# Topic #15: Postscript Intro

CSE 413, Autumn 2004
Programming Languages

http://www.cs.washington.edu/education/courses/413/04au/



## 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 (was 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

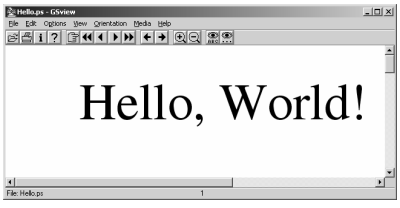
images from Doug Zongker papers



## Hello World! in Ghostscript


## Hello World! in GSView

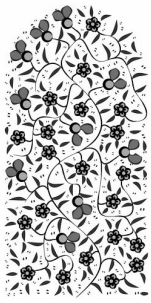
```
%!PS
/Times-Roman findfont 50 scalefont setfont
72 720 moveto
(Hello, World!) show
showpage
```






## 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
  - » 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 ⇒ 3 4 **add**
  - » describe a bezier curve ⇒

    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

```
GS>5 27
GS<2>pstack
27
5
GS<2>add
GS<1>pstack
32
GS<1>
```



## 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

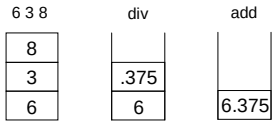
- Predefined arithmetic operators include
  - » add, div, idiv, mod, mul, sub
  - » abs, neg
  - » ceiling, floor, round, truncate
  - » sqrt
  - » atan, cos, sin, exp, ln, log
  - » rand, srand, rrand



## 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 ⇒ 9 7 8
    - 7 8 9 3 -1 roll ⇒ 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
- An array can be created with the array operator
  - » 3 array creates a 3-element array of all null



## Array example

```
GS>[1 2 3]
GS<1>pstack
[1 2 3]
GS<1>3 array
GS<2>pstack
[null null null]
[1 2 3]
GS<2>dup 0 (Hi!) put
GS<2>pstack
[(Hi!) null null]
[1 2 3]
GS<2>exch dup 2 333 put
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>
```



## Variables and Control Flow



## Variables

- Postscript uses dictionaries to associate a name with an object value
  - » /x 3 def
    - associate value 3 with key x
  - » /inch {72 mul} def
    - define function to convert from inches to points
- Postscript dictionaries are similar to the HashMaps that our SymbolTable class maintains in the compiler
  - » key / value pairs



## Several Dictionaries

- When the interpreter encounters a name, it searches the current dictionaries for that key
- At least three dictionaries are always present
  - » user dictionary
    - writeable dictionary in local virtual memory associates names with procedures and variables for the program
  - » global dictionary
    - writeable dictionary in global virtual memory
  - » system dictionary
    - read-only dictionary associates keywords with built-in actions

## Dictionary Stack

- References to the dictionaries are kept on the dictionary stack
- Interpreter looks up a key by searching the dictionaries from the top of stack down
  - » search starts with *current dictionary* on top of stack
  - » initially, user dictionary is top of stack
  - » system dictionary is bottom of stack
  - » can define and push additional user dictionaries on top

31

## Virtual Memory

- Postscript environment includes stacks and virtual memory
- Operand stack contains simple objects (eg, integers) and references to composite objects (eg, strings, arrays)
- Virtual memory (VM) is a storage pool for the values of all composite objects

32

## save and restore

- Simple user programs define their objects in local VM
- The save operator makes a snapshot of local VM
- The restore operator throws away the current local VM and restores the state from the last save
- Local VM with save/restore pairs is used to encapsulate information whose lifetime conforms to a hierarchical structure like a page

33

## Defining and using a variable

- Define a variable ppi and give it a value
  - » /ppi 72 def
  - » push the name ppi on the operand stack as a literal
  - » push the number 72 on the operand stack
  - » pop both items and store in the current dictionary using ppi as the key and 72 as the value
- Use the variable's value
  - » ppi 2 mul
  - » find the value of ppi (72) and push it
  - » push the number 2
  - » pop both operands, multiply, push the result

34

## Defining and using a procedure

- Define a procedure name and give it a value
  - » /inch {72 mul} def
  - » push the name inch on the operand stack as a literal
  - » push mark, 72, mul on the operand stack
  - » pop to the mark, create an executable array, and store in the current dictionary using inch as the key and the executable array as the value
- Use the procedure's value
  - » 2 inch
  - » push the number 2
  - » look up the name inch, find the procedure, execute
  - » push 72, pop both numbers, multiply, push the result

35

## fm constructors are procedures

```
% Circle constructor.

% FM call format => Circle(radius)
% PS call format => radius Circle.Circle
% Result: Reference to a fields array with
%         values set by arguments or defaults.

/Circle.Circle {
Circle.fields.SIZEOF array % Create the array
dup Circle.fields.radius   % radius field
4 -1 roll put              % store radius

dup Circle.fields.grayfill 0.5 put
dup Circle.fields.graystroke 0.0 put
dup Circle.fields.linewidth 1.0 put
} def
```

36

## Boolean operators

- Comparison operators
  - » eq, ne, gt, lt, ge, le
- logical operators
  - » not, and, or, xor
  - » true, false

```
GS>2 3 ge
GS<1>==
false
GS>2 2.0 eq ==
true
GS>(abc) (acc) lt ==
true
GS>[1 2 3] dup eq ==
true
GS>[1 2 3] [1 2 3] eq ==
false
GS>
```

37

## Conditionals and loops

- There are several operators for specifying the flow of control in a Postscript program
- Executable arrays are a basic element for the control flow operators
  - » the code block (executable array) is defined in-line
  - » {add 2 div} - calculate 2-value average
  - » the curly brackets defer interpretation of the code and force the creation of a new executable array (procedure) object

38

## if, ifelse operators

- Take a boolean object and one or two executable arrays on the stack.
- Select and execute one of the executable arrays depending on the boolean value
- leaves nothing on the stack

  - » *bool proc* if
  - » *bool* $proc_1$ $proc_2$ ifelse

39

## an if example

```
% if current point beyond right margin, do LF CR.

/chkforendofline
{

currentpoint pop           % discard y position
RM gt                      % current x > right margin?
{
0 lineheight neg translate % "linefeed"
LM 0 moveto                % "carriage return"
} if

} def
```

40

## repeat operator

- Repeat a procedure body n times
- *n proc* repeat

  GS>1 2 3 4 3 {pop} repeat
  GS<1>==

41

## for operator

- Controls the standard indexed counting loop

- *initial increment limit proc* for
  - » the control value is calculated
  - » if greater than limit, the loop exits
  - » otherwise the control value is pushed and the procedure is executed

```
GS>1 1 4 {} for
GS<4>pstack
4
3
2
1
GS<4>clear
GS>0 1 1 4 {add} for
GS<1>==
10
GS>
```

42

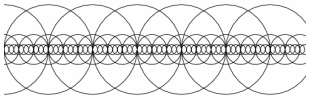
## loop and exit operators

- Repeat a procedure an indefinite number of times, usually until some condition is met
- The loop operator takes a procedure and executes it until an exit command is encountered within the procedure
- *proc* loop
  - » there must be an exit encountered within the body of the procedure, or the code will loop forever



## loop example

```
% call: radius y lineofcircles
/lineofcircles {
/ypos exch def
/radius exch def
/xpos 0 def
{xpos pagewidth le
  { doCircle increase-x }
  { exit }
  ifelse
} loop
} def
```



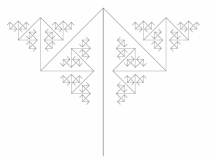


## Recursion

- A loop can be set up in a program by having a procedure call itself
  - » remember Scheme?
  - » recursion must always:



## Recursion example



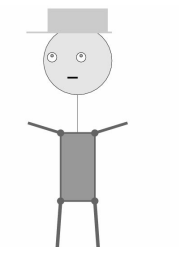
```
/fractArrow {
gsave
kXScale kYScale scale
kLineWidth setlinewidth
down
doLine
depth maxdepth le
{135 rotate fractArrow
-270 rotate fractArrow
} if
up
grestore
} def
```



## Text and Graphics



ex23\tiger.ps, from ghostscript examples



## Paths

- A drawing starts with a path on the current page
- path is a set of straight lines and curves that define:
  - » a region to be filled (fill)
  - » a trajectory that is to be drawn (stroke)

```
newpath
95 700 40 0 360 arc
closepath
1 .5 0 setrgbcolor
fill
```

```
newpath
80 720 30 0 360 arc
closepath
0.1 setgray
gsave
5 setlinewidth
stroke
grestore
1 .9 0 setrgbcolor
fill
```



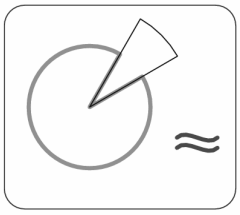
## basic path construction operators

- newpath
  - » initialize current path to be empty
- closepath
  - » Connect subpath back to its starting point
- moveto, rmoveto
  - » set current point to (x,y)
  - » set current point to (curX+dx, curY+dy)
- lineto, rlineto
  - » append straight line to (x,y)
  - » append straight line to (curX+dx, curY+dy)



## Curve path operators

- arc, arcn
  - » append clockwise, counterclockwise arcs
  - » x y r $angle_1$ $angle_2$ arc
- arct, arcto
  - » append tangent arcs
  - » $x_1$ $y_1$ $x_2$ $y_2$ r arct
- curveto, rcurveto
  - » append Bezier curve
  - » $x_1$ $y_1$ $x_2$ $y_2$ $x_3$ $y_3$ curveto



Arcs&Curves.ps



## saving the graphics state

- Sometimes we need to save the current graphics state (including the path) so that we can reuse it
  - » the stroke and fill operators clear the current path
  - » blocks of code may change path, gray value, line width, user coordinate system, etc
- gsave
  - » save a copy of the current state on the graphics state stack
- grestore
  - » restore to the state at the time of the last save



## Text

- Postscript treats text as just another way to define graphics paths
  - » The content of the text is maintained in a string object
  - » The visible representation of the text is determined by the font
  - » Fonts are stored as a set of curves for each letter
    - the representation is the *glyph* for this character in this font

Font: Tannarin BT FONT DESIGN IS FUN!

Font: Murray Hill *Font design is fun!*

http://www.myfonts.com/



## Using a font

- Find the information describing the font
  - » the info is in a font dictionary
  - » use the findfont operator
- Scale the font to the size needed
  - » original font is 1 unit high (usually 1 point)
  - » use the scalefont operator to scale
- Set the scaled font as the current font
  - » use the setfont operator

## Show a text string

```
/Palatino-Italic findfont
15 scalefont
setfont
72 720 moveto
(Font design is fun!) show

/NewCenturySchlbk-Roman 15 selectfont
72 700 moveto
(Font design is fun!) show

/StandardSymL 15 selectfont
72 680 moveto
(Font design is fun!) show
```

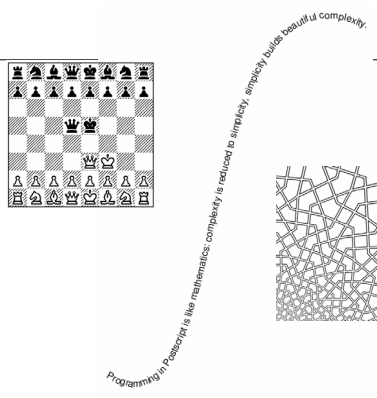
*Font design is fun!*
Font design is fun!
Φοντ δεσιγν ισ φυν!



## Fun with fonts

- Postscript provides much more power for dealing with fonts
  - » fonts are paths - they can be filled, stroked, clipped to, etc
  - » there are several glyph painting operators that provide a variety of width modification effects
  - » numerous font type definitions to support different ways of identifying the characters and defining the glyphs

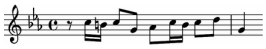





chess.ps
snake.ps
spiral.ps



## "The name is Pond ... LilyPond"



http://lilypond.org/web/index.html

- LilyPond is an "automated engraving system." It formats music beautifully and automatically, and has a friendly syntax for its input files.
  - » input is done in the form of a textual music language
  - » content (the music) and the layout are strictly separated
  - » users can extend the program by using the built-in Scheme interpreter.
  - » PostScript output is generated via the TeX typesetting system.



## Graphviz

- graphviz is a set of graph drawing tools
  - » dot - makes hierarchical layouts of directed graphs
  - » neato - makes "spring" model layouts of undirected graphs
- Graphs are described in DOT language
  - » abstract grammar defining DOT

    graph:[ **strict** ] (**graph** | **digraph**) [ ID ] '**{**' stmt_list '**}**'

    stmt_list:[ stmt [ '**;**' ] [ stmt_list ] ]

    etc
- Output in Postscript and other languages

http://www.research.att.com/sw/tools/graphviz/



```
strict graph ip_map {
        MH -- { ERC ALC WH1 HO1 IH1 };
        ALC -- MV -- AN;
        MV -- HO3 -- WH1;
        HO3 -- MT;
        MT -- LZ -- FL;
        MT -- FJ;
        MT -- ER;
        HO1 -- HR;
        MT -- DR;
        HO1 -- CB -- IH1 -- IHC;
        HO1 -- HV;
        {IHP IW IH1 IH2 IH4 } -- {IHP IW IH1 IH2 IH4 };
        {IH4 IH2} -- {MLM HV};
        HO1 -- HO3;
        HO3 -- IH1;
        IH2 -- IH4 [len=4];       // hack
}
```