Getting X Off The Hardware

Keith Packard
HP Cambridge Research Laboratory
keithp@keithp.com

Abstract

The X window system is generally implemented by directly inserting hardware manipulation code into the X server. Mode selection and 2D acceleration code are often executed in user mode and directly communicate with the hardware. The current architecture provides for separate 2D and 3D acceleration code, with the 2D code executed within the X server and the 3D code directly executed by the application, partially in user space and partially in the kernel. Video mode selection remains within the X server, creating an artificial dependency for 3D graphics on the correct operation of the window system. This paper lays out an alternative structure for X within the Linux environment where the responsibility for acceleration lies entirely within the existing 3D user/kernel library, the mode selection is delegated to an external library and the X server becomes a simple application layered on top of both of these. Various technical issues related to this architecture along with a discussion of input device handling will be discussed.

1 History

The X11[SG92] server architecture was designed assuming significant operating assistance for supporting input and output devices. How that has changed over the years will inform the discussion of the design direction proposed in this paper.

1.1 Original Architecture

One of the first 2D accelerated targets for X11 was the Digital QDSS (Dragon) board. The Dragon included a 1024x768 frame buffer with 4 or 8 bits for each pixel. The frame buffer was not addressable by the CPU, rather every graphics operation was performed by the coprocessor. The Dragon board had only a single video mode supporting the monitor supplied with the machine. A primitive terminal emulator in the kernel provided the text mode necessary to boot the machine.

Graphics commands to the processor were queued to a shared DMA buffer. The X server would block in the kernel waiting for space in the buffer when full.

Keyboard and mouse support were provided by another shared memory queue between the kernel and X server. Abstract event structures were constructed by the kernel from the raw device data, timestamped and placed in the shared queue. A file descriptor would be signalled when new data were inserted to awaken the X server, and the X server could also directly examine the queue indices which were stored in the shared segment. This low-overhead queue polling was used by the X server to check for new input after every X request was executed to reduce input latency.

The hardware sprite was handled in the kernel; it's movement was directly connected with the mouse driver so that it could be moved at interrupt time, leading to a responsive pointer even in the face of high CPU load within the X server and other applications. The keyboard controller managed the transition from ASCII console mode to key-transition X mode internally; abnormal termination of the X server would leave the underlying console session working normally.

1.2 The Slippery Slope

Early Sun workstations had unaccelerated frame buffers. Like the QDSS above, they used fixed monitors and had no need to support multiple video modes. As the hardware advanced, they did actually gain programmable timing hardware, but that was not configurable from the user mode applications.

The X server simply mapped the frame buffer into its address space and manipulated the pixel values directly. Around 1990, Sun shipped the cgsix frame buffer which included an accelerator. Unlike the QDSS, the cgsix frame buffer could be mapped by the CPU, and the accelerator documentation was not published by Sun. X11R4 included support for this card as a simple dumb frame buffer. As CPU access to the frame buffer was slower than with Sun's earlier unaccelerated frame buffers, the result was a much slower display.

By disassembling the provided SunWindows driver, the author was able to construct an accelerated X driver for X11R5 entirely in user mode. This driver could not block waiting for the accelerator to finish, rather it would spin, polling the accelerator until it indicated it was idle.

Keyboard and mouse support were provided by the kernel as files from which events could be read. The lack of any shared memory mechanism to signal available input meant that the original driver would not notice input events until the X server polled the kernel, something which could take significant time. As there was no kernel support for the pointer sprite, the X server was responsible for updating it as well, leading to poor mouse tracking when the CPU was busy.

To ameliorate the poor mouse tracking, the X server was modified to receive a signal when input was present on the file descriptors and immediately process the input. When supported, the hardware sprite would also be moved at this time, leading to improved tracking performance. Still, the fact that the X server itself was responsible for connecting the mouse motion to the sprite location meant that under high CPU load, the sprite would noticeably lag the mouse.

Kernel support for the keyboard consisted of a special mode setting which would transform the keyboard from an ASCII input device to reporting raw key transition events. Because the kernel didn't track what state the keyboard was in, the X server had to carefully reset the keyboard on exit back to ASCII mode or the user would no longer be able to interact with the console.

Placing the entire graphics driver in user mode eliminated the need to write a kernel driver, but marginalized overall system performance by forcing the CPU to busy-wait for the graphics engine.

Fixing the kernel to address these problems was never even considered; the problems didn't prevent the system from functioning, they only made it less than ideal.

1.3 The Dancing Bear

With widespread availability of commodity 386-based PC hardware, numerous vendors began shipping Unix (and Unix-like) operating systems for them. These originally did not include the X window system. A disparate group of users ported X to these systems without any support from the operating system vendors.

That these users managed to get X running on the early 386 hardware was an impressive feat. That they had to do everything without any kernel support only increased the difficulty.

Early PC graphics cards were simple frame buffers as far as graphics operations went, but configuring them to generate correct video timings was far from simple. Because monitors varied greatly, each graphics card could be programmed to generate many different video timings. Incorrect timings could destroy the monitor.

Keyboard support in these early 386-based Unix systems was much like the Sun operating system; the keyboard was essentially a serial device and could be placed in a mode which translated key transitions into ASCII or placed a mode which would report the raw bytes emitted from the keyboard.

The X server would read these raw bytes and convert them to X events. Again, there was latency here as the X server would not process them except when polling for input across all X clients and input devices. As with the Sun driver, if the X server terminated without switching the keyboard back to translated mode, it would not be usable for the console. This particular problem was eventually kludged in Linux by adding special key sequences to reset the keyboard to translated mode.

Mouse support really was just a kernel serial driver—PS/2 mice didn't exist, and so bus and serial mice were used. The X server itself would open the device, configure the communication parameters and parse the stream of bytes. As there was no hardware sprite support, the X server would also have to draw the cursor on the screen; that operation had to be synchronized with rendering and so would be delayed until the server was idle.

Because the X server itself was managing video mode configuration, an abnormal X server termination would leave the video card misconfigured and unusable as the console. Similarly, the keyboard driver would be left in untranslated mode, so the user couldn't even operate the computer blind to reboot.

The X server has assumed the same reliability requirements as the operating system kernel itself; bugs in the X server would render the system just as unusable as bugs in the kernel.

1.4 The Pit of Despair

With the addition of graphics acceleration to the x86 environment, the X server extended its user-mode operations to include manipulation of the accelerator. As with the Sun GX driver described above, these drivers included no kernel support and were forced to busy-wait for the hardware.

However, unlike the GX hardware, PC graphics hardware would often tie down the PCI bus while transferring data between the CPU and the graphics card. Incorrect manipulation of the hardware would result in the PCI bus locking and the system not even responding to network or disk activity. Unlike the simple keyboard translation problem described above, this cannot be be fixed in the operating system.

Because the graphics devices had no kernel driver support, there was no operating system management of their address space mappings. If the BIOS included with the system incorrectly mapped the graphics device, it fell to the X server to repair the PCI mapping spaces. Manipulating the PCI address configuration from a user-mode application would work only on systems without any dynamic management within the kernel.

If the machine included multiple graphics devices controlled through the standard VGA addresses, the X server would need to manipulate these PCI mappings on the fly to address the active card.

The overall goal was not to build the best system possible, but rather to make the code as portable as possible, even in the face of obviously incorrect system architecture.

1.5 A Glimmer of Hope

The Mesa project started as a software-only rasterizer for the OpenGL API. By providing a freely available implementation of this widely accepted API, people could run 3D applications on every machine, even those without custom 3D acceleration hardware. Of course, performance was a significant problem, especially as the 3D world moved from simple colored polygons to textures and complex lighting environments.

The Mesa developers started adding hardware support for the few cards for which documentation was available. At first, these were whole-screen drivers, but eventually the DRI project was started to support multiple 3D applications integrated into the X window system. Because of the desire to support secure direct rendering from multiple unprivileged applications, the DRI project had to include a kernel driver. That driver could manage device mappings, DMA and interrupt logic and even clean up the hardware when applications terminated abnormally.

All 3D commands are written to command buffers passed from the user application through the kernel to the device; the kernel can perform validation on the contents before queuing them for transfer to the graphics card. In many ways, this system is similar to the old Dragon X server described above.

The result is a system which is stable in the face of broken applications, and provides high performance and low CPU overhead. However, the DRI environment remains reliant on the X server to manage video mode selection and basic device input.

2 Forward to the Past

Given the dramatic changes in system architecture and performance characteristics since the original user-mode X server architecture was promulgated, it makes sense to look at how the system should be constructed from the ground up. Questions about where support for each operation should live will be addressed in turn, first starting with graphics acceleration, then video mode selection and finally (and most briefly) input devices.

3 Graphics Acceleration

X has always directly accessed the lowest levels of the system to accelerate 2D graphics. Even on the QDSS, it constructed the register-level instructions within the X server itself. With the inclusion of OpenGL[SAe99] 3D graphics in some systems, the system requires two separate graphics drivers, one for the X server operating strictly in 2D mode and the other inside the GL library for 3D operations. Improvements to the 3D support have no effect on 2D performance.

As a demonstration of how effectively OpenGL can implement the existing X server graphics operations, Peter Nilsson and David Reveman implemented the Glitz library[NR04] which supports the Render[Pac01] API on top of the OpenGL API. In a few months, they managed to provide dramatic acceleration for the Cairo graphics library[WP03] on any hardware with an OpenGL implementation. In contrast, the Render implementation within the X sample server using custom 2D drivers has never seen significant acceleration, even three and a half years after the extension was originally designed. Only a few drivers include even half-hearted attempts at acceleration.

The goal here is to have the X server use the OpenGL API for all graphics operations. Eliminating the custom 2D acceleration code will reduce the development burden. Using accelerated OpenGL drivers will provide dramatic performance improvements for important operations now ill-supported in existing X drivers. Work in this area will depend on the availability of stand-alone OpenGL drivers that work in the absence of an underlying window system. Fortunately, the Mesa project is busy developing the necessary infrastructure. Meanwhile, development can progress apace using the existing window-system dependent implementations, with the result that another X server is run just to configure the graphics hardware and set up the GL environment.

For cards without complete OpenGL acceleration, the desired goal is to provide DRI-like kernel functionality to support DMA and interrupts to enable efficient implementation of whatever useful operations the card does support. For 2D graphics, the operations needing acceleration are those limited by memory bandwidth—large area fills and copies. In particular acceleration of image composition results in dramatic performance improvements with minimal amounts of code. The spectacular amounts of code written in the past that provide modest acceleration for corner cases in the X protocol should be removed and those cases left to software to minimize driver implementation effort.

This architecture has been implemented by Eric Anholt in his kdrive-based Xati server[Anh04]. Using the existing DRI driver for the Radeon graphics card, he developed a 2D X driver with reasonable acceleration for common operations, including significant portions of the X render API. The driver uses only a small fraction of the Radeon DRI driver, a significantly smaller kernel driver would suffice for a ground-up implementation.

In summary, graphics cards should be supported in one of two ways:

1. With an OpenGL-based X server
2. With a 2D-only X server based on a simple loadable driver API.

3.1 Implications for Applications

None of the architectural decisions about the internal X server architecture change the nature of the existing X and Render APIs as the fundamental 2D interface for applications. Applications using the existing APIs will simply find them more efficient when the X server provides a better implementation for them. This means that applications needn't migrate to non-X APIs to gain access to reasonable acceleration.

However, applications that wish to use OpenGL should find a wider range of supported hardware as driver writers are given the choice of writing either an OpenGL or 2D driver, and aren't faced with the necessity of starting with a 2D driver just to support X.

In any case, use of the cairo graphics library provides insulation from this decision as it supports X and GL requiring only modest changes in initialization to select between them.

4 Video Mode Configuration

The area of video mode selection involves many different projects and interests; one significant goal of this discussion is to identify which areas are relevant to X and how those can be separated from the larger project.

4.1 Overview of the Problem

Back in 1984 when X was designed, graphics devices were fundamentally fixed in their relationship with the attached monitor. The hardware would be carefully designed to emit video timings compatible with the included monitor; there was no provision for adjusting video timings to adapt to different monitors, each video card had a single monitor connector.

Fast forward to 2004 when common video cards have two or more monitor connectors along with outputs for standard NTSC, SECAM or PAL video formats. The desire to dynamically adjust the display environment to accommodate different use modes is well supported within the Apple OS X and Microsoft Windows environments, but the X window system has remained largely stuck with its 1984 legacy.

4.2 X Attempts to Fix Things

X servers for PC operating systems adapted to simple video mode selection by creating a 'virtual' desktop at least as large as the largest desired mode and making the current mode view a subset of that, panning the display around to keep the mouse on the screen. For users able to accept this metaphor, this provided usable, if less than ideal support. Most of the time, however, having content off of the screen which could only be reached by moving the mouse was confusing. To help address this, the X Resize and Rotate extension (RandR)[GP01] was designed to notify applications of changes in the pixel size of the screen and allow programmatic selection among available video modes.

The RandR extension solved the simple single monitor case well enough, even permitting the set of available modes to change on the fly as monitors were switched. However, it failed to address the wider problem of supporting multiple different video outputs and the dynamic manipulation of content between them.

Statically, the X server can address each video output correctly and even select between a large display spanning a collection of outputs or separate displays on each video screen. However, there is no capability to adjust these configurations dynamically, nor even to automatically adapt to detected changes in the environment.

4.3 X is Only Part of the Universe

With 2D performance no longer a significant marketing tool, graphics hardware vendors have been focusing instead on differentiating their products based on video output (and input) capabilities. This has dramatically extended the options available to the user, and increased the support necessary within the operating system.

As the suite of possible video configuration options continues to expand, it seems impossible to construct a fixed, standard X extension capable of addressing all present and future needs. Therefore, a fully capable mechanism must provide some "back door" through which display drivers and user agents can communicate information about the video environment which is not directly relevant to the window system or applications running within it.

One other problem with the current environment is that video mode selection is not a requirement unique to the X window system. Numerous other graphical systems exist which are all dependent on this code. Currently, that is implemented separately for each video card supported by each system. The MxN combination of graphics systems and video cards means that only a few systems have support for a wide range of video cards. Support for systems aside from X is pretty sparse.

4.4 Who's in Charge Here, Anyway?

X itself places relatively modest demands on the system. The X server needs to be aware of what video cards are available, what video modes are available for each card and how to select the current mode. Within that mode there may be a wealth of information that is not relevant to the X server; it really only needs to know the pixel dimensions of each frame buffer, the physical dimension of pixels on each monitor and the geometric relationship among monitors. Details about which video port are in use, or how the various ports relate to the frame buffer are not important. Information about video input mechanisms are even less relevant.

As the X server need have no way of interpreting the complexity of the video mode environment, it should have no role in managing it. Rather, an external system should assume complete control and let the X server interact in its own simple way.

This external system could be implemented partially in the kernel and partially in user-mode. Doing this would allow the kernel to share the same logic for video mode selection during boot time for systems which don't automatically configure the video card suitably on power-on. In addition, alternate graphics systems would be able to share the same API for their own video mode configuration.

5 Input Device Support

In days of yore, the X environment supported exactly one kind of mouse and one (perhaps of an internationalized family) keyboard. Sadly, this is no longer the case. The wealth of available input devices has caused no small trouble in X configuration and management. Add to that the relative failure of the X Input extension to gain widespread acceptance in applications and the current environment is relegated to emulating that available in 1984.

5.1 Uniform Device Access

The first problem to attack is that of the current hodgepodge device support where the X server itself is responsible for parsing the raw bytestreams coming from the disparate input devices. Fortunately, the kernel has already solved that problem—the new /dev/input based drivers provide a uniform description of devices and standard interface to all. Converting the X server over to those interfaces is straightforward.

However, the /dev/input/mice interface has a significant advantage in todays world; it unifies all mouse devices into a single stream so that the X server doesn't have to deal with devices that come and go. So, to switch input mechanisms, the X server must first learn to deal with that.

5.2 Hotplug and HAL

Mice (and even keyboards) can be easily attached and detached from the machine. With USB, the system is even automatically notified about the coming and going of devices. What is missing here is a way of getting that notification delivered to the X server, having the X server connect to the new device (when appropriate), notifying X applications about the availability of the new device and integrating the devices events into the core pointer or keyboard event stream.

The Hardware Abstraction Layer (HAL)[Zeu] project is designed to act as an intermediary between the Linux Hotplug system and applications interested in following the state of devices connected to the machine. By interposing this mechanism, the complexity of discovering and selecting input devices for the X server can be moved into a separate system, leaving the X server with only the code necessary to read events from the devices specified by the HAL. One open question is whether this should be done by a direct connection between the X server and the HAL daemon or whether an X client could listen to HAL and transmit device state changes through the X protocol to the X server.

One additional change needed is to extend the X Input Extension to include notification of new and departed devices. That extension already permits the list of available devices to change over time, all that it lacks is the mechanism to notify applications when that occurs. Inside the X server implementation, the extension is in for some significantly more challenging changes as the current codebase assumes that the set of available devices is fixed at server initialization time.

6 Migrating Devices

When X was developed, each display consisted of a single keyboard and mouse along with a fixed set of monitors. That collection was used for a single login session, and the input devices never moved. All of that has now changed; input devices come and go, computers get plugged into video projectors, multiple users login to the same display. The dynamic nature of the modern environment requires some changes to the X protocol in the form of new or modified extensions.

6.1 Whose Mouse Is This?

Input devices are generally located in physical proximity to the related output device. In a system with multiple output devices and multiple input devices, there is no existing mechanism to identify which device is where. Perhaps some future hardware advance will include geographic information along with the bus topology.

The best we can probably do for now is to provide a mechanism to encode in the HAL database the logical grouping of input and output devices. That way the X server would receive from the HAL the set of devices to use at startup time and then accept ongoing changes in that as the system was reconfigured.

One problem with this simplistic approach is that it doesn't permit the migration of input devices from one grouping to another; one can easily imagine the user holding a wireless pointing device to attempt to interact with the "wrong" display. Some mechanism for dynamically reconfiguring the association database will need to be included.

6.2 Hotplugging Video Hardware

While most systems have no ability to add or remove graphics cards, it's not unheard of—many handheld computers support CF video adapters. On the other hand, nearly all systems do support "hotplugging" of the actual display device or devices. Many can even detect the presence or absence of a monitor enabling true auto-detection and automatic reconfiguration.

When a new monitor is connected, the X server needs to adapt its configuration to include it. In the case where the set of physical screens are gathered together as a single logical screen, the change can be reflected by resizing that single screen as supported by the RandR extension. However, if each physical screen is exposed to applications as a separate logical screen, then the X server must somehow adapt to the presence of a new screen and report that information to applications. This will require an extension.

In terms of the existing X server implementation, the changes are rather more dramatic. Again, it has some deep-seated assumptions that the set of hardware under its control will not change after startup. Fixing these will keep developers entertained for some time.

6.3 Virtual Terminal Switching

One capability Linux has had for a long time is the ability to rapidly switch among multiple sessions with "virtual terminals". The X server itself uses this to preserve a system console, running on a separate terminal ensures that the system console can be viewed by simply switching to the appropriate virtual terminal. Given this, multiple X servers can be started on the same hardware, each one on a different virtual terminal and rapidly switched among.

The virtual terminal mechanism manages only the primary graphics device and the system keyboard. Management of other graphics and input devices is purely by convention. The result is that multiple simultaneous X sessions are not easily supported by the standard build of the X server. The X server targeted at a non-primary graphics device needs to avoid configuring the virtual terminal. However, this also eliminates the ability for that device to support multiple sessions; there cannot be virtual terminal switching on a device which is not associated with any virtual terminals.

With the HAL providing some indication of which devices should be affiliated into a single session configuration, the X server can at least select them appropriately. Similarly, the X server should be able to detect which device is the console keyboard and manage virtual terminals from there. Whether the kernel needs to add support for virtual terminals on the other graphics/keyboard devices is not something X needs to answer.

The final problem is that of other input devices; when switching virtual terminals, the X server conventionally drops its connection to the other input devices, presuming that whatever other program is about to run will want to use the same ones. While that does work, it leaves open the possibility that an error in the X server will leave these devices connected and deny other applications access to them. Perhaps it would be better if the kernel was involved in the process and directing input among multiple consumers automatically as VT affiliation changed.

7 Conclusion

Adapting the X window system to work effectively and competently in the modern environment will take some significant changes in architecture, however throughout this process existing applications will continue to operate largely unaffected. If this were not true, the fundamental motivation for the ongoing existence of the window system would be in doubt.

Migrating responsibility for device management out of the X server and back where it belongs inside the kernel will allow for improvements in system stability, power management and correct operation in a dynamic environment. Performance of the resulting system should improve as the kernel can take better advantage of the hardware than is possible in user mode.

Sharing graphics acceleration between 2D and 3D applications will reduce the effort needed to support new graphics hardware. Migrating the video mode selection will allow all graphics systems to take advantage of it. This should permit some interesting exploration in system architecture.

Significant work remains in defining the precise architecture of the kernel video drivers; these drivers need to support console operations, frame buffer device access and DRI (or other) 3D acceleration. Common memory allocation mechanism seem necessary, along with figuring out a reasonable division of labor between kernel and user mode for video mode selection.

Other work remains to resolve conflicts over sharing devices among multiple sessions and creating a mechanism for associating specific input and output devices together.

The resulting system regains much of the flavor of the original X11 server architecture. The overall picture of a system which provides hardware support at the right level in the architecture appears to have wide support among the relevant projects making the future prospects bright.

References

[Anh04] Eric Anholt. High Performance X Servers in the Kdrive Architecture. In *FREENIX Track, 2004 Usenix Annual Technical Conference*, Boston, MA, July 2004. USENIX.

[GP01] Jim Gettys and Keith Packard. The X Resize and Rotate Extension - RandR. In *FREENIX Track, 2001 Usenix Annual Technical Conference*, Boston, MA, June 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.

[Pac01] Keith Packard. Design and Implementation of the X Rendering Extension. In *FREENIX Track, 2001 Usenix Annual Technical Conference*, Boston, MA, June 2001. USENIX.

[SAe99] Mark Segal, Kurt Akeley, and Jon Leach (ed). *The OpenGL Graphics System: A Specification*. SGI, 1999.

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

[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.

[Zeu] David Zeuthen. HAL Specification 0.2. `http://freedesktop.org/~david/hal-0.2/spec/hal-spec.html`.