Java2D Graphics Design Plan

Requirements

0. Support generic Java2D drawing in ordinary heavyweight NSWindows
1. Support embedding applet content in NPAPI browsers across processes
2. Support embedding other Quartz or OpenGL based drawing via JAWT

- - including sub-embedding in applet content
  3. Be highly performant

Remaining work

1. Connect the Java2D OpenGL rendering pipe to an IOSurface texture for each content view
2. Wire the CAOpenGLLayer to copy the entire IOSurface texture (with appropriate locking on the IOSurface)
3. Implement the JAWT API to connect provided CALayers as sub-layers of the top-level window's CALayer
4. Complete implementations of the "lightweight" AWT controls
5. Determine which surface primitive to expose for out-of-process rendering in a web browser plug-in: CALayers or IOSurfaces
    - Connect the NPAPI plug-in to the chosen mechanism

Completed work

- Render all Java2D drawing in OpenGL
- Bring up Cocoa-based event system
- Created a branch of macosx-port jdk workspace for development
- Prototyped CAOpenGLLayer drawing directly from Java2D OpenGL rendering pipe
    - ✅ Renders to window on screen
    - ⚠️ Bounds hardcoded
    - ⚠️ Requires further API to be exposed to an applet
    - ❌ Fails to draw entire scene, need intermediate buffer
- Prototyped IOSurface drawing into an NPAPI plug-in
    - ✅ Works in a browser
    - ⚠️ Does not connect to Java2D OpenGL rendering pipe
    - ❌ Cannot support sub-embedding (JAWT)
- Determined that both and IOSurface and a CALayer need to be used together to form a complete working system

Q&A

Q: Why use CoreAnimation layers?

A: CALayers can be connected to a hierarchical tree of arbitrary rendering surfaces across a variety of rendering technologies (not just OpenGL), which make them ideal for embedding content from disparate technologies (SWT, QuickTime, JOGL, LWJGL, etc). These trees can have branches that span across processes, but require new API to be introduced to JavaRuntimeSupport.framework to do so. CALayers can only be drawn to on the main (AppKit) thread.

Q: Why use IOSurfaces?

A: IOSurfaces can wrap OpenGL texture resources and share them between processes and the graphics card. An IOSurface is a good container for the pixels of the Java2D OpenGL pipeline, because there is no unusual threading requirements and the texture can hold the entire Java2D scene (unlike a CALayer which retains no state). IOSurfaces have no natural affordance for embedding, layering, or chaining sub-surfaces, so they are not an appropriate substrate for embedding.

Q: Why not use CoreGraphics drawing in applet plug-ins?

A: CoreGraphics draws to in-process memory. CoreGraphics has no natural affordance for either cross-process drawing nor embedded drawing. Using CoreGraphics (as opposed to CoreAnimation) drawing in an NPAPI plug-in is unlikely to achieve a simple nor performant solution.

Q: Can Quartz and CALayers be used to make a functional graphics system?

A: Yes, but CALayers require all drawing to occur on the main (AppKit) thread, and the drawing would still be going to a malloc()'d in-memory array of pixels. OpenGL commands directly to an IOSurface texture provide us a way to drive drawing directly to the card, and then flip the entire scene into a (potentially shared) CALayer on the main thread.

Q: What is the macosx-port currently using to render content onscreen?

A: Every window has an NSOpenGLContext which targeted by the Java2D OpenGL renderer, as well as an off-screen "scratch" context. This NSOpenGLContext is assigned directly to the root AWTView of the NSWindow, which basically connects the window's back-buffer pixels to the OpenGL context.

Q: Why use OpenGL? What happened to the Quartz and Sun2D renderers in Apple's Java SE 6?

A: The Quartz and Sun2D rendering pipelines in Java SE 6 are strictly in-memory only drawing routines which target the window back-buffer which is shared memory with the WindowServer. Since WindowServer windows are not easily shared (impossible using only API) and are in-memory only structures, they do not form the ideal substrate to build a performant, cross-process, and embeddable graphics system on.

Q: How does Apple's Plugin2 work with the Quartz/Sun2D drawing in Java SE 6 today, and also support embedding?

A: Since the Quartz and Sun2D rendering pipelines in Java SE 6 only support rendering to an NSWindow (among other obnoxious requirements of the NSView-based AWT heavyweights), applet content is rendering into an invisible NSWindow. After each update, a request is punted onto the main (AppKit) thread, which then creates a CGImageRef from the updated rectangle of the underlying window back-buffer pixels, and copies it into a CALayer. This CALayer is shared across processes back to the applet plug-in process, and is vended directly to the NPAPI via the CoreAnimation drawing model.