PostScript Internals

15-463 Graphics II
Spring 1999
Background

❖ PostScript raster image processor for Mac
  ● All Level 1 features
  ● Some support for color and multi-bit devices

❖ Undergrad independent study: MacRIP

❖ Commercial product: TScript
  ● Sold by TeleTypesetting Co.
  ● Still around (!)
PostScript Features

- Device/resolution independence
- Orthogonality
  - Vector shapes, images, text treated uniformly
  - e.g. transforms and clips images and text
- “Composability”
- Complete language
- High-quality outline fonts
Focus

Level 1 implementation
- Level 2 adds many complex features
- Level 3 adds even more

Laser printer-like output device
- One bit per pixel
- Medium resolution: ~300 dpi
  - 2400x3000 pixels on a page = 1Mb frame buffer
- Non-interactive/batch model
Topics

Language Overview
Language Implementation
Graphics Overview
Scan Conversion and Clipping
Fonts
Images and Halftones
Language Overview
Syntax

Stream of tokens with little structure
- Postfix notation
- No precedence, lexical scope, etc.

Tokens
- Integer and real: 3  4.0  5e6
- String: (Call the doctor.)
- Name: John  yaya  3plus4  ==proc
- Procedure: {add 2 div}
More Object Types

- Array: vector of arbitrary objects
- Dictionary: finite mapping on objects
- Operator: built-in procedure
- Boolean: true and false
- Null
- Mark
Stacks

- Operand stack: accumulates arguments
- Execution stack: object to evaluate next
- Dictionary stack: explicit variable scope
- Types checked at run time
  - All objects have an inherent type
Object Attributes

Literal: push to the operand stack

Executable
- Name: look up on dictionary stack
- Array: execute elements in order
- String: parse and execute code
- Operator: execute built-in operation

Access
- unlimited > read-only > execute-only > none
Virtual Memory

▷ *Virtual memory* is just the allocation heap

▷ **save** “snapshots” all mutable objects
  - Strings, arrays, dictionaries

▷ **restore** returns virtual memory to a previous snapshot
  - All intervening mutations are undone
  - Throw away all new objects

▷ Good for batch processing model
Language Implementation
Object Representation

```c
struct object {
    unsigned short type:4, exec:1, access:2;
    unsigned short length;
    union {
        int integer;
        float real;
        unsigned char *string;
        struct name *name;
        struct object *array;
        struct dict *dict;
        unsigned int operator;
        int boolean;
    } u;
};
```
Dictionary Representation

- Typically a hash table based on *keys*
- Corresponding *values* in parallel array

```c
struct dict {
    unsigned int access;
    unsigned short length;
    unsigned short maxlength;
    struct object *keys[maxlength];
    struct object *values[maxlength];
};
```
Name Representation

- Typically a global hash table for all names
- Cache with current binding for fast lookup

```c
struct name {
    struct name *next;
    struct object cache;
    unsigned short hash;
    unsigned short length;
    unsigned char string[length];
};
```
Implementing Virtual Memory

Allocate objects linearly from a large arena

save remember current allocation pointer

restore resets allocation pointer

What about mutated values?

- Could just block copy active heap: slow!
- Better to save location on first modification
- restore just walks through the “undo list”
Graphics Overview
Path

Sequence of line and curve segments
- Need not be connected or closed
- Connected sequences of segments are subpaths

Specified by path elements
- **moveto** starts a new, disconnected subpath
- **lineto** specifies a connected line
- **curveto** specifies a connected, cubic Bézier
- **closepath** connects an open subpath to its start
Graphics State

⁻ Collects parameters for graphics operators
  * Operators implicitly refer to current gstate
⁻ Saved and restored by gsave and grestore
⁻ Some specific parameters
  * Current matrix allows affine transformations
  * Current color is color to paint with
  * Current path is shape to fill or outline
  * Current clipping path restricts painted area
  * Current font determines appearance of text
Graphics Operators

❖ **fill** paints inside of current path
  - Uses non-zero winding number rule
  - Permits arbitrary self-intersections
  - Implicitly closes all open subpaths

❖ **stroke** outlines current path

❖ **image** renders a rectangular pixmap

❖ **show** renders a string using current font
Scan Conversion and Clipping
Flattening Curves

Flattening approximates curves by lines
- Current flatness parameter limits deviation (in pixels) from true curve

\texttt{flattenpath} flattens current path (in place)

Recursive subdivision can work well

Forward differencing has a faster inner loop

\begin{align*}
x[t+1] &= x[t] + dx[t] \\
dx[t+1] &= dx[t] + ddx[t] \\
ddx[t+1] &= ddx[t] + dddx[0]
\end{align*}
Approximating Circular Arcs

Arcs are approximated by cubic Béziers
- *Required*, since user can iterate over paths
- Some affine transformations of arcs are not arcs

Each arc segment $\leq 90^\circ$ gets one curve

Control points are along tangents to arc
- $F = \frac{4}{3} \left( \frac{1}{1 + \sqrt{1 + (d/r)^2}} \right)$
Filling Flattened Paths

- Can use active edge lists (Foley+van Dam)
- Linear DDA doesn’t need edge structures
- clear x transition lists
  - loop curve segments in current path
  - loop t using curve DDA
  - loop y using line DDA
    - store x coordinate on transition list for y
- repeat for clip path
- sort transition lists
- fill intersection of “inside” intervals according to rule
Stroking Flattened Paths

- Stroke of a path is a path itself
- Precise specification of line shape
  - Current *line width*
  - Current *line join*
  - Current *line cap*
- **strokepath** replaces path with its stroke
- Special case for rendering zero-width lines
Clipping Flattened Paths

- **clip** intersects current path and clip path
- Computes polygon intersections
- Scan convert path and clip in parallel
  - Use interior of both paths for rasterization
- Can generate trapezoids from modified scan converter
  - Sample at segment extrema and intersections
  - Reconstruct original segments, where possible
Fonts
Font Representation

Fonts come in two flavors
- Type 1 are condensed path descriptions
- Type 3 are ordinary PostScript programs

Font matrix defines character coordinates

Font encoding maps character codes to character names

Font cache retains bitmaps for most commonly used characters
Type 3 BuildChar

Algorithm:
- Check font cache for character mask
- Concatenate font matrix with current matrix
- Call BuildChar with font dictionary and character code
- Save bits in font cache, if appropriate

Typical BuildChar procedure:
- Look up character name in Encoding vector
- Set character width and bounding box
- Construct path for character outline
- fill
Type 1 Font Hints

ミューズ rasterizer at low resolutions

Blue values declare standard heights of character features (from baseline)

Stem width hints declare standard widths of character features

Character stem hints identify stems in character outlines
Interpreting Font Hints

ẑ All feature heights for a given blue value are rounded consistently
  • “Fuzz” parameter is slop for matching heights
ẑ All standard stem widths are rounded consistently
ẑ *Overshoot suppression* gives “flat” and “round” characters same height
ẑ *Flex feature* straightens shallow curves
Images and Halftones
Images

- **image** specifies absolute color values
- **imagemask** pours color through a stencil
- **Matrix** specifies pixel coordinate system
- **Procedure** supplies pixel/bitmap values
Image Rendering

- Reverse sample through inverted matrix
- Scan convert clip path as additional mask
- Use anti-aliasing for multi-bit devices
Halftones

Laser printers can’t place pixels in isolation => Can’t use standard dithering techniques

Frequency specifies cells per inch

Angle specifies orientation of grid lines

Spot procedure determines shape of cells
  Circular spots are typical

Example: 60 lpi = 25 grays at 300 dpi
Halftone Rendering

★ Offset cells into a repeating tile
  ● Usually, only discrete angles are available
★ Call spot function on pixel centers
★ Set $n$ pixels with least spot values
  ● $n = \text{round}((1-\text{gray}\_\text{level}) \times \text{spot}\_\text{area})$
Extensions

❖ Multi-bit devices
❖ Level 2
   • Forms and patterns
   • Color spaces
   • User paths and graphics states
❖ Display PostScript
   • Concurrency
   • View clip
References

☞ Adobe Type 1 Font Format
☞ “Tutorial on Forward Differencing”, Bob Wallis, Graphics Gems I
☞ “Fast Scan Conversion of Arbitrary Polygons”, Bob Wallis, Graphics Gems I