The (Re)Architecture of the X Window System

Keith Packard and Jim Gettys
keithp@hp.com, jim.gettys@hp.com
Cambridge Research Laboratory, HP Labs

Abstract

The X Window System, Version 11, is the standard window system on Linux and UNIX systems. X11, designed in 1987, was "state of the art" at that time. From its inception, X has been a network transparent window system in which X client applications can run on any machine in a network using an X server running on any display. While there have been some significant extensions to X over its history (e.g. OpenGL support), X's design lay fallow over much of the 1990's. With the increasing interest in open source systems, it was no longer sufficient for modern applications and a significant overhaul is now well underway. This paper describes revisions to the architecture of the window system used in a growing fraction of desktops and embedded systems.

While part of this work is "good citizenship" required by open source, some of the architectural problems solved ease the ability of open source applications to print their results, and some of the techniques developed are believed to be in advance of the commercial computer industry.

The challenges being faced include:

- X's fundamentally flawed font architecture made it difficult to implement good WYSIWYG systems
- Inadequate 2D graphics, which had always been intended to be augmented and/or replaced
- Developers are loathe to adopt any new technology that limits the distribution of their applications
- Legal requirements for accessibility for screen magnifiers are difficult to implement
- Modern user interface eye candy, which sport translucent graphics, windows, drop shadows, etc.
- Full integration of applications into 3 D environments
- Collaborative shared use of X (e.g. multiple simultaneous use of projector walls or other shared applications)

While some of this work has been published elsewhere, there has never been any overview paper describing this work as an integrated whole, and the compositing manager work described below is novel as of fall 2003. This work represents a long term effort that started in 1999, and will continue for several years more.

Text and Graphics

X's obsolete 2D bit-blit based text and graphics system problems were most urgent. The development of the Gnome and KDE GUI environments in the period 1997-2000 had shown X11's fundamental soundness, but confirmed the authors' belief that the rendering system in X was woefully inadequate. One of us participated in the original X11 design meetings where the intent was to augment the rendering design at a later date; but the "GUI Wars" of the late 1980's doomed effort in this area. Good printing support has been particularly difficult to implement in X applications.

Most applications now composite images in sophisticated ways, whether it be in Flash media players, or subtly as part of anti-aliased characters. Bit-Blit is not sufficient for these applications, and these modern applications were (if only by their use of modern toolkits) all resorting to pixel based image manipulation. The screen pixels are retrieved from the window system, composited in clients, and then restored to the screen, rather than directly composited in hardware, resulting in poor performance. Inspired by the Plan 9 window system implemented, a graphics model based on Porter/Duff image compositing was chosen. This work resulted in the X Render extension and is described in detail at http://keithp.com/~keithp/talks/usenix2001.

X11's core graphics exposed fonts as a server side abstraction. This font model was, at best, marginally adequate by 1987 standards. Even WYSIWYG systems of that era found them insufficient. Much additional information embedded in fonts (e.g. kerning tables) were not available from X whatsoever. Current competitive systems implement anti-aliased outline fonts. Discovering the Unicode coverage of a font, required by current toolkits for internationalization, was causing major performance problems. Deploying new server side font technology is slow, as X is a distributed system, and many X servers are seldom (or never) updated.

Therefore, a more fundamental change in X's architecture was undertaken: to no longer use server side fonts at all, but to allow applications direct access to font files and have the window system cache and composite glyphs onto the screen.

The first implementation of the new font system (described at http://keithp.com/~keithp/talks/xtc2001) taught a vital lesson. Xft1 provided anti-aliased text and proper font naming/substitution support, but reverted to the core X11 bitmap fonts if the Render extension was not present. Xft1 included the first implementation what is called "subpixel decimation," which provides

higher quality subpixel based rendering than Microsoft's ClearType technology in a completely general algorithm.

Despite these advances, Xft1 received at best a lukewarm reception. If an application developer wanted anti-aliased text universally, Xft1 did not help them, since it relied on the Render extension which had not yet been widely deployed; instead, the developer would be faced with two implementations, and higher maintenance costs. This (in retrospect obvious) rational behavior of application developers shows the high importance of backwards compatibility; X extensions intended for application developers' use must be designed in a downward compatible form *whenever* possible, and should enable a *complete* conversion to a new facility, so that multiple code paths in applications do not need testing and maintenance. These principles have guided later development.

The font installation, naming, substitution, and internationalization problems were separated from Xft into a library named Fontconfig, (described in http://keithp.com/~keithp/talks/guadec2002) since some printer only applications need this functionality independent of the window system. Fontconfig provides internationalization features in advance of those in commercial systems such as Windows or OS X, and enables trivial font installation with good performance even when using thousands of fonts. Xft2 was also modified to operate against legacy X servers lacking the Render extension.

Xft2 and Fontconfig's solving of several major problems *and* lack of deployment barriers enabled rapid acceptance and deployment in the open source community, seeing almost universal use and uptake in less than one calendar year. They have been widely deployed on Linux systems since the end of 2002. They also "future proof" open source systems against coming improvements in font systems (e.g. OpenType), as the window system is no longer a gating item for font technology.

Sun Microsystems implemented a server side font extension for X in the last several years; for the reasons outlined in this section, it was not adopted by open source developers.

While Xft2 and Fontconfig finally freed application developers from the tyranny of X11's core font system, improved performance (see http://keithp.com/~keithp/talks/usenix2003/), and at a stroke simplified their printing problems, it has still left a substantial burden on applications. The X11 core graphics, even augmented by the Render extension, lack convenient facilities for many applications for even simple primitives like splines, tasteful wide lines, stroking paths, etc, much less provide simple ways for applications to print the results on paper.

Cairo

The Cairo library (www.cairographics.org), primarily implemented by Carl Worth of ISI, is designed to solve this problem. Cairo provides a stateful user-level API with support for the PDF 1.4 imaging model. Cairo provides operations including stroking and filling Bézier cubic splines, transforming and compositing translucent images, and anti-aliased text rendering. The PostScript drawing model has been adapted for use within applications. Extensions needed to support much of the PDF 1.4 imaging operations have been included. This integration of the familiar PostScript operational model within the native application language environments provides a simple and powerful new tool for graphics application development.

Cairo's rendering algorithms use work done in the 1980's by Guibas, Ramshaw, and Stolfi, which has never been exploited in Postscript or in Windows. The implementation is fast, precise, and numerically stable, supports hardware acceleration, and is in advance of commercial systems.

Cairo is in the late stages of development and is being widely adopted in the open source community. It includes the ability to render to Postscript, which should greatly improve applications' printing support. Work to incorporate Cairo in the Gnome and KDE desktop environments is well underway, as are ports to Windows and Macintosh. As with Xft2, Cairo works with all X servers, even those without the Render extension.

Accessibility and Eye-Candy

Several years ago, one of us implemented a prototype X system that used image compositing as the fundamental primitive for constructing the screen representation of the window hierarchy contents. Child window contents were composited to their parent windows which were incrementally composed to their parents until the final screen image was formed, enabling translucent windows. The problem with this simplistic model was twofold -- first, a naïve implementation consumed enormous resources as each window required two complete off screen buffers (one for the window contents themselves, and one for the window contents composited with the children) and took huge amounts of time to build the final screen image as it recursively composited windows together. Secondly, the policy governing the compositing was hardwired into the X server. An architecture for exposing the same semantics with less overhead seemed almost possible, and pieces of it were implemented (miext/layer). However, no complete system was fielded, and every copy of the code tracked down and destroyed to prevent its escape into the wild.

Both Mac OS X and DirectFB perform window-level compositing by creating off-screen buffers for each top-level window (in OS X, the window system is not nested, so there are only top-level windows). The screen image is then formed by taking the resulting images and blending them together on the screen. Without handling the nested window case, both of these systems provide the desired functionality with a simple implementation. This simple approach is inadequate for X as some desktop environments nest the whole system inside a single top-level window to allow panning, and X's long history has shown the value of separating mechanism from policy (Gnome and KDE were developed over 10 years after X11's design). The fix is pretty easy—allow applications to select which pieces of the window hierarchy are to be

stored off-screen and which are to be drawn to their parent storage.

With window hierarchy contents stored in off-screen buffers, an external application can now control how the screen contents are constructed from the constituent sub-windows and whatever other graphical elements are desired.

This eliminated the complexities surrounding precisely what semantics would be offered in window-level compositing within the X server and the design of the underlying X extensions. They were replaced by some concerns over the performance implications of using an external agent (the "Compositing Manager") to execute the requests needed to present the screen image. Note that every visible pixel is under the control of the compositing manager, so screen updates are limited to how fast that application can get the bits painted to the screen.

The architecture is split across three new extensions:

- Composite, which controls which sub-hierarchies within the window tree are rendered to separate buffers.
- Damage, which tracks modified areas with windows, informing the Composting Manager which areas of the off-screen hierarchy components have changed.
- Xfixes, which includes new Region objects permitting all of the above computation to be performed indirectly within the X server, avoiding round trips.

Multiple applications can take advantage of the off screen window contents, allowing thumbnail or screen magnifier applications to be included in the desktop environment.

To allow applications other than the compositing manager to present alpha-blended content to the screen, a new X Visual was added to the server. At 32 bits deep, it provides 8 bits of red, green and blue along with 8 bits of alpha value. Applications can create windows using this visual and the compositing manager can composite them onto the screen.

Nothing in this fundamental design indicates that it is used for constructing translucent windows; redirection of window contents and notification of window content change seems pretty far removed from one of the final goals. But note the compositing manger can use whatever X requests it likes to paint the combined image, including requests from the Render extension, which does know how to blend translucent images together. The final image is constructed programmatically so the possible presentation on the screen is limited only by the fertile imagination of the numerous eye-candy developers, and not restricted to any policy imposed by the base window system. And vital to rapid deployment, most applications can be completely oblivious to this background legerdemain.

In this design, such sophisticated effects need only be applied at frame update rates on only modified sections of the screen rather than at the rate applications perform graphics; this constant behavior is highly desirable in systems.

The results can be seen at http://freedesktop.org/~keithp/screenshots/. These also demonstrate the abilities of Cairo, Xft2 and the sophisticated font rendering already deployed on open source systems.

At this time, a full prototype X server implementation is working of the new compositing facilities described in this section that provides hardware accelerated Render support on a single hardware platform and limited video mode selection, and is available on the www.freedesktop.org.

Next Steps

We believe we are slightly more than half way through the process of rearchitecting and reimplementing the X Window System. The existing prototype needs to become a production system requiring significant infrastructure work as described in this section.

OpenGL based X

Current X-based systems which support OpenGL do so by encapsulating the OpenGL environment within X windows. As such, an OpenGL application cannot manipulate X objects with OpenGL drawing commands.

Using OpenGL as the basis for the X server itself will place X objects such as pixmaps and off-screen window contents inside OpenGL objects allowing applications to use the full OpenGL command set to manipulate them.

In concert with the new compositing extensions, conventional X applications can then be integrated into 3D immersive environments such as Croquet (www.opencroquet.org), or Sun's Looking Glass. X application contents can be used as textures and mapped onto any surface desired in those environments.

This work is underway, but not demonstrable at this date.

Mobility, Collaboration, and Other Topics

X's original intended environment included highly mobile students, and a hope, never generally realized for X, was the migration of applications between X servers.

The user should be able to travel between systems running X and retrieve your running applications (with suitable authentication and authorization). The user should be able to log out and "park" applications somewhere for later retrieval, either on the same display, or elsewhere. Users should be able to replicate an application's display on a wall projector for presentation. Applications should be able to easily survive the loss of the X server (most commonly caused by the loss of the underlying TCP connection, when running remotely).

Toolkit implementers typically did not understand and share this poorly enunciated vision and were primarily driven by pressing immediate needs, and X's design and implementation made migration or replication difficult to implement as an afterthought.

As a result, migration (and replication) was seldom implemented, and early toolkits such as Xt made it even more difficult. Emacs is the only widespread application capable of both migration and replication, and it avoided using any toolkit.

Recent work in some of the modern toolkits (e.g. GTK+) and evolution of X itself make much of this vision demonstrable in current applications. Some work in the X infrastructure is underway to enable the prototype in GTK+ to be finished.

Similarly, input devices need to become full-fledged network data sources, to enable much looser coupling of keyboards, mice, game consoles and projectors and displays; the challenge here will be the authentication, authorization and security issues this will raise.

The more than 10 year old color management facilities in X have never seen widespread use. This area is ripe for revisiting.

We are more than happy to hear from anyone interested in helping in this effort.