The (Re)Architecture of the X Window System

James Gettys
jim.gettys@hp.com

Keith Packard
keithp@keithp.com

HP Cambridge Research Laboratory

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

1 Introduction

While part of this work on the X window system [SG92] 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
- Users desire modern user interface eye candy, which sport translucent graphics and 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.

2 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, as fonts have were opaque X server side objects not directly accessible by 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 model first implemented in the Plan 9 window system, a graphics model based on Porter/Duff [PD84] image compositing was chosen. This work resulted in the X Render extension [Pac01a].

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 [Pac01b] 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 [Pla00] 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 [Pac02], (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 over the last several years; for the reasons outlined in this section, it has not been adopted by open source developers.

While Xft2 and Fontconfig finally freed application developers from the tyranny of X11's core font system, improved performance [PG03], 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.

3 Cairo

The Cairo library [WP03], developed by one of the authors in conjunction with by Carl Worth of ISI, is designed to solve this problem. Cairo provides a state full 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 [GRS83] along with work by John Hobby [Hob85], 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.

Of particular note is the current development of Glitz [NR04], an OpenGL backend for Cairo, being developed by a pair of master's students in Sweden. Not only is it showing that a high speed implementation of Cairo is possible, it implements an interface very similar to the X Render extension's interface. More about this in the OpenGL section below.

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 and a PDF back end is planned, 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 Apple's MacIntosh, and it is being used by the Mono project. As with Xft2, Cairo works with all X servers, even those without the Render extension.

4 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 [Hun04] 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.

There is very strong "pull" from both commercial and non-commercial users of X for this work and the current early version will likely be shipped as part of the next X.org Foundation X Window System release, sometime this summer. Since there has not been sufficient exposure through widespread use, further changes will certainly be required further experience with the facilities are gained in a much larger audience; as these can be made without affecting existing applications, immediate deployment is both possible and extremely desirable.

The mechanisms described above realize a fundamentally more interesting architecture than either Windows or Mac OSX, where the compositing policy is hardwired into the window system. We expect a fertile explosion of experimentation, experience (both good and bad), and a winnowing of ideas as these facilities gain wider exposure.

5 Input Transformation

In the "naïve", eye candy use of the new compositing functions no transformation of input events are required, as input to windows remains at the same geometric position on the screen, even though the windows are first rendered off screen. More sophisticated use, for example, screen readers or immersive environments such as Croquet [SRRK02], or Sun's Looking Glass [KJ04] requires transformation of input events from where they first occur on the visible screen to the actual position in the windows (being rendered from off screen), since the window's contents may have been arbitrarily transformed or even texture mapped onto shapes on the screen.

As part of Sun Microsystem's award winning work on accessibility in open source for screen readers, Sun has developed the XEvIE extension [Kre], which allows external clients to transform input events. This looks like a good starting point for the somewhat more general problem that 3D systems pose, and with some modification can serve both the accessibility needs and those of more sophisticated applications.

6 Synchronization

Synchronization is probably the largest remaining challenge posed by compositing. While composite has eliminated much flashing of the screen since window exposure is eliminated, this does not solve the challenge of the compositing manager happening to copy an application's window to the frame buffer in the middle of an application painting a sequence of updates. No "tearing" of single graphics operations take place since the X server is single threaded, and all graphics operations are run to completion.

The X Synchronization extension (XSync) [GCGW92], widely available but to date seldom used, provides a general set of mechanisms for applications to synchronize with each other, with real time, and potentially with other system provided counters. XSync's original design intent intended system provided counters for vertical retrace interrupts, audio sample clocks, and similar system facilities, enabling very tight synchronization of graphics operations with these time bases. Work has begun on Linux to provide these counters at long last, when available, to flesh out the design originally put in place and tested in the early 1990's.

A possible design for solving the application synchronization problem at low overhead may be to mark sections of requests with increments of XSync counters: if the count is odd (or even) the window would be unstable/stable. The compositing manager might then copy the window only if the window is in a stable state. Some details and possibly extensions to XSync will need to be worked out, if this approach is pursued.

7 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.

7.1 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.

A "proof of concept" of implementation of the X Render extension is being done as part of the Glitz back-end for Cairo, which is showing very good performance for render based applications. Whether the "core" X graphics will require any OpenGL extensions is still somewhat an open question.

In concert with the new compositing extensions, conventional X applications can then be integrated into 3D environments such as Croquet, 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.

7.2 Kernel support for graphics cards

In current open source systems, graphics cards are supported in a manner totally unlike that of any other operating system, and unlike previous device drivers for the X Window System on commercial UNIX systems. There is no single central kernel driver responsible for managing access to the hardware. Instead, a large set of cooperating user and kernel mode systems are involved in mutual support of the hardware, including the X server (for 2D graphic), the direct-rendering infrastructure (DRI) (for accelerated 3D graphics), the kernel frame buffer driver (for text console emulation), the General ATI TV and Overlay Software (GATOS) (for video input and output) and alternate 2D graphics systems like DirectFB.

Two of these systems, the kernel frame buffer driver and the X server both include code to configure the graphics card "video mode"—the settings needed to send the correct video signals to monitors connected to the card. Three of these systems, DRI, the X server and GATOS, all include code for managing the memory space within the graphics card. All of these systems directly manipulate hardware registers without any coordination among them.

The X server has no kernel component for 2D graphics. Long-latency operations cannot use interrupts, instead the X server spins while polling status registers. DMA is difficult or impossible to configure in this environment. Perhaps the most egregious problem is that the X server reconfigures the PCI bus to correct BIOS mapping errors without informing the operating system kernel. Kernel access to devices while this remapping is going on may find the related devices mismapped.

To rationalize this situation, various groups and vendors are coordinating efforts to create a single kernel-level entity responsible for basic device management, but this effort has just begun.

7.3 Housecleaning and Latency Elimination and Latency Hiding

Serious attempts were made in the early 1990's to multi-thread the X server itself, with the discovery that the threading overhead in the X server is a net performance loss [Smi92].

Applications, however, often need to be multi-threaded. The primary C binding to the X protocol is called Xlib, and its current implementation by one of us dates from 1987. While it was partially developed on a Firefly multiprocessor workstation of that era, something almost unheard of at that date, and some consideration of multi-threaded applications were taken in its implementation, its internal transport facilities were never expected/intended to be preserved when serious multi-threaded operating systems became available. Unfortunately, rather than a full rewrite as one of us expected, multi-threaded support was debugged into existence using the original code base and the resulting code is very bug-prone and hard to maintain. Additionally, over the years, Xlib became a "kitchen sink" library, including functionality well beyond its primary use as a binding to the X protocol. We have both seriously regretted the precedents both of us set introducing extraneous functionality into Xlib, causing it to be one of the largest libraries on UNIX/Linux systems. Due to better facilities in modern toolkits and system libraries, more than half of Xlib's current footprint is obsolete code or data.

While serious work was done in X11's design to mitigate latency, X's performance, particularly over low speed networks, is often limited by round trip latency, and with retrospect much more can be done [PG03]. As this work shows, client side fonts have made a significant improvement in startup latency, and work has already been completed in toolkits to mitigate some of the other hot spots. Much of the latency can be retrieved by some simple techniques already underway, but some require more sophisticated techniques that the current Xlib implementation is not capable of. Potentially 90the latency as of 2003 can be recovered by various techniques. The XCB library [MS01] by Bart Massey and Jamey Sharp is both carefully engineered to be multithreaded and to expose interfaces that will allow for latency hiding.

Since libraries linked against different basic X transport systems would cause havoc in the same address space, a Xlib compatibility layer (XCL) has been developed that provides the "traditional" X library API, using the original Xlib stubs, but replacing the internal transport and locking system, which will allow for much more useful latency hiding interfaces. The XCB/XCL version of Xlib is now able to run essentially all applications, and after a shakedown period, should be able to replace the existing Xlib transport soon. Other bindings than the traditional Xlib bindings then become possible in the same address space, and we may see toolkits adopt those bindings at substantial savings in space.

7.4 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. A more detailed description of this vision is available in [Get02].

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 (Xlib) 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 HAL and DBUS projects hosted at freedesktop.org are working on at least part of the solutions for the user interface challenges posed by hotplug of input devices.

7.5 Color Management

The existing color management facilities in X are over 10 years old, have never seen widespread use, and do not meet current needs. This area is ripe for revisiting. Marti Maria Saguer's LittleCMS [Mar] may be of use here. For the first time, we have the opportunity to "get it right" from one end to the other if we choose to make the investment.

7.6 Security and Authentication

Transport security has become an burning issue; X is network transparent (applications can run on any system in a network, using remote displays), yet we dare no longer use X over the network directly due to password grabbing kits in the hands of script kiddies. SSH [BS01] provides such facilities via port forwarding and is being used as a temporary stopgap. Urgent work on something better is vital to enable scaling and avoid the performance and latency issues introduced by transit of extra processes, particularly on (Linux Terminal Server Project (LTSP [McQ02]) servers, which are beginning break out of their initial use in schools and other non security sensitive environments into very sensitive commercial environments.

Another aspect of security arises between applications sharing a display. In the early and mid 1990's efforts were made as a result of the compartmented mode workstation projects to make it much more difficult for applications to share or steal data from each other on a X display. These facilities are very inflexible, and have gone almost unused.

As projectors and other shared displays become common over the next five years, applications from multiple users sharing a display will become commonplace. In such environments, different people may be using the same display at the same time and would like some level of assurance that their application's data is not being grabbed by the other user's application.

Eamon Walsh has, as part of the SELinux project [Wal04], been working to replace the existing X Security extension with an extension that, as in SELinux, will allow multiple different security policies to be developed external to the X server. This should allow multiple different policies to be available to suit the varied uses: normal workstations, secure workstations, shared displays in conference rooms, etc.

7.7 Compression and Image Transport

Many/most modern applications and desktops, including the most commonly used application (a web browser) are now intensive users of synthetic and natural images. The previous attempt (XIE [SSF+96]) to provide compressed image transport failed due to excessive complexity and over ambition of the designers, has never been significantly used, and is now in fact not even shipped as part of current X distributions.

Today, many images are being read from disk or the network in compressed form, uncompressed into memory in the X client, moved to the X server (where they often occupy another copy of the uncompressed data). If we add general data compression to X (or run X over ssh with compression enabled) the data would be both compressed and uncompressed on its way to the X server. A simple replacement for XIE (if the complexity slippery slope can be avoided in a second attempt) would be worthwhile, along with other general compression of the X protocol.

Results in our 2003 Usenix X Network Performance paper show that, in real application workloads (the startup of a Gnome desktop), using even simple GZIP [Gai93] style compression can make a tremendous difference in a network environment, with a factor of 300(!) savings in bandwidth. Apparently the synthetic images used in many current UI's are extremely good candidates for compression. A simple X extension that could encapsulate one or more X requests into the extension request would avoid multiple compression/uncompression of the same data in the system where an image transport extension was also present. The basic X protocol framework is actually very byte efficient relative to most conventional RPC systems, with a basic X request only occupying 4 bytes (contrast this with HTTP or CORBA, in which a simple request is more than 100 bytes).

With the great recent interest in LTSP in commercial environments, work here would be extremely well spent, saving both memory and CPU, and network bandwidth.

We are more than happy to hear from anyone interested in helping in this effort to bring X into the new millennium.

References

[BS01] Daniel J. Barrett and Richard Silverman. *SSH, The Secure Shell: The Definitive Guide*. O'Reilly & Associates, Inc., 2001.

[Gai93] Jean-Loup Gailly. *Gzip: The Data Compression Program*. iUniverse.com, 1.2.4 edition, 1993.

[GCGW92] Tim Glauert, Dave Carver, James Gettys, and David Wiggins. X Synchronization Extension Protocol, Version 3.0. X consortium standard, 1992.

[Get02] James Gettys. The Future is Coming, Where the X Window System Should Go. In *FREENIX Track, 2002 Usenix Annual Technical Conference*, Monterey, CA, June 2002. USENIX.

[GRS83] Leo Guibas, Lyle Ramshaw, and Jorge Stolfi. A kinetic framework for computational geometry. In *Proceedings of the IEEE 1983 24th Annual Symposium on the Foundations of Computer Science*, pages 100–111. IEEE Computer Society Press, 1983.

[Hob85] John D. Hobby. *Digitized Brush Trajectories*. PhD thesis, Stanford University, 1985. Also *Stanford Report STAN-CS-85-1070*.

[Hun04] A. Hundt. DirectFB Overview (v0.2 for DirectFB 0.9.21), February 2004. `http://www.directfb.org/documentation`.

[KJ04] H. Kawahara and D. Johnson. Project Looking Glass: 3D Desktop Exploration. In *X Developers Conference*, Cambridge, MA, April 2004.

[Kre] S. Kreitman. XEvIE - X Event Interception Extension. `http://freedesktop.org/~stukreit/xevie.html`.

[Mar] M. Maria. Little CMS Engine 1.12 API Definition. Technical report. `http://www.littlecms.com/lcmsapi.txt`.

[McQ02] Jim McQuillan. LTSP - Linux Terminal Server Project, Version 3.0. Technical report, March 2002. `http://www.ltsp.org/documentation/ltsp-3.0-4-en.html`.

[MS01] Bart Massey and Jamey Sharp. XCB: An X protocol c binding. In *XFree86 Technical Conference*, Oakland, CA, November 2001. USENIX.

[NR04] Peter Nilsson and David Reveman. Glitz: Hardware Accelerated Image Compositing using OpenGL. In *FREENIX Track, 2004 Usenix Annual Technical Conference*, Boston, MA, July 2004. USENIX.

[Pac01a] Keith Packard. Design and Implementation of the X Rendering Extension. In *FREENIX Track, 2001 Usenix Annual Technical Conference*, Boston, MA, June 2001. USENIX.

[Pac01b] Keith Packard. The Xft Font Library: Architecture and Users Guide. In *XFree86 Technical Conference*, Oakland, CA, November 2001. USENIX.

[Pac02] Keith Packard. Font Configuration and Customization for Open Source Systems. In *2002 Gnome User's and Developers European Conference*, Seville, Spain, April 2002. Gnome.

[PD84] Thomas Porter and Tom Duff. Compositing Digital Images. *Computer Graphics*, 18(3):253–259, July 1984.

[PG03] Keith Packard and James Gettys. X Window System Network Performance. In *FREENIX Track, 2003 Usenix Annual Technical Conference*, San Antonio, TX, June 2003. USENIX.

[Pla00] J. Platt. Optimal filtering for patterned displays. *IEEE Signal Processing Letters*, 7(7):179–180, 2000.

[SG92] Robert W. Scheifler and James Gettys. *X Window System*. Digital Press, third edition, 1992.

[Smi92] John Smith. The Multi-Threaded X Server. *The X Resource*, 1:73–89, Winter 1992.

[SRRK02] D. Smith, A. Raab, D. Reed, and A. Kay. Croquet: The Users Manual, October 2002. `http://glab.cs.uni-magdeburg.de/~croquet/downloads/Croquet0.1.pdf`.

[SSF+96] Robert N.C. Shelley, Robert W. Scheifler, Ben Fahy, Jim Fulton, Keith Packard, Joe Mauro, Richard Hennessy, and Tom Vaughn. X Image Extension Protocol Version 5.02. X consortium standard, 1996.

[Wal04] Eamon Walsh. Integrating XFree86 With Security-Enhanced Linux. In *X Developers Conference*, Cambridge, MA, April 2004. `http://freedesktop.org/Software/XDevConf/x-security-walsh.pdf`.

[WP03] Carl Worth and Keith Packard. Xr: Cross-device Rendering for Vector Graphics. In *Proceedings* *of the Ottawa Linux Symposium*, Ottawa, ON, July 2003. OLS.