Is 2-D graphics the next killer app for Linux?

Raph Levien
artofcode LLC

Abstract

This paper describes some of the technical issues and challenges involved in high quality 2D graphics displays. We further describe a number of projects that provide infrastructure for 2D graphics applications under Linux and other free operating systems.

1 Introduction

The compelling technical strengths of Linux and other free software systems in multitasking and networking have brought it considerable success in the area of Web servers. In this presentation, I demonstrate that the similar technical strengths free software is gaining in 2D graphics, and argue that this area could be the next "killer app" for Linux.

Advances in display technology present an opportunity and a challenge for 2D graphics software. As resolutions pass 140 dpi for CRT's and 200 dpi for LCD's, the user experience becomes a sharp, clear display of information rather than a field of pixels, with a dilemma between smooth but fuzzy antialiasing, or sharp but jaggy edges. These displays pose a formidable challenge to drive well. First, software must be designed to be scalable, or resolution independent. Second, at these resolutions, good antialiasing is crucial for the highest quality. Third, these displays require generating pixels at a very high bandwidth.

Fortunately, between existing technology and new projects in development, Linux will meet these challenges. We will present and demonstrate four particularly instructive technologies:

- Ghostscript is a mature program, long having provided scalable, resolution independent rendering for both screen and printer. We will demonstrate the new transparency and blending capabilities of PDF 1.4, scheduled for the next major release.
- Nautilus, the new Gnome file manager from Eazel, has the core technology in place for scalable display, including vector graphics icons and scalable antialiased text. Nautilus uses the Libart rendering engine and the Gnome Canvas.
- The XFree86 Render extension, currently in development, will provide hardware acceleration for 2D graphics primitives including antialiased vector path and text rendering, alpha compositing, and semitransparent images.
- Keith Packard has developed an experimental X server that can alpha composite between windows. User interfaces built on top of this technology have the potential to rival the most advanced proprietary systems, including Mac OS X's Aqua user interface.

The core technology is coming. The challenge now is to make sure that they're integrated throughout all applications. This presentation will help show the way for users, developers of applications, and those with an interest in where the technology is going.

2 The evolution of resolution

The resolution of computer displays has been experiencing steady improvement for quite some time. While a thorough examination of historical data is beyond the scope of this paper, we examine the trends and underlying technology, and offer some predictions. We also take a look at the theoretically useful limits of resolution in terms of its ability to be perceived by the human visual system.

First, what is resolution? The fundamental parameters are pixels per inch and color depth. In pure information-theoretic terms, the maximum information carrying capacity of a square inch of display is the square of the resolution in pixels per inch multiplied by the color depth in bits.

These figures don't tell the whole story, however. Display devices don't display these bits of information perfectly and purely; they almost always add some form of degradation to the image. The form and amount of degradation varies with the type and of the device. The primary form of degradation in CRT's tends to be blurring, primarily caused by the aperture grill or shadow mask used to make the device capable of color. In inkjet printers, the degradation is primarily inaccurate positioning of ink drops, combined with ink spreading on the page. In laser printers, it's toner spreading and noise caused by the xerographic process. Good quality LCD's offer quite little degredation, although distortion of the tone range is common, especially for off-axis viewing.

Nonetheless, most displays in widespread use today operate fairly close to the theoretical limits imposed by their resolution and color depth.

The introduction of the Macintosh in 1983 set a standard for displays in consumer computers: 75 pixels per inch, albeit with only a single bit per pixel of color depth.

As of the mid-90's, the resolution on PC's was not considerably greater, typically 96 pixels per inch, although color depth had increased to 16 or 24 bits truecolor.

Today, most people still run their displays at around 96 pixels per inch, largely due to software limitations. However, higher resolution consumer devices are available. The DELL Inspiron 8000 laptop, for example, is available with a 1400 x 1050 pixel 14.1" LCD display, for a resolution of 124 dpi. Similarly, higher-end monitors such as the ViewSonic PF815 typically have maxiumum resolutions in the 120 dpi range. The Nokia 446PRO has a maximum resolution of 133 dpi.

200 dpi LCD screens have been fabricated as research prototypes (see http://www.research.ibm.com/roentgen/) and have just begun low-volume production [IBM00]. It is reasonable to expect that displays of this resolution will be available commercially soon.

Display resolutions beyond 200 dpi are feasible, but won't provide much improvement in actual display quality. This assertion may seem surprising to those who note the dramatic difference in quality between 300 and 600 dpi laser printers, for example. However, laser printers have one bit per pixel color depth. As such, they produce jaggy edges and strokes quantized to integer pixel widths.

The human visual system has no hard resolution cutoff. Instead, contrast sensitivity decreases as spatial frequencies increase. The peak sensitivity is approximately 3 cycles per degree (corresponding to roughly 18 dpi at an 18 inch viewing distance), is roughly 10% of the peak sensitivity at 10 cycles per degree (corresponding to 60 dpi), and trails off to 1% of peak at 30 cycles per degree (corresponding to 190 dpi).

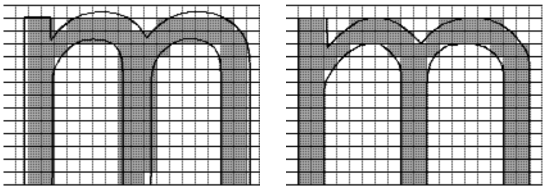
Of course, individuals vary in visual acuity, and some may also prefer viewing distances closer than 18 inches. However, it is reasonable to conclude that a high quality 200 dpi display with good color depth represents a "magic point" in display technology, beyond which the gains are marginal.

3 Hinting and antialiasing

Hinting is a technique for improving the quality of text rendering [Mic98]. It generally works by gently distorting the shape so that it is aligned to the pixel grid before rendering. Hinting improves rendering in a number of ways, including:

- Making stroke widths uniform.
- Making characters symmetrical.
- Avoiding flat or "pimply" curve extrema.

Thus for one bit deep displays, hinting is almost always a win. A large number of technologies for font hinting exist, including Metafont [Knu86], Adobe's Type 1 format, and TrueType. All of these techniques embed "hints" in the font to direct the renderer how to distort the font shape (hence the name). Some renderers work on the basis of "autohinting" and do not require hints embedded in the font. These include John Hobby's groundbreaking research on automatic glyph rendering [Hob93], as well as the new autohinting module in FreeType.



Unhinted vs. hinted glyph rasterization. From [Hob93].

Antialiasing is a technique for replacing jagged edges with a grayscale representation. As such, it is only applicable to displays with more than one bit of color depth. In a non-antialiased display of a shape, each pixel is colored according to whether or not its center is inside the shape. Thus, a diagonal edge is rendered with a jagged edge as the edge steps from one row of pixels to the next. In an antialiased display, the pixel is colored according to the *fraction* of the pixel inside the shape. If the entire pixel is inside, it is colored black. If a portion of the pixel is inside, it is colored gray. Thus, diagonal edges are rendered with soft ramps of gray rather than jaggies.

On multibit displays, antialiasing is a tradeoff. It gets rid of the "jaggies" and improves the accuracy of rendering overall, but at a price: softness or fuzziness of the edges, and also lower contrast.

One technique used to mitigate these effects is to combine antialiasing and hinting. The most important effect of this technique is to align vertical and horizontal strokes to the pixel grid, so that the edges of these strokes have full contrast. However, diagonal and curved edges still suffer from softness.

The biggest drawback to hinting is that it distorts the letter shapes. This prevents the display from being smoothly scalable, and also detracts from the accuracy of the displayed text, which is particularly annoying in desktop publishing applications. This problem acutely effects the spacing between letters, as hinting quantizes widths and offsets to integer coordinates. Yet, at today's resolutions, the improvement in edge sharpness and contrast makes hinting in conjunction with antialiasing an appealing tradeoff.

As resolution increases, however, the tradeoff tilts in favor of *not* performing hinting. For glyphs rendered to sizes of 20 pixels, stroke width for normal fonts increases to two pixels. Strokes two pixels or wider always have a fully-on pixel in their interior, while for thinner strokes this is not guaranteed unless the stroke edges are aligned to pixel boundaries. Thus, contrast of unhinted antialiased strokes is fairly poor at low resolutions, steadily improves as resolution increases, and then acheives full contrast at approximately 140 dpi and higher. Thus, at high resolutions, the distortion caused by hinting letterforms is not offset by any improvement in contrast.

I conclude, then, that when high resolution displays become commonplace, the battles today over hinting technology will become as irrelevant as once-fierce battles over 8-bit pseudocolor dithering are today.

3.1 Software support for scalability

I found this quote ad for Apple's Cinema Display amusing:

> **Easy on your eyes**
> Pixel density is something else to watch out for. After all, you don't want to have to squint to see images because the pixel density at high resolution makes them too small to see without magnification. With the Apple Cinema Display, the pixel density allows you to use the display at the maximum resolution all the time - and still view everything at a size and sharpness that's easy on your eyes.

If I'm reading it correctly, it's praising the *low* resolution of the display (86 dpi) as being easy on the ideas. This is counterintuitive: all other things being equal, a higher resolution should *always* be a higher quality display in all respects.

The problem here, of course, is that not all other things are equal. In particular, most software written today is not designed for scalable displays. With unscalable software, elements of the display are forced to become smaller as the resolution increases—a state of affairs that definitely can lead to squinting.

Designing software for scalable displays is a considerable challenge. The infrastructure technology has been available for some time (including Display PostScript, Display PDF, Libart, etc.), but most applications are written in terms of hardwired pixels. This is, surprisingly, even true for UI applications written for the supposedly "next-generation" Aqua UI platform in Mac OS X, even though the underlying Display PDF technology is quite scalable[App00].

In the world of the Linux desktop, things are slightly better. For one, the configurability of apps (while creating a usability mess beyond the scope of this paper) generally means you can use larger fonts and graphic elements. The Gnome Canvas goes one step further by providing an adjustable zoom parameter, capable of scaling the entire display by a constant scale factor. Applications using the Gnome Canvas for display thus gain the ability to scale quite easily.

One such application, Nautilus, is particularly appealing because it exposes the zoom control to the user. It also provides more detail as the zoom factor increases. As such, Nautilus will probably yield excellent results when used on higher resolution displays.

In many ways, the need for scalable displays parallels the "Y2K" problem. Just as 20th century programmers or should have known that the year 2000 would arrive, GUI programmers today know (or should know) that high resolution displays are coming. To avoid lots of expensive retrofitting when this happens, simply design for it now.

Unfortunately, I expect that the unscalability of today's popular GUI software will hold back the commercial market for high resolution displays. Most likely, specialized applications, such as supercomputer vizualization, medical display devices, and map readers, will drive the market for these devices in the short term. Many of these applications will be running under Linux and will be able to take full advantage of the 2D rendering infrastructure available.

4 2D Graphics Infrastructure

Fortunately, free software provides quite a bit of infrastructure for 2D graphics, with development continuing rapidly.

This paper will discuss in more detail three projects which I feel are especially valuable resources to 2D graphics applications: Ghostscript, Libart, and the XFree86 Render extension. No slight is intended to other projects, some of which may be quite promising.

4.1 Ghostscript

Ghostscript's development dates back to 1986, as a project of L. Peter Deutsch. Version 1.0 was released in August of 1988. Since then, it has become a fixture in Unix systems. I took over the maintainership of Ghostscript in August 2000.

Ghostscript tracks the development of Adobe's PostScript and PDF standards quite closely. In particular, it has tracked the development of the PostScript/PDF imaging model. Up to PostScript LangageLevel 3, this imaging model was based on the "cut and stencil" model, in which an object is either fully transparent or fully opaque at each pixel. Even so, the base imaging model is quite powerful for most printing applications, including Bezier paths, a wide range of stroking parameters, and both Type 1 and TrueType fonts (as well as the myriad variants of each of these font formats). It also supports RGB and CMYK color spaces (Adobe's DeviceN and Separation color spaces for "spot colors" are not yet implemented in Ghostscript). The LanguageLevel 3 spec added sophisticated gradients, based on Gouraud triangle meshes, Coons patches, or Tensor product splines, in addition to the more usual linear and radial gradients.

However, for many graphical applications, the underlying "cut and stencil" is simply too limiting. Thus, most 2D graphics efforts have extended the basic PostScript imaging model to include compositing of semi-transparent objects. With this extension, drop shadow and "glow" effects are quite straightforward. Compositing RGBA images (the usual red, green, blue plus an added "alpha" channel for controlling transparency) renders soft (or "feathered") edges realistically, and also allows fine control over antialiasing.

With the imminent introduction of the PDF 1.4 standard, the PDF imaging model surpasses any other 2D graphics system in wide use. At the time of this writing, the full PDF 1.4 specification is not yet published, but Adobe has released a technical note describing the transparency and blending operations [Ado00].

4.2 Libart

Libart is a 2D graphics library optimized for interactive displays. It supports most of the PostScript imaging model, and has full support for antialiasing and transparency. Libart is the engine for the antialiased renderer in the Gnome Canvas [MqL00].

One special Libart feature is microtile arrays, a lightweight and efficient approximate representation of specific regions. These microtile arrays are particularly useful for minimizing the re-display area on incremental display, speeding up response time and making motion smoother.

Libart is basically designed to the SVG imaging model. It is used as the basis for librsvg, the batch SVG renderer embedded in Nautilus. At present, it lacks the advanced gradients of PostScript LanguageLevel 3, as well as the advanced blending options of PDF 1.4. In addition, it is optimized for RGB displays and does not support CMYK or other color spaces needed for printing.

The future plan is for Libart and the Ghostscript graphics library to merge. This merged library will hopefully provide the smooth antialiasing features and ease of integration of the current Libart, along with a complete implementation of the PDF 1.4 imaging model.

4.3 XFree86 Render

The XFree86 Render extension is one of the more exciting developments in the land of free 2D graphics in a while. The basic idea is simple: provide access to the incredibly high speed of modern video hardware, through an interface consistent with the X11 base.

The primitives offered by Render are more or less exactly those needed to do sophisticated 2D graphics: RGBA images, antialiased glyph compositing, antialised shapes (rendered as lots of small triangles), and gradients (rendered as lots of Gouraud-shaded triangles).

The speed of hardware acceleration for these primitives is impressive. Keith Packard reports at least one order of magnitude, and in many cases two. This opens up many new possibilities. For example, real-time scrolling has become expected. Carefully implemented, real-time zooming may become realistic. For some applications, such as map display, it could be a very useful feature.

The greatest challenge posed by the Render extension is integration with applications. This work has already begun - Keith Packard has posted patches to KDE that make it display antialiased text. There is a considerable amount of work remaining, but I am confident that it will proceed apace, fueled by the astonishing performance improvements offered by hardware.

5 2D Graphics Applications

The Gimp, an image manipualation program, is one of the highest profile graphics applications available. It is one of the first free software applications with a modern user interface. Its success is partly responsible for inspiring the Gnome desktop project.

While there are other image manipulation programs (notably KImageShop), the Gimp is the tool of the choice for those seriosly doing graphics. Unfortunately, the situation is markedly different in other fields.

There are over a dozen projects to create a usable vector graphics editor. None of them are anywhere near their commercial counterparts, such as Adobe Illustrator, Corel Draw, or Macromedia FreeHand. The list of free vector apps includes: xfig, tgif, Gill, Sodipodi, GYVE, KIllustrator, Sketch, ivtools, ImPress, geist, GILT, and possibly others.

I believe a big part of the problem is the lack of a good standard data format for vector graphics. While images are widely interchangeable in standard formats, each vector illustration program, proprietary or free, has tended to have its own format. I had great hopes that SVG would become such a standard, but based on my own experiences trying to implement Gill, it is likely its complexity and dependence on other bleeding edge technologies will hamper its widespread adoption.

6 Conclusion

Expectations for the 2D imaging model have been steadily rising. Where the PostScript “cut and stencil” imaging model was once considered sufficient, antialiasing and alpha transparency are requirements for any modern 2D graphics application. High resolution displays and accelerated video hardware will bring advanced 2D graphics capabilities to ordinary users over the next few years. New applications can and should be written to be scalable, and to make good use of the free software infrastructure available.

References

[Ado00] Adobe, Transparency in PDF. Technical Note #5407, May 2000.

[App00] Apple Computer, Inside Mac OS X, Adopting the Aqua Interface, 2000.

[Hob93] Hobby, J. D. Generating Automatically Tuned Bitmaps from Outlines, JACM 40(1), 1993.

[IBM00] IBM ships world's highest-resolution computer display, November 2000. http://www.ibm.com/news/2000/11/10.phtml

[Kan99] Kang, H. Digital Color Halftoning. SPIE and IEEE Press, 1999.

[Knu86] Knuth, D. E. The Metafont Book. Addison-Wesley, 1986.

[Mic98] Microsoft Corp. Introduction to hinting. http://www.microsoft.com/typography/hinting/hinting.htm

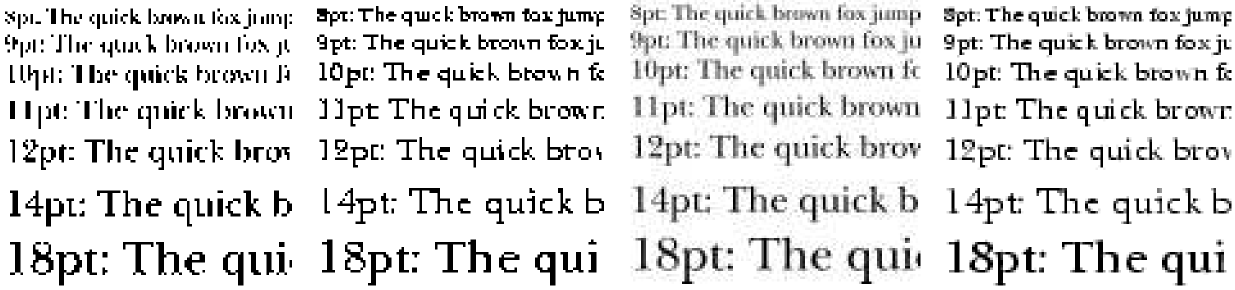
[MqL00] Mena-Quintero, F. and R. Levien, The GNOME Canvas: a Generic Engine for Structured Graphics, Usenix Technical Conference, San Diego June 2000.

[Nai91] Naiman, A. C., The Use of Grayscale for Improved Character Presentation. PhD thesis, Technical Report CSRI-253, U. Toronto, March 1991.

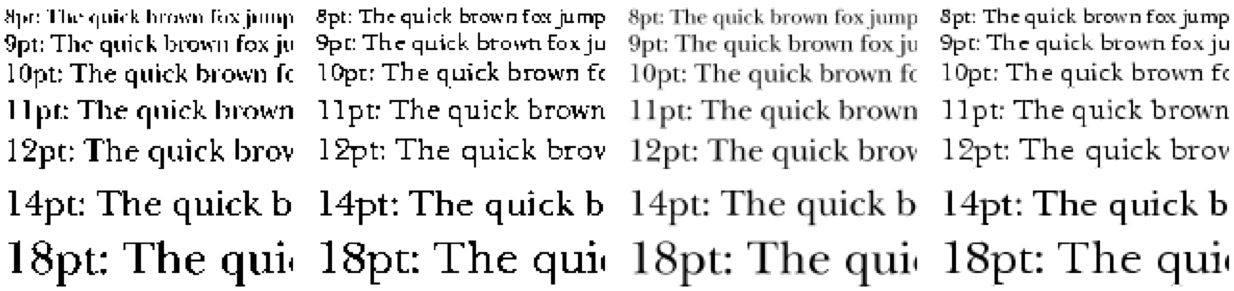
[Pac00] Packard, K. A New Rendering Model for X. Usenix Technical Conference 2000.

Appendix A

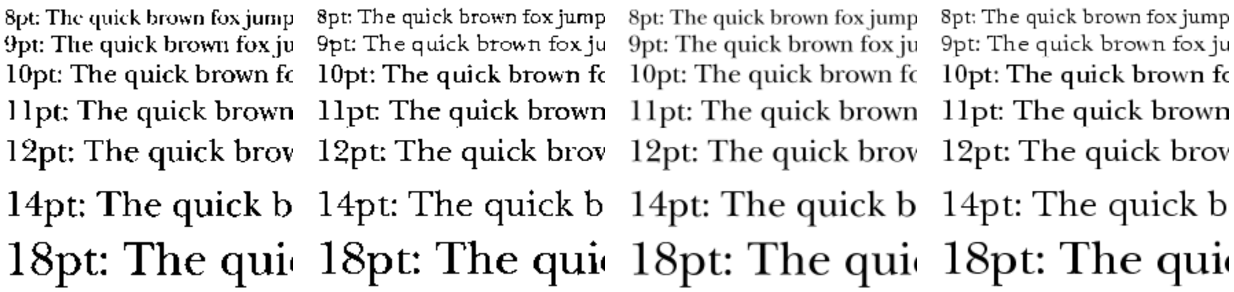
This appendix shows samples of text rendered at different resolutions, and with different options for hinting and antialiasing. All samples were rendered with Ghostscript 6.50.



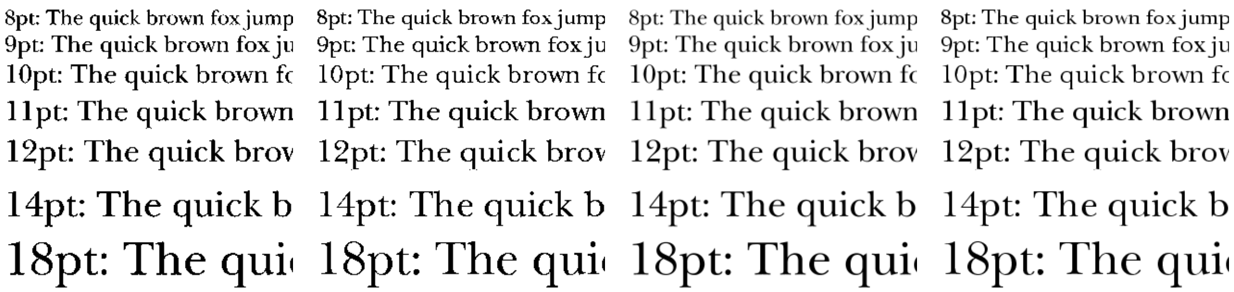
75 dpi: no antialiasing or hinting, hinting only, antialiasing only, hinting and antialiasing.



100 dpi: no antialiasing or hinting, hinting only, antialiasing only, hinting and antialiasing.



140 dpi: no antialiasing or hinting, hinting only, antialiasing only, hinting and antialiasing.



200 dpi: no antialiasing or hinting, hinting only, antialiasing only, hinting and antialiasing.