The State of Accelerated-X™

Wm. E. Davis
Xi Graphics, Inc.

This paper is intended for those who are using or considering the use of UNIX® operating systems (including Linux-based OSs) with the X Window System and current commercial graphics hardware, and who need good graphics sub-system software with a fast, solid graphics driver for that graphics hardware, and prompt customer support for that sub-system software. In addition to these requirements, there may be a need for support of special graphics features or a particular OS or computer platform. While the subject can be very technical, the paper is written more for the Manager of Product Development or the Program Manager rather than for the software engineer.

Xi Graphics, Inc. has been developing and licensing its own implementation of the X Window System ("X") and the "graphics drivers" that are used with this implementation for over ten years now. The trade name for the software is Accelerated-X, chosen partly because the software uses the graphics hardware to its fullest capability to accelerate the graphics operations - i.e., the software is "hardware-accelerated" - and partly because Xi Graphics' entire implemention of X is architected for high performance - "acceleration through architecture," so to speak.

X Window System - What is it?

X goes back some twenty years and had it origins at MIT in the Athena Project. Many papers have been written on the subject, so we won't spend much time on its history here. X has progressed through a number of major releases over those years and has passed from MIT to the X Consortium to X.org who is responsible for maintaining the specifications of X, generally known as X11Rx.x.

The specifications for X focus on the "what, not how" or put another way, "Policy not Procedures." Thus the outcomes are specified, but how one gets there is up to the implementer. A "Sample Implementation" (SI) is available with the specifications to show how one might go about implementing the X sub-system. The SI is not production software, so in order to produce a really usable package, a great deal of re-do is needed. And each re-do is different. Thus the internals of the X server - the major portion of X - is different among HP, Sun, IBM, Xi Graphics, X.org (the successor to XFree86), and so on.

A typical UNIX operating system is shown in Figure 1. In this particular case, the Solaris system is named, but it could be HP/UX or AIX and the picture would be the same.

The X portion of the OS is the X server (including the graphics driver). The kernel and kernel drivers for I/O & Comm interfaces, the GUI, and the window manager are parts of Solaris, and are not part of X. All of the slate-colored boxes are provided by a single source - Sun Microsystems in this case - and will most assuredly work together properly. The X implementation is a Sun product, known as Xsun. The graphics driver portion of the X server, known in the trade as the "ddx" - or (graphics) device dependent part of X - is also a Sun product. The GUI, window manager and kernel stuff are Sun products.

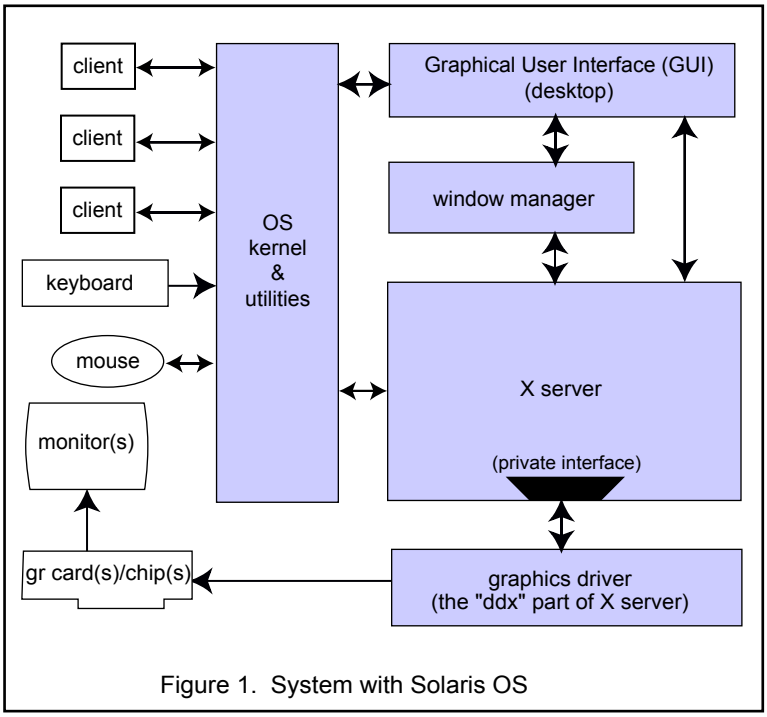

Figure 1. System with Solaris OS

Isolating X from the OS and expanding it a bit to show some more details, we have Figure 2 which depicts remote clients (they are not on the same computer as is the display server). As the authors of the 1992 book *The X Window System Server*, Elias Israel and Erik Fortune noted, "The server is such a large and complex system that we couldn't fit all the interesting topics between the covers of one book" (which had over 500 pages), depicting the X server in a simple way was also a bit challenging. Another book published in the same year, *X Window System*, by Robert W. Scheifler & James Gettys, is well over 900 pages. So the X Window System is a sizable body of work to which we can attest, since Xi Graphics makes its living developing, licensing and supporting X sub-system software for many graphics hardware architectures, OSs, and computer platforms. Unfortunately, there are some in the user community who have treated X as "just another utility" much to their detriment. Maybe this paper can help prevent more of these blunders.

The X Server Disected (a little)

Graphics driver - Beginning at the "back-end," the item most often (erroneously) referred to as the "graphics driver" - the ddx - is the part of the X server that is dedicated to a particular graphics architecture - the graphics chip and associated output channels to drive the attached monitors. Recently the complexity of the "channel" part of graphics cards has increased substantially, with up to four monitors (as this is written) being driven from a graphics card with a single graphics chip. It has been common to see two monitors connected to one graphics chip, but four is pretty impressive (e.g., the Matrox QID cards). The graphics driver has the resposibility of using all of

the available hardware on the graphics chip/card to extract the maximun graphics rendering and display performance from that hardware, in both 2D and 3D operations. The driver can also be made to use available graphics hardware in the 3D section of the chip to help implement interesting features for 2D displays (at least in the case of Accelerated-X drivers).

The ddx is only the "device dependent" part of the graphics driver; there is another part of the driver - "the device independent" part, known as the "dix" - that is in the larger part of the X server shown in Figure 2. If the X server architecture already includes the framework or infrastructure to accomodate all of the features and capabilities needed by a previously unsupported new graphics chip/card, then only the ddx, the "graphics driver" need be written. Most graphics chips of the same vintage tend to have similar features and capabilities implemented in slightly different ways and with different register designations, memory sizes, clock speeds, and so on. Once the dix portion of the X server has been properly designed to implement the archicture from one manufacturer, it should also support the (somewhat similar) architectures from other manufacturers. That leaves only the ddx portion of the (total) graphics driver to be written.

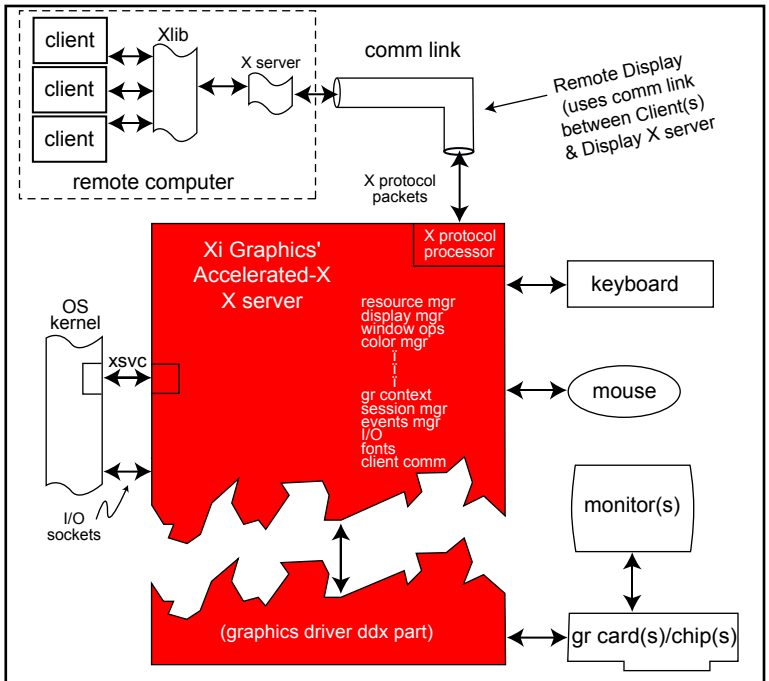

Figure 2. Accelerated-X on local computer, clients are remote.

Since Xi Graphics has been privileged to have access (under Non-Disclosure Agreements) to the confidential information on the details of graphics chip hardware architecture from a good number of chip manufacturers over the years, we have been able to architect the X servers in ways to easily accomodate the second, third, fourth manufacturer's chips as they came along, without having to do much tweaking on the basic structure. The result has been Accelrated-X server implementations that are well designed, stable both in operation and code base, and feature rich, which makes the task of adding support for a new chip to the portfolio straight-forward and rather routine.

The ddx and dix together provide all of the graphics features that can be used in a graphics-based X system for a particular graphics architecture, limited only by the features supported in hardware

when it comes to hardware acceleration. Features such as rotating the displayed image(s), stretching images across multiple monitors driven by separate graphics chips, anti-aliasing text, using hardware overlay image planes and video windows, providing fast and accurate record/playback capabilities, etc., can be implemented with relative ease in a well designed X server. Otherwise, implementing such features/capabilities can result in some severe hacking, with the usual poor performance, instabiltiy, and difficult maintenance - as is found in many freeware (open source) graphics systems

A **kernel driver** we call the "xsvc driver" is included with each Accelerated-X server/driver package. The xsvc driver is the only kernel driver required by Accelertate-X, and is used primarily for establishing with the kernel the memory and DMA resources X will need. Because our graphics sub-system does not depend upon the OS kernel to do "graphics work," it is divorced as much as possible from the particular kernel on which it is running. Thus portability is simplified and maintenance is minimized relative to supporting the many UNIX OSs and their various versions.

Contrast this "keep the kernel at arms' length" approach with the open source efforts at X.org and its contributors. X.org developers - in concert with the Linux kernel developers - seem to strive to move more and more of the graphics sub-system into the kernel, and are currently even working to move peripheral device drivers into the kernel on a per device basis, instead of having a general interface to which peripheral devices can communicate, such as is used with Solaris for example. One motivation for this effort, it seems, is to force proprietary device drivers to be open source, since anything in the Linux kernel must be open source. Never mind that it makes a hash out of maintenance, and will eliminate the availablility of some peripherals for Linux users. Open Source is the objective. Or maybe the Religion.

X11 Server Extensions were anticipated to be needed to accomodate new features and capabilities over time, and many have been sanctioned. Accelerated-X supports the official extensions, and some of the unofficial ones. (An "unofficial" extension is defined here as an extension that found its way into an X.org release without the benefit of specifications for such and/or being officially adopted by the organization responsible for maintaining the standards/specifications of X). A couple of such unofficial extensions come to mind as examples - the RandR and Render extensions. There are other examples. Unlike the oversignt of OpenGL, X seems to have lost the oversight and controls mechanisms that were present to some degree in the past. This is unfortunate, since it is apparent that those mechanisms are sorely needed. In the case of the Render extension, it has been included in releases by X.org, without the benefit of specifications, and Xi Graphics has declined to implement portions of it because those portions seem to have no reason to exist. "Render" is used in Accelerated-X primarily with fonts.

High performance and low maintenance are - or should be - serious considerations for X users. High performance is not obtained just by the hardware. Xi Graphics has seen many examples where the latest, hotest graphics hardware has been brought to its knees by poor graphics sub-system software performance. While these items are not represented by a box or two in one of the Figures, software that is poorly architected and badly designed will almost always exhibit poor performance and high maintenance requirements. The (generally low) level of performance, and the (generally high) level of maintenance needs of open source X sub-system software exemplifies the difference between the open source "community" approach in all of its glory, and the commercial, for profit, "customers are King" approach in all its "evil."

Accelerated-X graphics drivers are designed for high performance and low maintenance. This is also true of the other parts of the X server architecture of course, but the other parts are not subject to the high rate of change as are the graphics driver parts. Xi Graphics licenses its X graphics sub-system, requiring a license fee for each computer on which it is installed. In the face of competition from free open source X servers and graphics drivers, this policy would seem unworkable. And for the home user who will put up with unstable operation and no available service except for the plea "Have X.org graphics, Please Help, God Bless" broadcast on the Internet, the idea of paying for something that can be had for free is unthinkable.

For organizations who can not tolerate poor performance and unstable operation in their graphics-based system, and who need the assurance of fast customer support when a bug surfaces or a change is needed, the free X server solutions may not be the ticket. Xi Graphics charges a fee for the use of its software, but the customer support is free. This model is exactly the opposite of the freeware open source X servers/drivers, where the emphasis is to get something "out there" and worry about problems later. This creates a market for the individuals who market their services as troubleshooters in Linux, but the overall results can be quite expensive to the customer in terms of moneys paid, production delays, lost business, and lost customers. Because Xi Graphics provides its standard customer support services for free, our emphasis is on implementing good architectural designs that are thoroughly tested on a number of boxes and graphics cards/chips before releasing the software. We devote a lot of effort in making sure not much service is needed, or otherwise we wouldn't be in business.

An OpenSource Myth

The myth that availablity of the source code for the X server/graphics driver graphics sub-system would solve all (or at least most) problems of X graphics, is just that - a myth. First, there is the underlying myth that the graphics chip manufacturers such as ATI, Intel, Matrox, and Nvidia, who have invested hundreds of millions of dollars developing ever increasingly sophisticated graphics

technology in their graphics chip designs trying to gain an advantage over each other, will decide to make this expensive (and highly protected) technology publicly available. Something about a "cold day" comes to mind. But even if these firms made their technology available - by releasing the source code of their graphics drivers - the complexity of the technology is such that only those well versed in the art, as the lawyers say, could begin to understand the code and make improvements to it or fix bugs in a manner that would be useful in a commercial system.

But the bigger point is this: there *are* individuals who are well versed in the art of writing graphics drivers for graphics chips, and most of them work at ATI, Intel, Matrox, and Nvidia writing drivers for those complex chips for use on Windows and on UNIX (X). Yet the complaint level from users of those drivers for X (mostly on Linux) is very high. These individuals are experts, well trained in the science, and experienced in the art, and, presumably very bright. So why do the "Linux drivers" produced by these firms have so many complaints lodged against them, while the Linux drivers from Xi Graphics get universal kudos from their users on the same graphics chips (except Nvidia chips, which Xi Graphics is not allowed - by Nvidia - to support, but whose users constantly approach Xi Graphics for Linux support)? If you guessed the X server, you get the prize.

Freeware Open Source X servers

The graphics driver writers at the graphics chip manufacturers start in a deep hole when beginning to write a graphics driver for Linux. That hole is caused by the fact that the open source freeware X servers - primarily those known as XFree86 and X.org servers - are very poorly architected, designed and implemented. The Sample Implementation (SI) that was provided back about '87 was just an example of how one *might* design the code to implement the X Window System as was specified back then. The specifications stressed "Policy not Procedures," or put another way, "not How, but What". The SI was a "how it might be done" and was probably intended to give some hints about how one might implement the specifications. The commercial X servers that implemented the X specifications apparently used very little code from the SI without extensive changes. These servers were designed by commercial enterprises who had customers to satisfy, and competitors with which to contend. The good name of the firms was at stake, and the X servers were required to be of sufficiently high quality to not besmirch that name.

In the case of the X servers produced by the open source "community" over the past twenty years, the motivation of the contributors seems to have been considerably different from that of the commerical firms. David Dawes, who at the time was heading the XFree86 freeware effort, explained to the author - when he asked why the XFree86 servers were of such poor quality - that the effort was to quickly get "something" out here, and then worry about quality later. In a nutshell, that is the explanation of how the hole got created in which the Linux graphics driver developers at

ATI, Matrox, Nvidia, et al, now find themselves: the X servers to which they are designing their Linux drivers are the open source servers, and the open source servers are not of commercial quality, are poorly architected and implemented, and have been thoroughly hacked over the years in attempts to add the capability to support more modern graphics architectures.

The open source server developers did not (and do not) have to respond to customers, worry about cost of support of delivered product, consider long term implications to the "bottom line," and worry about besmirching the name of the organization. Most contributors to the freeware X server effort provided their time for free - a mention of their contribution was reward enough - (although often the contributor was on the payroll of others, so did not have to go hungry while contributing software development for free). And, there was no real single point of control (in the guidance context), so the total effort was sort of a free-for-all (pun not intended).

The overall result is that the state of the [open source] Linux graphics, as described by Smirl[JS] is pretty dismal. So bad, in fact, that there is a push by some in the open source community to toss X and start over from scratch to design another system based on OpenGL from the ground up. Even Jim Gettys, one of the original members of the Athena team at MIT where X had its beginnings about 1984, indicated recently [JG] that version X11 released in 1987, included "Inadequate 2D graphics, which had always been intended to be augmented and/or replaced".

Well, ten years ago, Xi Graphics did just that, within the X specifications, of course, and has produced strong 2D graphics performance for over ten years now. We didn't use much of the SI implementation - we did our own. And over the years we have continued to bring the capabilities of the X server along to keep up with the technical advances and capabilities of the graphics hardware. The open source X servers suffer greatly in this respect, and the Linux developers at the graphics chip manufacturers are (probably) acutely aware of this. It is not X that needs to be tossed; it is the open source X servers that should be tossed. If twenty years of open source development has botched the current open source X server effort, who is to believe that the open source movement will be able to do better at developing a replacement for X based on OpenGL? Which gets us to another part of the X server - OpenGL and the GLX extension.

Accelerated-X and OpenGL

Xi Graphics delivered its first OpenGL 3D product in 1999, and it had support for over 30 graphics cards from a number of graphic chip manufacturers. The OpenGL pipeline provided in the product was developed "from scratch" by Xi Graphics, and the Accelerated-X servers were upgraded to provide smooth integration of the GLX extension and to maintain for 3D rendering the same high standards maintained for 2D-only products. In the intervening years, both the OpenGL pipeline

and the X servers have been extended to add features and capabilities in both. When first introduced, our pipeline supported OpenGL v1.1, which was upgraded to v1.2 by Sept '01, and a year later to v 1.3 (with most of the requirements for v1.4, including GLX v1.4 with Pbuffer support). Currently v1.5 is fully supported with some later extensions. Figure 3 depicts OpenGL and X set up for indirect rendering, with GLX capabilities in libGL at both ends.

Unlike new releases of OpenGL software by X.org, when Xi Graphics releases upgraded OpenGL support, it has been tested on a large number of graphics cards/chips (we have a huge inventory of graphics cards and motherboards), many versions of UNIX and Linux kernels, a bunch of x86 platforms, and perhaps a SPARC and/or PA RISC platform. And, the graphics chip support starts with the newer graphics hardware, not the oldest, as is often the case with the open source server/drivers. While this may sound like a massive undertaking to someone well versed in the efforts expended generating open source X and OpenGL software for (just) Linux kernels, it actually is done by a small team, thanks to the architecture and design of the underlying Accelerated-X graphics sub-system for X and OpenGL, and to the fact that there is a business-oriented organization and hierachy guiding the effort.

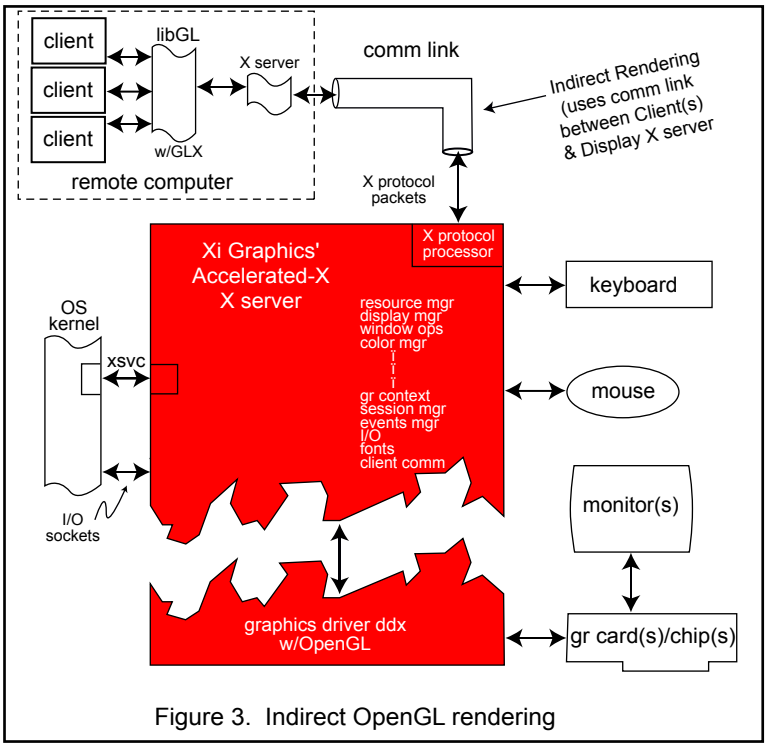

Figure 3. Indirect OpenGL rendering

Direct Rendering With OpenGL

The X.org OpenGL efforts are severely handicapped by the approach taken to implement direct rendering of OpenGL when both the client and X server are on the same computer. The open source approach is "marketed" as "Direct Rendering Infrastructure" (DRI), and "Direct Rendering Manager" (DRM). Architecturally, it is a real mess, resulting in a complicated structure that requires separate drivers be built for each Linux kernel version for each graphics chip type. Each time the Linux group changes the kernel a bit (which now happens frequently), the DRI requires a change. UGH! Not only that, moving to a UNIX kernel such as Solaris, HP/UX, etc., also requires significant changes.

Xi Graphics provides direct rendering capability in Accelerated-X OpenGL. Indirect rendering (which uses GLX lib error checking) is a much "safer" path for OpenGL on X, but it is slower because it is designed to use the communications path that is required when OpenGL applications are run on one machine, say in Boulder, Colorado, and the graphical results displayed on another computer, say in Washington, DC. But if our customers in Boulder want to run in direct mode, they can bypass the GLX X packets and comm link and run in direct rendering mode, and obtain a significant increase in speed. With indirect rendering (as is depicted in Figure 3), there are "guard rails" along the road, in the form of error checking of the OpenGL commands when using the GLX lib and the X Protocol mechanism. These guard rails are removed when direct rendering is selected, so one can "run off a cliff" much more easily, meaning that with direct rendering, the application has a good deal of direct control of the graphics hardware, and can royally screw up the operations with erroneous commands.

Xi Graphics adopts the philosophy that the OpenGL applications will be written correctly, and will run without problems in direct mode on a well designed system. To assume otherwise, one would have to do extensive error checking and erect complicated barriers in the graphics sub-system in an attempt to catch all (or even most) application errors and prevent them from causing system problems. Because OpenGL is very complicated, it would be impossible to completely insulate the sub-system from apps errors, and any serious attempt to do so would defeat the propose of using direct rendering in the first place - faster operation. Extensive error checking causes poor performance, so one is reduced to chasing one's own tail. If an application causes trouble in direct mode, running it in indirect mode usually allows the application developer to quickly find the problem(s) and make corrections.

The freeware X.org OpenGL server/driver developers have taken the other approach, and attempt to prevent faulty OpenGL applications from doing harm to the graphics sub-system. Not only is the performance poor as a result, the "Direct Rendering Infrastructure" and "Direct Rendering Manager" used to replace the "guard rails" in GLX causes a separate graphics driver to be required for most Linux kernel versions, and most versions of the X server core for each chip supported.

With the Linux kernels changing at a very rapid clip nowadays, as are the X server releases from X.org, this "save the system from inept OpenGL developers" philosophy causes everyone else lots of grief, it seems. But, hey, it's free software! That is, it's free if the cost of initially getting it to work in a product and the on-going updates and bug fixes due to changes in the underlying OS kernel and/or the clients over the life of the product is not considered.

Direct Rendering in Accelerated-X

Figure 4 depicts direct rendering with Xi Graphics' approach. Notice that there is still only one kernel driver - the "X Services Module," or xsvc. It is the same one as is used for 2D X only rendering. The client(s) are on the same computer as the display server, so the X Protocol link can be eliminated. When direct rendering mode is used, an OpenGL application is "allowed" to bypass the X server for many operations and control the graphics hardware directly. As noted earlier, the guard rails come down, as do the road signs warning of danger. Xi Graphics assumes the OpenGL applications developers know what they are doing and how to find and correct errors. (Using indirect mode is very helpful in this regard). We did not design the direct mode to be idiot proof. On the contrary, we designed it for the competent.

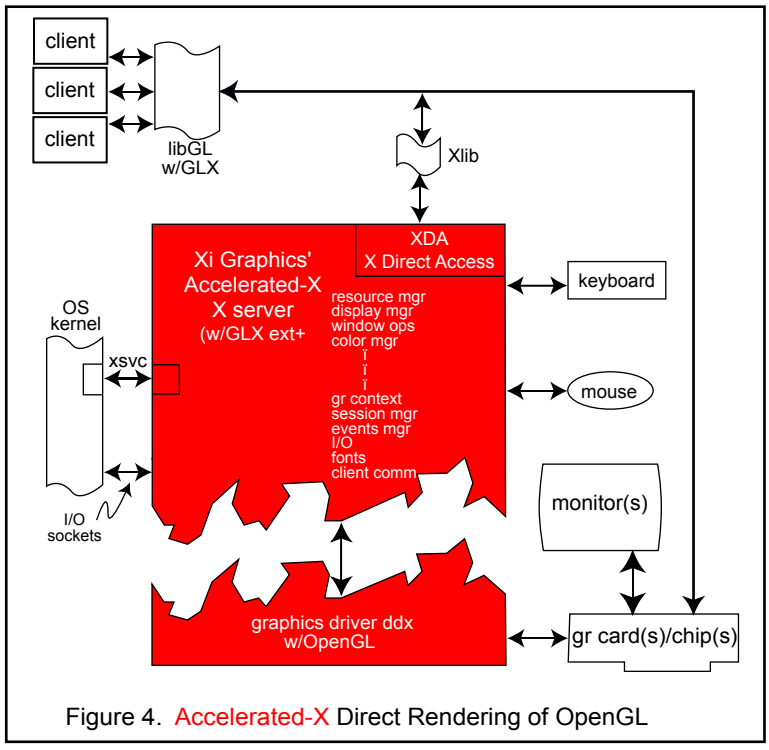

Figure 4. Accelerated-X Direct Rendering of OpenGL

The necessary coordination of states and exclusivity of resources and other "housekeeping" chores caused by the dual access to the graphics hardware is managed by the XDA module. So, the result is clean, and requires minimal changes to existing structure to accomodate direct OpenGL rendering, and is quite fast. For Xi Graphics' benefit, the straightforward (simple, actually) approach means that supporting various OS and OS versions, and different graphics hardware is pretty much routine. Contrast this with the open source approach taken by XFree86/X.org.

Open Source Direct Rendering Infrastructure - DRI

Quoting one of the key developers of DRI, (italics added)

> "*The DRI is not a single, isolated piece of software. Instead, the DRI is composed of a number of distinct modules. The following briefly describes those modules and where they fit into a Linux system.*"

First he describes the kernel modules.

> "*For each 3D hardware driver there is a kernel module. This module deals with DMA, AGP memory management, resource locking, and secure hardware access. In order to support multiple, simultaneous 3D applications the 3D graphics hardware*

must be treated as a shared resource. Locking is required to provide mutual exclusion. DMA transfers and the AGP interface are used to send buffers of graphics commands to the hardware. Finally, there must be security to prevent out-of-control clients from crashing the hardware. ... Since internal Linux kernel interfaces and data structures may be changed at any time, DRI kernel modules must be specially compiled for a particular kernel version." [BP]

Why DRI kernel Modules?

Pushing things into the kernel can cause bad things to happen. Especially in the "Linux World." The Linux kernel seems to have a new revision about once a month. Changes are made that "break" existing systems. One almost gets the feeling that breaking existing systems is "a good thing" if what breaks is some binary code from a "closed source." That religious thing again.

Frequent kernel changes effecting many kernel drivers and application modules mean that the maintainers of those drivers and modules have a difficult time making their stuff work on the many kernels floating around in the Linux space. Just ask Nvidia, ATI, et al. So when the DRI developers say that "*Since internal Linux kernal interfaces and data structures may be changed at any time, DRI kernel modules must be specially compiled for a particular kernel version*", one is left shaking one's head.

The argument that because the source is always available in the "pure" Linux World, the mismatch of kernels and modules is no big deal - "just pull all of the correct modules together and recompile" - just doesn't seem to make sense in the Real World. Plug and Play would seem to be a lot more desirable.

And another thing. This matter of "security" being one of the "objectives" of DRI: "... *there must be security to prevent out-of-control clients from crashing the hardware.*" In order to get this "security," the Direct Rendering Manager (DRM) - the module that performs similar functions to Accelerated–X's XDA module - must be put into the kernel. Huh? Not only that, packets are still required by DRI; they just don't have to go over a comm link. But they are generated, and they are then "analized" by the DRM - in the kernel - in an effort to make the system "secure" against out-of-control OpenGL applications. Xi Graphics thinks this is an impossible goal to begin with, and discarded such ideas, but the XFree86 community disagreed and proceeded to create a very large processing overhead in the attempt at "security." So direct rendering in XFree/X.org systems is not so fast, to say the least. But wasn't direct rendering of OpenGL supposed to make things go faster? One can get a headache from shaking one's head at all of this. Figure 5 is an attempt at depicting an XFree86/X.org based Linux system. It ain't easy.

Kernel-based DRM

The DRM is responsible for a lot of things in the DRI. It, rather than libGL, sends commands to the graphics card(s)/chip(s). This is after it analizes the DRI packet contents generated from the commands from libGL to ensure the "security" of the system by not letting "out-of-control clients crash the hardware". With multiple clients, the DRM is responsible for maintaining context coherency of the graphics engine(s) with the clients, does context switching (galore), reports to clients X events, and so on. It is a real busy module. And it is doing all of this *in the kernel*. If it hiccups, it just might actually bring down the whole system. Imagine that - the graphics system bringing down everything. That seems to be a strange way of building a system that is protected from graphics software screwups.

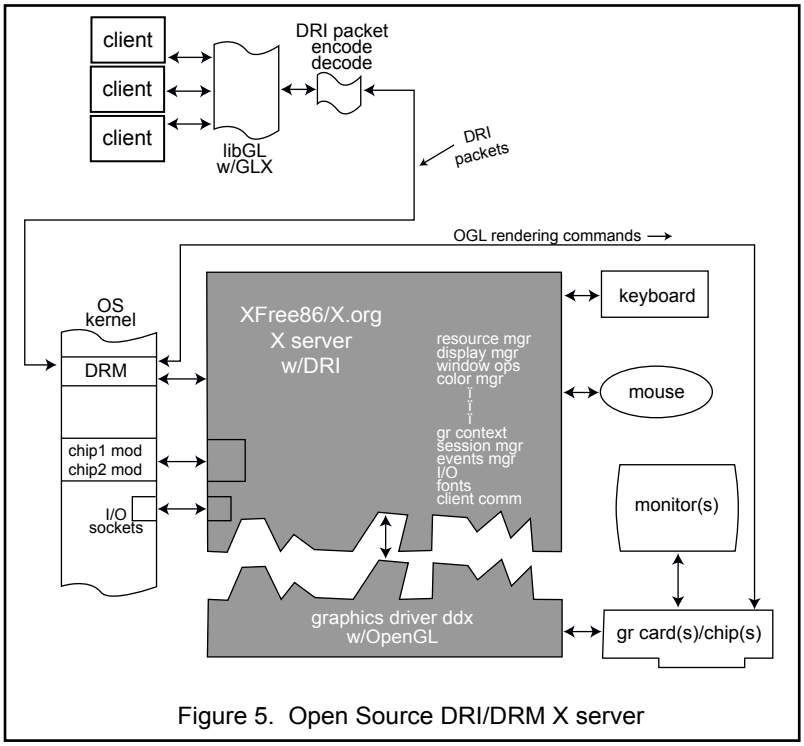

Figure 5. Open Source DRI/DRM X server

"For each 3D hardware driver there is a kernel module." Let's see, now. Only one kernel driver is needed for each type of graphics chip. One for Matrox G400, one for Matrox Parhelia, for ATI RADEON R200, for ATI R300, etc. Not too bad, it seems, until one asks why is there a kernel module per chip type? Xi Graphics supports a lot of graphics chip types, and does not have a single kernel module specific to any one of them. That sure saves us a lot of work! Especially when one considers that the Linux kernel is changing at a fast pace. Those poor open source graphics folks, having to maintain all of this kernel stuff. Hope they can find time to get in a little fishing and snowboarding.

With all of this complicated DRI stuff going on, one might be concerned with the stability of the X Window System, on Linux particularly, especially when running OpenGL clients. But knowing how much of the system is actually running in the kernel, one might get the willies. The kernel would seem to be spending an awful lot of time doing graphics, instead of doing real UNIX kernel stuff, while graphics are done in user space - where the kernel can protect the system from those out-of-control OpenGL applications. Product and Program Managers who would like to avoid such things should seriously condider using only commercial (closed source) graphics sub-systems on UNIX - and that includes Linux - systems.

--0--

[JS] "The State of Linux Graphics" August 30, 2005 by Jon Smirl
http://jonsmirl.googlepages.com/graphics.html

[JG] "The (Re)Architecture of the X Window System" June 15, 2004
by James Gettys and Keith Packard
HP Cambridge Research Laboratory

[BP] "Introduction to the Direct Rendering Infrastructure" by Brian Paul
August 10., 2000