Sampled images
 - » digitized photos, sketches, images



Imaging Model ...

- Current page
 - » independent of the device on which the image will actually be drawn - device independence
 - » starts out empty, then painting operators are used to add opaque inking to the page
- Current path
 - » set of points, lines and curves
 - » path itself is not a mark on the page, it's just a path
 - » inking is done by stroking and/or filling the path



... Imaging Model

- Clipping path
 - » boundary of the area that may be drawn upon
 - » starts out matching the size of the default page area
 - » may be set to any path



Coordinate Systems

- A Postscript program describes positions in terms of the *user coordinate system* or *user space*
 - » independent of the printer's device coordinates
 - » units are 1/72 inch, approximately one point
- Positions on a page are described as (x,y) pairs in a coordinate system imposed on the page
 - » this is the device space
- Coordinates in *user space* are automatically translated to *device space* when the page is printed

User Coordinates


```
/boxpath {
  newpath
  0 0 moveto
  0 72 rlineto
  72 0 rlineto
  0 -72 rlineto
  closepath
} def

36 700 translate
boxpath
0 setgray
fill

72 -14.14 translate
45 rotate
boxpath
2 setlinewidth
0.8 setgray
stroke
```



Programming Language ...

- A nice and relatively compact programming language
- Stack oriented
 - » operand stack, dictionary stack
- Postfix notation
 - » operators work on data from the stack
 - » $3+4 \Rightarrow 3\ 4\ \mathbf{add}$
 - » describe a bezier curve \Rightarrow 
 - » `72 720 moveto 82 750 92 690 102 720 curveto`

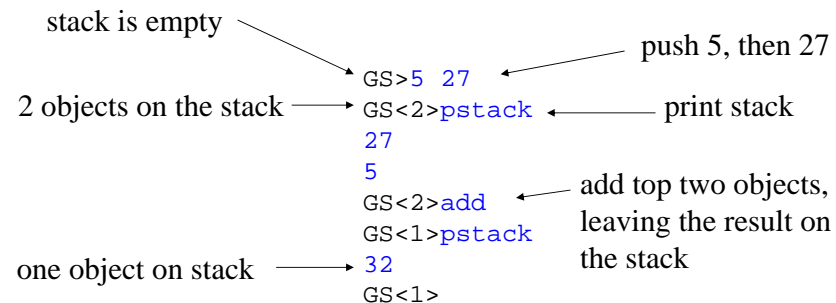
... Programming Language

- Data Types
 - » includes reals, booleans, arrays, strings
 - » procedures are executable arrays
- Flexible
 - » dictionaries to store user defined objects
 - » static and dynamic graphics capabilities
- Accessible source
 - » Printable file is actually just a text program that can be edited with any text editor

Stack

- The stack is an area of memory where Postscript objects can be stored until consumed
 - » last in, first out
 - » push and pop are generally implicit. Numbers are always pushed as they are encountered. Operators can consume objects off the stack and push new objects onto the stack
 - » several operators work directly on the stack to move the top few elements around

Stack operation



Operators

- An operator is a word that causes the Postscript interpreter to carry out some action
- PS searches internal dictionaries to see if the word is an operator name
 - » If listed, PS looks up the associated definition and executes it
 - » Many predefined operators
 - » Can define new operators as needed

Defined Arithmetic Operators

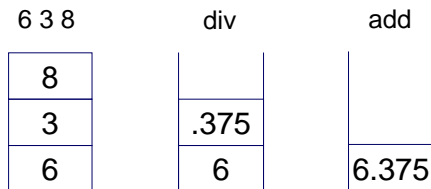
- Predefined arithmetic operators include
 - » add, div, idiv, mod, mul, sub
 - » abs, neg
 - » ceiling, floor, round, truncate
 - » sqrt
 - » atan, cos, sin, exp, ln, log
 - » rand, srand, rrand

Stack operations

- Objects on the operand stack are consumed from the top: last in, first out

- » $6 + (3 / 8)$
- » 6 3 8 div add

```
GS>6 3 8
GS<3>div
GS<2>add
GS<1>=
6.375
GS>
```



Stack operations

Postscript also allows you to rearrange and duplicate the top elements of the stack so that the needed operands are on top

```
GS>4 5 % stack has two objects on it now
GS<2>2 array % Create a 2-element array and leave reference on stack
GS<3>dup % duplicate the reference
GS<4>0 % we are going to set element 0
GS<5> % stack is now: 4 5 arr arr 0
GS<5>5 -1 roll % stack is now: 5 arr arr 0 4
GS<5>put % stack is now: 5 arr
GS<2>dup 1 % stack is now: 5 arr arr 1
GS<4>4 -1 roll % stack is now arr arr 1 5
GS<4>put % stack is now: arr
GS<1>pstack
[4 5]
GS<1>
```

dup, pop and roll

- dup
 - » duplicate the top item on the stack. Useful for array references that are consumed by operations like put and get
- pop
 - » remove top element from stack and discard
- roll
 - » roll stack contents. Consumes two numbers from top of stack. Top number is how many times and what direction to rotate, second number is how many objects are to be rotated
 - 7 8 9 3 1 roll \Rightarrow 9 7 8
 - 7 8 9 3 -1 roll \Rightarrow 8 9 7

Operand Stack Operators

- pop
- exch
- dup, copy, index
- roll
- clear, count
- mark, cleartomark, counttomark

Interactive Operand Stack Operators

- **==**
 - » pops an object from the stack and produces a text representation of the object as best it can
- **pstack**
 - » prints the contents of the stack, but does not remove anything from the stack

Arrays

- Postscript arrays are 1-dimensional collections of objects
 - » indexed from 0
 - » objects can be of any type - integers, strings, other arrays, dictionaries, etc
- Similar to Object arrays in Java, although Postscript arrays can hold primitive elements as well as reference elements

Creating an array

- An array can be created by directly specifying the elements enclosed in square brackets
 - » [16 (twelve) 8] creates a 3-element array
 - number 16, string "twelve", number 8
 - » [(sum) 6 14 add] creates a 2-element array
 - string "sum", number 20
 - note that the operator **add** is executed
- An array can be created with the **array** operator
 - » 3 array creates a 3-element array of all null

Array example

There are now two array references on the stack. →

```
GS>[1 2 3]           create an array
GS<1>pstack
[1 2 3]
GS<1>3 array         create an empty array
GS<2>pstack
[null null null]
[1 2 3]
GS<2>dup 0 (Hi!) put  set element 0 to string
GS<2>pstack
[(Hi!) null null]
[1 2 3]
GS<2>exch dup 2 333 put set element 2 to 333
GS<2>pstack
[1 2 333]
[(Hi!) null null]
GS<2>
```

Array bracket operators

[

- » put a mark on the stack. The following elements are going to be scooped up in an array

]

- » create an array containing all elements back to the topmost mark. The array is created on the heap in virtual memory, and an array reference is left on the stack

example: [1 1 1 add] ⇒ 2-element array : [1 2]

Array operators put and get

- *array index any put*
 - » puts *any* at *index* of *array*
 - » removes all three objects from the stack
- *array index get*
 - » gets the object at *index* of *array*
 - » removes both objects from the stack and returns the indexed element on the top of the stack
- Note that if you want to use the array reference again you need to **dup** it or name it

Array operators

- array, [,]
- length
- get, put, getinterval, putinterval
- astore, aload, copy
- forall

```
GS>0 [ 1 2 3] {add} forall ==
```

```
6
```

```
GS>
```