

Problems with Java and Fonts

Ben Galbraith (ben@galbraiths.org), August 2004

Introduction

This paper is concerned with two categories of problems with Java fonts:

1. Those that inhibit accurate emulation of Windows' font rendering (especially Windows XP)
2. Those that impair the creation of "beautiful", cross-platform rich client applications

First, a series of objective observations will be made that identify problems across these categories. Next, these observations will be used as the basis for various conclusions, some objective and some at least partially subjective. Then, a set of entirely subjective recommendations based on the conclusions will be presented. Finally, details will follow that include a test program, screen captures, and information about the testing platforms.

Observations

1. Java2D and Windows XP use a different basis for font point sizes.
 - a. Java2D “point” appears to be approximately $\frac{3}{4}$ of a Windows “point.”
2. Swing makes it too difficult to use anti-aliased fonts.
 - a. Windows automatically anti-aliases 14 point fonts and higher (roughly equivalent to a 19 point Java2D font)
 - b. Swing does not expose any mechanism for emulating this behavior in any standard `LookAndFeel` implementation
 - i. Developers must either install custom UI delegates or create/subclass Swing components
3. Java2D font renderer has occasional glitches (and/or inconsistencies with Windows’ font renderer)
 - a. Random examples of garbled glyphs:
 - i. Arial 13pt (compare to Windows’ 10pt rendering): x, S, X
 - ii. Microsoft Sans Serif 11pt (Windows’ 8pt): x, C, M, Q, 8, 9,)
 - iii. Times New Roman 16pt (Windows’ 12pt): 8
 - iv. Tahoma 12pt (Windows’ 9 pt): D
 - v. Tahoma 13pt (Windows’ 10 pt): q
 - vi. Tahoma 16pt (Windows’ 12 pt): a, 0
4. Java2D font anti-aliasing generally does a good job as compared to Windows XP anti-aliasing
 - a. Java2D renders slightly darker than Windows XP at equivalent font sizes
 - i. E.g., Times New Roman 27pt vs. Windows 20pt
5. Java2D lacks subpixel rendering
 - a. Java2D font rendering with or without anti-aliasing is noticeably inferior to Windows font rendering with subpixel rendering (“ClearType”)
6. Java2D font rendering is inferior to OS X font rendering
 - a. Inferiority especially apparent at small font sizes

7. Java2D does not provide for easy manipulation of character spacing
 - a. However, users can roll their own abstraction built on lower-level elements of Java2D API to directly manipulate character spacing

Conclusions

1. Windows font emulation is at times difficult and often impossible

- a. On Windows systems not using subpixel rendering, Java2D is good at emulating fonts drawn at sizes less than 14 Windows points.
 - i. However, occasional Java2D rendering glitches can degrade an application's appearance.
 - ii. At Windows point sizes of 14 or higher, Java2D does a fair job of emulating Windows when anti-aliasing is turned on. However, Swing makes it very difficult to accomplish this behavior.
- b. On Windows XP systems using subpixel rendering, Java2D fails to emulate Windows fonts effectively. Java applications as a result look noticeably inferior.
 - i. This includes any laptop or LCD-equipped desktop computer sold in the past few years.

2. Creating cross-platform Java applications with "gorgeous" font rendering is difficult

- a. Macintosh OS X's superb font rendering has raised expectations for "beautiful" applications
 - i. Longhorn's font rendering, currently slated to be superior to OS X, will likely spur improvements in OS X and thus further raise expectations by the end of the decade.
- b. "Beautiful" font rendering is a product of both high-quality glyph rendering and "proper" character spacing.
 - i. Java2D's font rendering with anti-aliasing, while good, is often inferior to OS X's current implementation and appears to be inferior to Longhorn's announced implementation (and existing Longhorn demos). To be fair, at times Java2D's font rendering is nearly as good as OS X.
 - ii. Java2D has no algorithm or heuristic for adjusting character spacing for maximum quality. Manually adjusting character spacing is possible but difficult. (Note that achieving optimum character spacing is more than applying kerning tables.)

Recommendations

1. **Enhance Swing to allow for easy use of font anti-aliasing in components.** It should also be easy to establish a bottom threshold below which anti-aliasing is turned off by default. It is needlessly tedious for developers to incorporate anti-aliasing in Java applications with default `LookAndFeel` implementations; “corporate developers” may find it very difficult.
2. **Fix existing font rendering glitches.**
3. **Implement subpixel rendering or allow for integration with Windows font renderer.** Java2D-based applications on Windows XP will continue to stick out like a sore thumb until this is fixed. Java does a good enough job of font emulation for printing; could an optional integration with the native font renderer for screen display whilst using Java2D font rendering for printing be a good solution?

If Java emulates subpixel rendering, a facility for determining the state of the native operating systems’ subpixel rendering (i.e., on, or off) should be provided, and Swing should be enhanced to “just do the right thing.”

4. **Increase quality of font rendering.** Cross-platform Java applications can’t match the beauty of Windows’ subpixel rendering and OS X’s anti-aliasing¹ today. As expectations for GUI quality increase through the decade, Java will fall further and further behind unless its font rendering is upgraded.

This upgrade should probably consist of higher quality glyph rendering with anti-aliasing, a subpixel rendering mechanism, application of font kerning tables, and a user-configurable heuristic/algorithm for adjusting character spacing for best appearance.

¹ Java applications on OS X don’t use Java2D for font rendering; Apple’s JRE uses Quartz (OS X 2D drawing API) just as older Sun JRE’s used the native platform.

Details

All observations and screen captures were made with Sun's JDK 1.4.2_03 running on Windows XP SP2 and Apple's "Java 1.4.2 Update 1" (based on Sun JDK 1.4.2_05) running on OS X 10.3.5.

An SWT/Swing application, **Font Viewer**, was created to analyze the differences between Windows XP and Java2D font rendering. It is available at <http://galbraiths.org/fontviewer.zip>. It requires the included Windows DLLs to be in the operating system's PATH (or in JRE/bin or a directory specified by the `java.library.path` system property). The ZIP file only includes SWT native libraries for Windows; Font Viewer can be run on other platforms by obtaining the SWT native libraries for those platforms from eclipse.org.

Minor note: Font Viewer generally disposes of all SWT light-weight peers as required; however, it does have a very minor Windows GDI memory leak (a few Font and Color instances whose disposal would have required more effort than its author was willing to make).

The following screen captures are included to illustrate several of the items listed in the **Observations** section of this paper. Screen captures are numbered with the scheme [observation number]-[capture number].

the quick, brown fox

Screen Capture 1-1: Windows XP rendering of Arial at 20 points

the quick, brown fox

Screen Capture 1-2: Java2D rendering of Arial at 27 points

Windows XP - 10 points

the quick, brown fox; jumps over. the lazy(?) dog! 0123456789
THE QUICK, BROWN FOX; JUMPS OVER. THE LAZY(?) DOG!

Java2D - 13 points

the quick, brown fox; jumps over. the lazy(?) dog! 0123456789
THE QUICK, BROWN FOX; JUMPS OVER. THE LAZY(?) DOG!

Screen Capture 3-1: Misrendered Arial glyphs (shown at 2x normal size)

Windows XP - 8 points

the quick, brown fox; jumps over. the lazy(?) dog! 0123456789
THE QUICK, BROWN FOX; JUMPS OVER. THE LAZY(?) DOG! 0123456789

Java2D - 11 points

the quick, brown fox; jumps over. the lazy(?) dog! 0123456789
THE QUICK, BROWN FOX; JUMPS OVER. THE LAZY(?) DOG! 0123456789

Screen Capture 3-2: Misrendered Microsoft Sans Serif glyphs
(shown at 2x normal size)

the quick, brown fox; jumps over. the lazy(?) dog! 0123456789
THE QUICK, BROWN FOX; JUMPS OVER. THE LAZY(?) DOG! 0123456789

Screen Capture 4-1: Windows XP rendering of Times New Roman at 20 points

the quick, brown fox; jumps over. the lazy(?) dog! 0123456789
THE QUICK, BROWN FOX; JUMPS OVER. THE LAZY(?) DOG! 0123456789

Screen Capture 4-2: Java2D anti-aliased rendering of Times New Roman at 27 points;
note that it appears slightly darker than the Windows equivalent.

The quick brown fox jumps over the lazy dog.
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

Screen Capture 5-1: Windows XP ClearType (subpixel) rendering of Courier New 8pt.
ClearType requires an LCD for proper display; CRT monitors do not have adequate precision to achieve the desired effect
(shown at 2x normal size)

The quick brown fox jumps over the lazy dog.
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

Screen Capture 5-2: Java2D anti-aliased rendering of Courier New 11pt.
Note how the Windows XP ClearType rendering has much more consistent and readable glyphs (shown at 2x normal size)

The quick brown fox jumps over the lazy dog.
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

Screen Capture 5-3: Windows XP ClearType (subpixel) rendering of Microsoft Sans Serif at 8pt (shown at 2x normal size).

The quick brown fox jumps over the lazy dog.
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

Screen Capture 5-4: Java2D anti-aliased rendering of Microsoft Sans Serif at 8pt. Note that using anti-aliasing to compensate for subpixel rendering, as shown here, yields poor results, such as the malformed M, O, R, and Q characters (shown at 2x normal size).

The quick brown fox jumps over the lazy dog.

Screen Capture 6-1: OS X font rendering of Arial 12pt (via Java2D)

The quick brown fox jumps over the lazy dog.

Screen Capture 6-2: Java2D font rendering of Arial 12pt

The quick brown fox
jumps over the lazy dog.

Screen Capture 6-3: OS X font rendering of Goudy Old Style 43pt

The quick brown fox
jumps over the lazy dog.

Screen Capture 6-4: Java2D font rendering of Goudy Old Style 43pt.
Note that OS X version has better character spacing,
baseline alignment, and noticeably better glyph rendering