Tips for Fast 2D Image Processing in Java

Mark McKay
mark@kitfox.com
http://www.kitfox.com

Swing and AWT

- AWT is a heavyweight windowing API. Each control allocates it's own memory for rendering.
- Swing is lightweight. Every component is a JWindow shares the same video buffer for drawing.
- Swing is double buffered. When you repaint a component, the drawings are written to the back buffer of the root window. When the window has finished rendering, the back buffer is swapped to the front. This provides fast, flicker free drawing with a small memory footprint.
- Some APIs need to be heavyweight and have their own video memory. JOGL and Java3D Canvases are examples of such components.

Tips for Faster Rendering

- java.awt.RenderingHints allow the user to specify trade-offs between nice looking and fast rendering. Turning off anti-aliasing or choosing a less expensive interpolation method can speed things up.
- Keep it simple. Don't draw elements that will be completely obscured by other elements. Don't draw elements that will be offsreen. Don't use use curved shapes when rectangles will do.
- Use clip areas to shrink the area of a component that needs rendering.
- If using java.awt.geom.Area, convert to a GeneralPath before rendering or using as a clip area.
- javax.swing.CellRendererPane can let you have Swing do most of the work for you. This is a great way to draw JComponents anywhere onscreen during the paintComponent() call. Useful for designing your own custom controls that display formatted data. This is the trick JTables and JLists use to draw their content.

Animation

- Animation is not much different from drawing regularly.
- Each animated element should be described by a set of parameters. These parameters can be updated over time. When an element's parameters change, it's appearance will change too.
- Cache as much as possible. It's much faster to draw something to an offscreen buffer once and then blit it back onscreen when needed than it is to redraw it from scratch each time.
- Use lazy evaluation. When parameters change, set a flag saying that the element needs to update it's appearance – but don't actually update the appearance until it next needs to be drawn.

Fast Images

- Java has many ways to create and manipulate images.
- Image ops are fastest when the source and destination images have the same data format.
- Use GraphicsConfiguration.createCompatableImage() to create the image that best matches the Component you are drawing to.
- Java accelerates some image data processing with native code, but not others. If using BufferedImages, use BufferedImage.TYPE_INT_ARGB or BufferedImage.TYPE_4BYTE_ARGB to take advantage of this.
- BufferedImage.TYPE_INT_ARGB seems to be fast in practice.

Volatile Images

- BufferedImages are handled in software. VolatileImages exist in hardware memory.
- As of Java 1.4, BufferedImages are automatically 'accelerated' – ie, the runtime engine automatically converts them to Volatile images when it detects this will provide a speed advantage.
- VolatileImages are faster because they are stored in video hardware - but their content can be lost at any time, so they need to be carefully monitored and updated.
- Blits between images with incompatible data types have to be done in software. As a result, accelerated BufferedImages become decelerated, which causes a slowdown. To ensure images stay volatile, be careful to only perform blits with compatible images.

Writable Rasters

- For intensive pixel operations, it might be best to use a WritableRaster.
- A raster encapsulates a DataBuffer of pixel data, and a SampleModel that describes how the data buffer is partitioned into individual pixels.
- The DataBuffer can be quickly accessed and manipulated. Also, because it can be treated as numerical data, you can apply unusual operations to it, such as Fourier analysis.
- A WritableRaster can be combined with a ColorModel to produce a BufferedImage that can be used in standard graphics routines.
- It is a good idea to choose the default ColorModel to ensure JDK native code controls things behind the scenes.
- GradientPaint and TexturePaint use WritableRasters to render their content.

Procedural Shaders

- Combine OpenGL with Java2D/Swing
- Use JOGL pbuffers to create 3D images and render it to an offscreen buffer. Then blit the buffer to screen.
- OpenGL shaders provide dynamic, accelerated image filtering ops: Gaussian blurs, brightness/contrast controls, embossing, bump map shading – pretty much any 2D filtering operation.

JAI

- An extensive library for fancy, fast image compositing and manipulation.
- Much of it is accelerated in native code. Java interpreted code is used when native code for your platform doesn't exist.
- Allows use of huge images by partitioning large images into tiles.