High Performance GUIs

Kevin Ellis

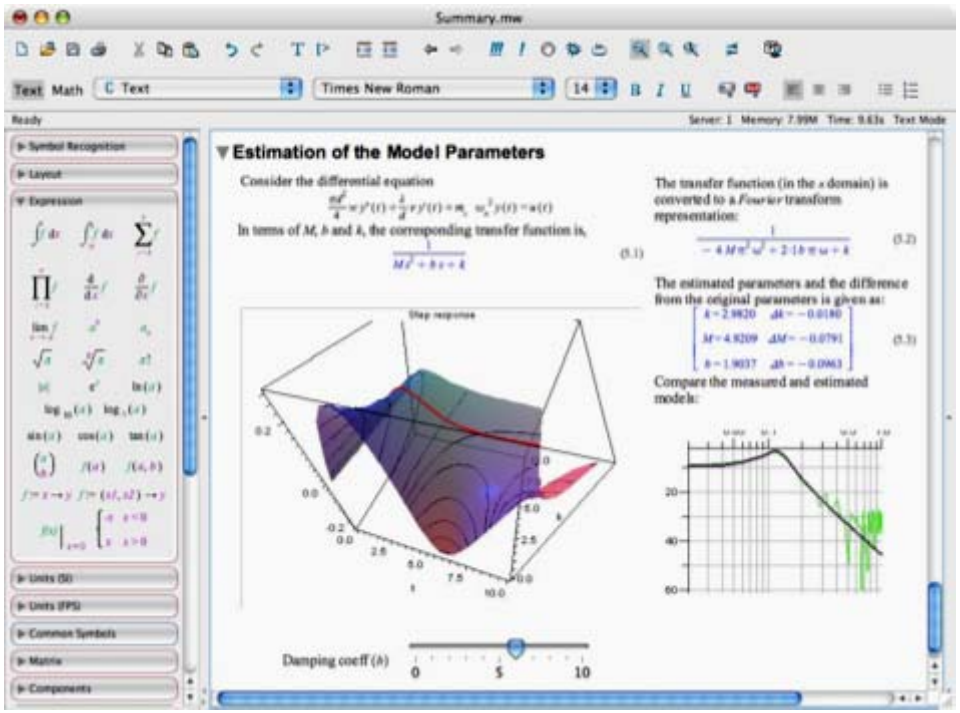
GUI Manager
Maplesoft
www.maplesoft.com

TS-1305

Goal

To explore the inner workings of the graphics API and demonstrate strategies for optimizing the performance of a graphical user interface

Solid Performance Is Possible in Java™ Technology



Maple 10 Worksheet

Agenda

Overview of the Graphics API

Graphics Pipeline

Graphics Primitives and Tools

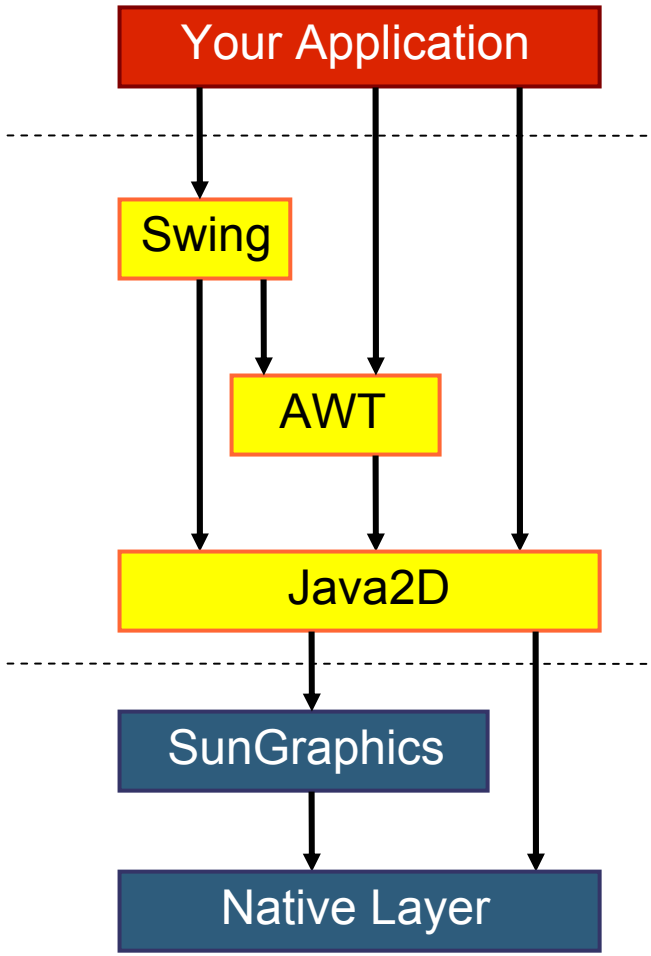
Optimization

Demo

Summary

Overview of the Graphics API

- Interact with several layers of the graphics API
- Design choices at higher levels can affect performance at lower levels
- Bottom two layers are vendor specific



Overview of the Graphics API

- At what level should I be focusing my optimization efforts?

Overview of the Graphics API

- At what level should I be focusing my optimization efforts?
  - Depending on the application, all levels may be important
    - Swing: Optimize object creation
      Set rendering flags
    - AWT: Optimize event handling
    - Java2D: Direct control over object rendering
      Graphical underpinnings to Swing and AWT

Agenda

Overview of the Graphics API

Graphics Pipeline

Graphics Primitives and Tools

Optimization

Demo

Summary

Graphics Pipeline

- Graphics and Graphics2D are interfaces
- Vendor specific implementations for onscreen rendering
- Some optimizations are not cross-platform
  - Notable differences in Sun and Apple implementation of text and antialiasing support

Evolution of the Graphics Pipeline

- J2SE™ 1.4
  - Improved data sharing across pipelines
  - Hardware acceleration for offscreen images
  - Pluggable image I/O framework
  - OpenType fonts
- Java™ SE 6
  - Improved text antialiasing
  - Single-threaded rendering
  - Curved primitive rasterization
- J2SE™ 1.5
- Hardware acceleration using OpenGL
  - Text rendering performance
  - Improved font handling

Graphics Pipeline

- Ideally, built-in optimizations would provide necessary performance
- In reality, further optimizations are sometimes required to reduce the burden on the graphics pipeline

Agenda

Overview of the Graphics API

Graphics Pipeline

Graphics Primitives and Tools

Optimization

Demo

Summary

Graphics Primitives and Tools

- Graphics
  - Lines
  - Polylines
  - Circles
  - Arcs
  - Text
  - Image
  - Clipping
  - Affine transforms
- Graphics2D
  - Shape
  - Composite
  - Rendering hints

Graphics Primitives and Tools

- Graphics2D offers more flexibility
- Greater choice → More ways to address performance issues
  → More potential for suboptimal solutions
- "Ockham's Razor"
  - Given a choice of two equally valid alternatives, take the simpler one

Ockham's Razor: Example

- How many ways are there to draw rectangles?

Ockham's Razor: Example

- How many ways are there to draw rectangles?
  - 1) Use java.awt.Rectangle
    - Advantages
      - Simple
      - Fast
      - Flexibility of shape API
    - Disadvantage
      - Possible loss of precision under transformation

Ockham's Razor: Example

- How many ways are there to draw rectangles?
  - 2) Use java.awt.geom.Rectangle2D
    - Advantages
      - Maintain floating point precision
      - Fast
      - Flexibility of Shape API

Ockham's Razor: Example

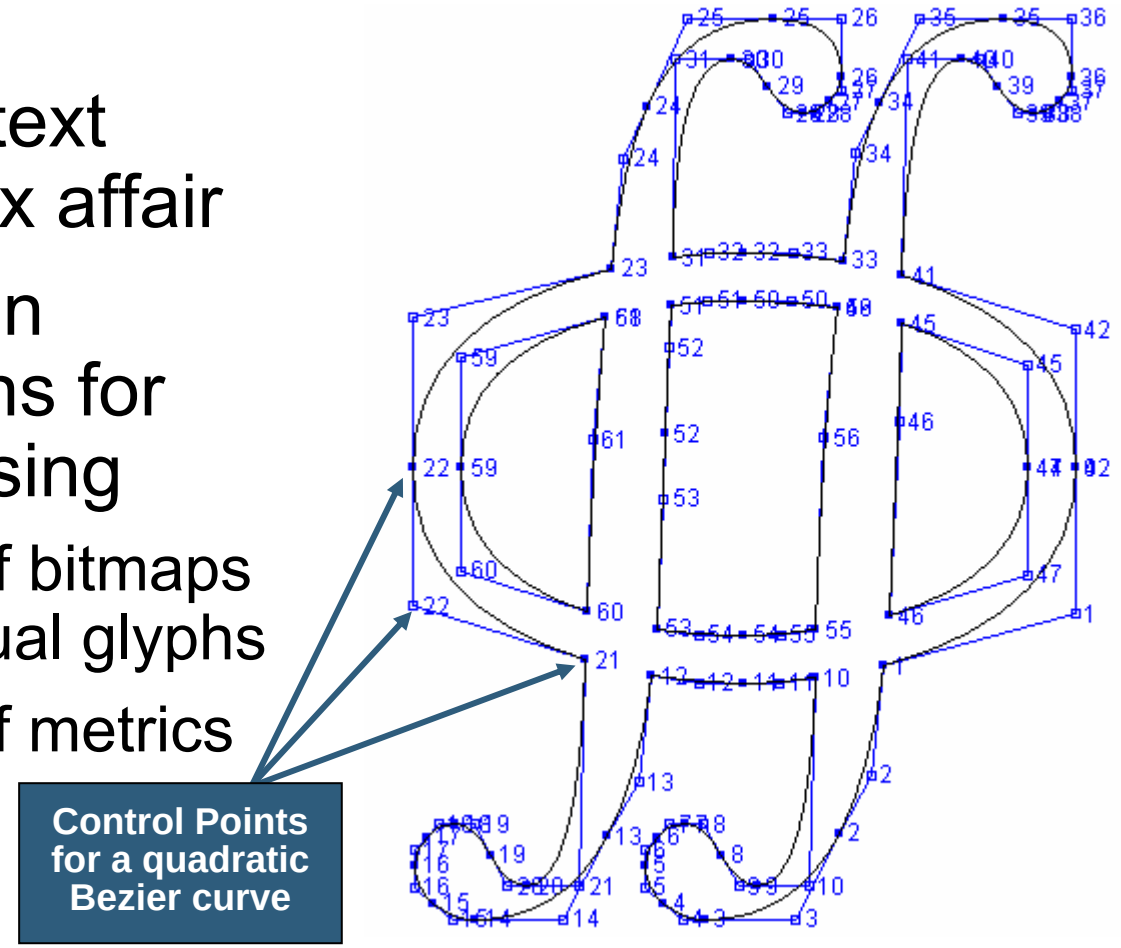
- How many ways are there to draw rectangles?
  - 3) Use java.awt.geom.GeneralPath
    - Advantage
      - Very general solution
    - Disadvantages
      - Computational complexity
      - Filling a general polynomial is much more expensive than filling a rectangle: often requires vertex sorting, winding rules…

Ockham's Razor: Example

- How many ways are there to draw a rectangle?
  - 4) Use java.awt.BasicStroke
    - WARNING: Kludge alert!!!
    - Bookkeeping to determine intersection with the clip boundary is more complex
    - Rendering algorithm is less efficient
    - Coordinates become awkward

Managing Graphics Complexity

- Rendering text is a complex affair
- Many built-in optimizations for text processing
  - Caching of bitmaps for individual glyphs
  - Caching of metrics



Control Points for a quadratic Bezier curve

Caching Tips

- Consider using cached images for repetitive non-trivial glyphs
- Consider caching size information to avoid unnecessary recalculation
  - General path iterates over the path to construct the bounds
  - Expensive if done repeatedly
- Cache = Managed Memory Leak
  - Avoid overuse
  - MRU cache
  - Weak reference

Tips for Repainting

- Consider one of the following approaches to facilitate tracking “dirty” shapes
  - Apply transform to shape rather than graphics context
  - Maintain rectangle for bookkeeping which is inverse transform applied to clip region
    - Quick mechanism for testing overlap with clip region
    - Bookkeeping in same coordinate space as shapes
- Painting a shape outside the clip region is not free
- **Don’t paint what you don’t have to!**

Computing the Inverse Transform

- Transform

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Inverse

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{\lambda} \begin{bmatrix} m_{2,2} & -m_{1,2} \\ -m_{2,1} & m_{1,1} \end{bmatrix} \begin{bmatrix} x' - m_{1,3} \\ y' - m_{2,3} \end{bmatrix}$$

$$\lambda = m_{1,1} \cdot m_{2,2} - m_{2,1} \cdot m_{1,2}$$

Rendering Hints

- Control over rendering behavior
  - Antialiasing on/off
  - Choice of interpolation algorithms
  - Render speed versus quality
- Tips
  - Consider turning off anti-aliasing for a moving object (e.g., scrolling or animation)

Agenda

Overview of the Graphics API

Graphics Pipeline

Graphics Primitives and Tools

Optimization

Demo

Summary

Optimization

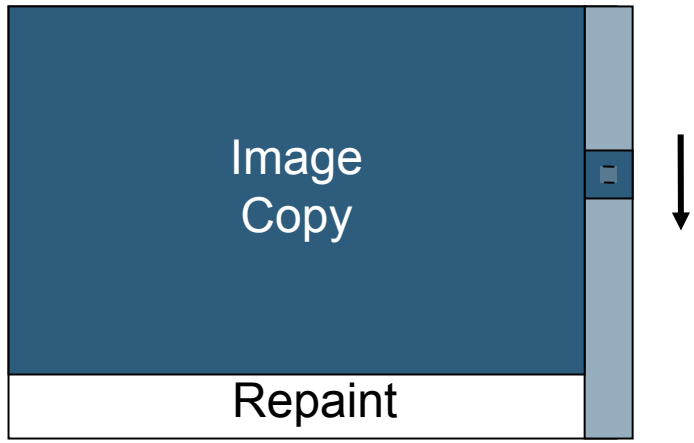
- Best strategy is application dependent
- Some are very easy to implement
- More detailed optimizations include
  - Intermediate Image
  - Spatial partitioning
  - Dynamic algorithms

Intermediate Image

- Speed versus memory tradeoff
- Often faster to render an image than a collection of objects

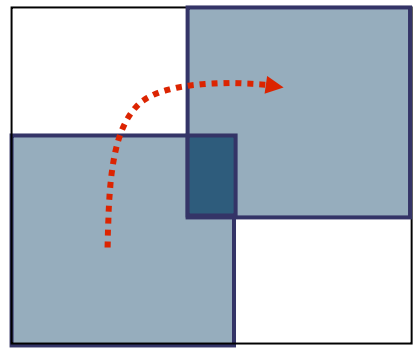
---

- Used by one of the scroll modes built into JViewport (BACKINGSTORE_SCROLL_MODE)
- Note: There is a more efficient mode based on copyArea (BLIT_SCROLL_MODE)



Using Intermediate Images

- Also useful in editing operations
- Create an intermediate image
  - Component without object being edited
- Component's paint renders image and overlays object



Dragging an object

Creating the Image

```
GraphicsConfiguration config =
    component.getGraphicsConfiguration();
image = config.createCompatibleImage( w, h );
```

Paint Implementation

```
public void paint( Graphics g) {
  if( image == null && useImage ) {
    image = createImage();
    Graphics imageg = image.createGraphics();
    drawContents( imageg );
    imageg.dispose();
  }
  if( image != null ) {
    g.drawImage( image, 0, 0, w, h,
      0, 0, w, h, Color.WHITE, null );
  } else {
    drawContents( g );
  }
  // additional painting not in drawContents
}
```

Bookkeeping

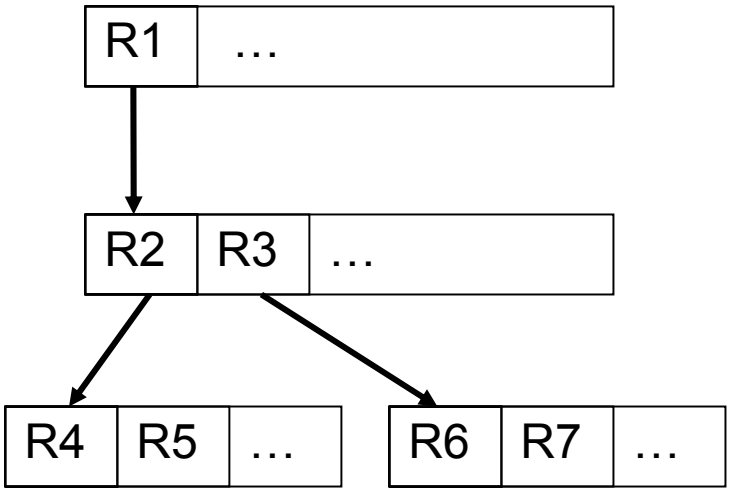
- Set image to null on a component resize to force recreation in next paint call
- For large documents, it may be necessary to discard offscreen images to save memory
- MRU and weak references may be used to help manage memory consumption

Spatial Partitioning

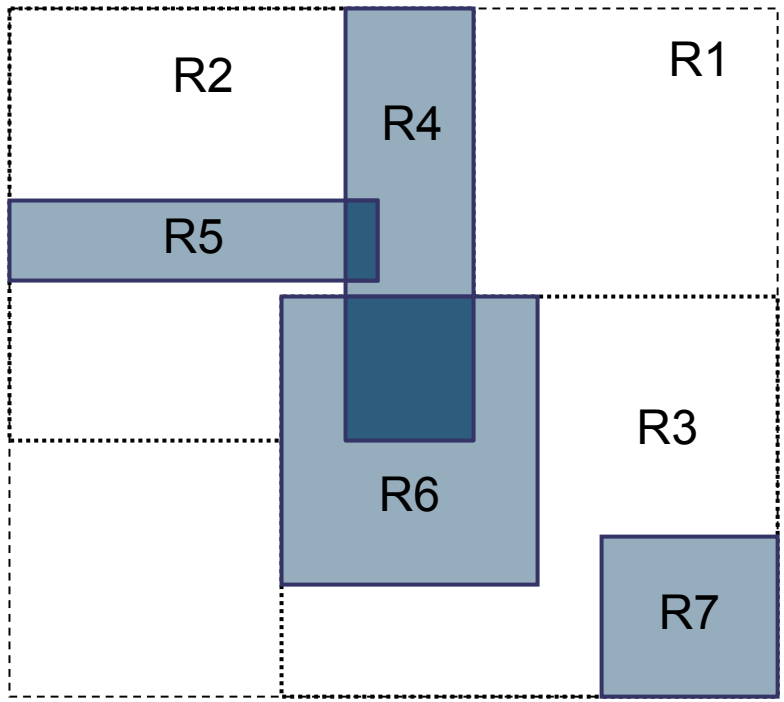
- Hierarchal organization of objects based on onscreen position
- Paint only what is necessary

```
Set N = root node
  if N intersects clip region
      if N is a branch node
          for each child in N
              recurse into child
      else
          paint leaf node
```

Spatial Partitioning



R*-tree decomposition



DEMO

Summary

- Solid performance is achievable for a graphics rich Java™ based application
- Optimization strategies built into the graphics API offer hints at how to solve related performance problems
- Useful strategies include
  - Intermediate Images
  - Spatial decomposition

For More Information

- A Scrolly, Clippy Swing Optimization
  - http://www.oreillynet.com/pub/wlg/9144?wlg=yes
- R*-Tree decompositions
  - http://www.sai.msu.su/~megera/postgres/gist/papers/Rstar3.pdf
- TS-3690 Handwriting recognition
- Demo
  - http://www.desktopjava.com

Q&A

High Performance GUIs

Kevin Ellis

GUI Manager
Maplesoft
www.maplesoft.com

TS-1305