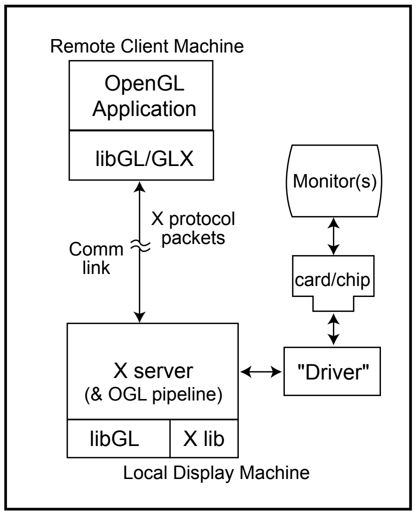
Having Problems w/Linux (freeware) Graphics?

The past few years has seen the popularity of Linux increase substantially. While Linux earned its reputation for stable, reliable reputation in "headless" (no graphics hardware) applications such as running Apache Web servers, it has since moved into more mainstream applicaions that often are graphics intensive. That reputation for stable, reliable operation has not followed. Linux systems with extensive and/or demanding graphics requirements have numerous problems if the graphics sub-system software is dependent upon XFree86/X.org (freeware) X servers.

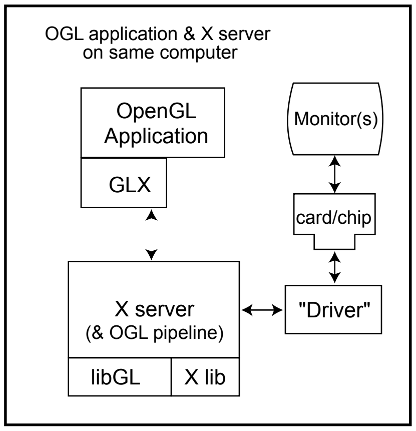
Linux users not familar with the details of graphics software often believe that the Linux operating system is made by the Linux group, and the graphics driver is provided by the graphics card manufacturer (the Microsoft Model). Users who are familiar with the details know that Linux is a (UNIX-like) kernel, the X Window System ("X") is the graphics portion of a UNIX system using graphics, and the graphics driver for a particular card is a (relatively small) part of the X graphics sub-system software. They also know that the internals of the graphics sub-system software is radically different between Microsoft systems and UNIX systems. In fact, MS OS architecture is radically different from UNIX OS architecture. This may account for some of the initial difficulties encountered by "Windows only" SysAdmins when moving to Linux. UNIX and Windows really are different animals.

However, battle-hardened UNIX users moving off Solaris or AIX or HP/UX have also experienced difficulties when moving to Linux for systems that have heavy graphics requirements. This paper is an attempt to shed some light on some of the causes for such difficulties, and to lead the readers to try Xi Graphics' Accelerated-X™ brand of UNIX/Linux graphics sub-sytem software.

In the figure to the right is a simplified diagram of a UNIX system displaying on one computer OpenGL graphics images specified/created by another (remote) UNIX system computer. This is referred to as a "remote client" configuration, where the "client" is the application, and the graphics display computer is the "server" system. The two computers communicate with each other via "X Protocol Packets" containing queries and commands from the client, and answers and data to the client. For graphics intensive displays, the comm link can be a source of slow system performance, since OpenGL will be issuing large numbers of drawing commands in the process of making images.   When the client (applications) program can be on the same computer as the display server - as shown in the next



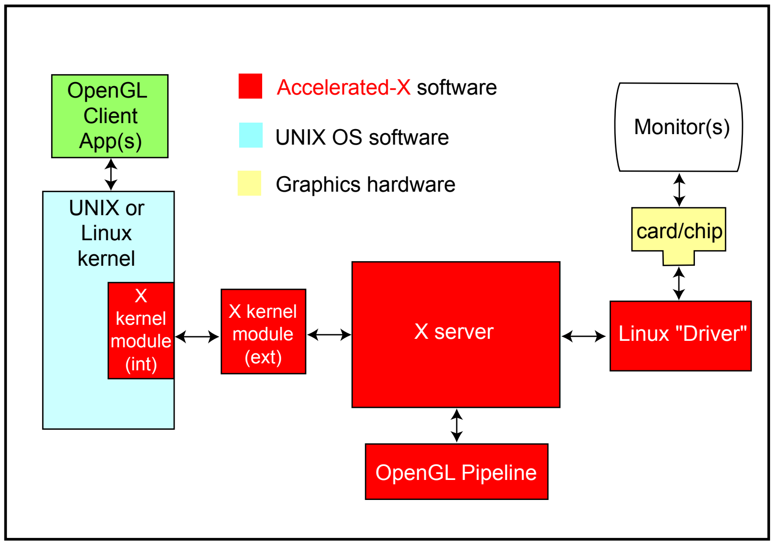
Typical UNIX Remote-client OpenGL Display Configuration figure to the right - this comm link and the X Protocol packet encode/decode logic can be bypassed, greatly speeding up OpenGL (and other related) operations. Notice that one of the OpenGL libraries is eliminated, since both the client and server side can use the same libGL. A GLX "glue code" is still required on the client side. This code allows the OpenGL application to use X, and the X server will have the GLX extension implemented in its logic.


OGL application & X server on same computer
OpenGL Application
GLX
Monitor(s)
card/chip
X server (& OGL pipeline)
"Driver"
libGL
X lib

Typical UNIX Direct-rendering OpenGL Display Configuration

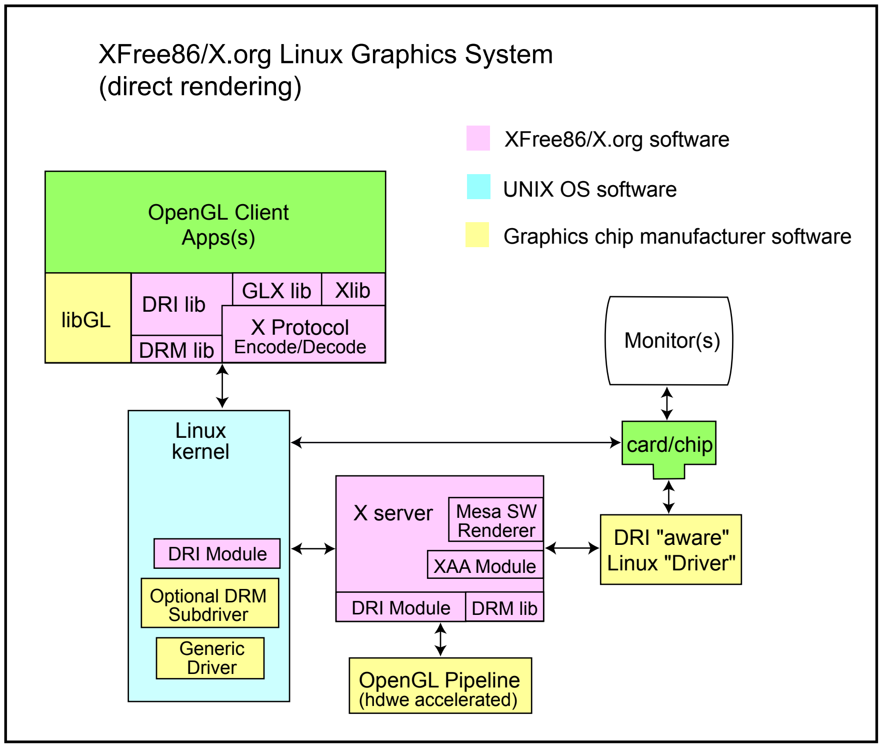
A system with applications and display server present on the same computer can still operate in indirect mode, using the X Protocol packets. This mode of operation provides error checking of commands as the packets are generated and processed, which is a useful diagnostic tool if the direct mode is suspected of generating incorrect commands or sequences of commands. Operating in direct mode removes some of the "guard rails" (command error checking) present in the indirect mode, allowing errant OpenGL code to screw up the X sub-system. Runing the suspect applications in indirect mode often uncovers the OpenGL code errors.

When a Linux or UNIX system is assembled that uses Xi Graphics' Accelerated-X brand of (commercial, high-quality) X Window System software for grahical display, the system is similar to that shown in the figure below. Xi Graphics develops its own X servers and drivers (ddx's) and OpenGL rendering pipeline, depicted in red. We also make UNIX and Linux kernel modules, we call "xsvc modules" that interface the X server to the kernel for initialization, shut-down, and run-time resource management. Part of the kernel module is inside the kernel; the rest is outside. Xi Graphics adheres to the UNIX principle that applications are outside the kernel, to the maximim extent possible, so the kernel bit that is inside the kernel is quite small - about 160KB.   Note that there is no XFree86/X.org driver, server, or OGL rendering code used.



Typical Accelerated-X™ OpenGL Graphics System

The figure at the right depicts an OpenGL system that uses XFree86/X.org X server and kernel modules, a driver and OpenGL rendering pipeline from a graphics chip manufacturer, and a Linux kernel from the Linux Group, all set up for direct rendering. XFree86/X.org uses what is termed the Direct Rendering Infrastructure ("DRI") and the associated Direct Rendering Module ("DRM") to implement direct OpenGL rendering when the OpenGL applications are resident on the X display machine.



The DRI mechanism seems to be overly complicated and involved for what is basically a method of eliminating the X packet protocol link and most of the X server involvement in order to enable fast operation of OpenGL image generation.   Essentially, the X server is bypassed by OpenGL commands sent to direct to the graphics hardware. X is used to set up and control the windows and other housekeeping functions, but is then not involved with the bulk of OpenGL rendering operations. This speeds up OpenGL rendering tremendously.

The DRI architecture, however, extracts quite a price for increased OpenGL performance. An assumption was made by the DRI developers that eliminating the X protocol packet checking - by doing away with the use of the packets for OpenGL commands - was not a good thing. Apparently there was concern that errant OpenGL commands could crash the grahics card or graphics software (which is true), and this must not be allowed to happen. To address the issue, DRI still requires OpenGL packets (special to DRI), and they are checked by a module inside the kernel. If no problems are found, the commands are sent on to the graphics card. On the other hand, it is an impossible task to catch every OpenGL command error or to uncover every nasty scheme to screw up the display or crash the system, so keeping the packets and moving the checking operation into the kernel seems self-defeating.

Proponents of the DRI scheme claim that checking the validity of the OpenGL packets is a security requirement, so that a client cannot cause a machine crash. Uh, in direct mode? The client in on the same machine as the OS (the system). If the user is not to be trusted, don't let him on the machine! The security argument seems a bit thin. Especially when the complication caused by DRI involves the kernel in a lot of work that rightly belongs in aplications space. DRI is an example of the XFree86/X.org, along with the folks at the Linux group, violating a basic UNIX principle - the one about user programs (applications) do not reside in kernel space. OpenGL and X are applicaitons.

DRI breaks up the graphics driver software into two parts, with one part in the kernel. The "DRI aware" ddx is required to required to have involvement with the kernel. More complication. The graphics card is getting commands from more than one source, and the sources are not coordinated, except through the kernel.

It appears that the complications added to the UNIX/Linux kernel and to the graphics sub-system software by the DRI architecture addresses a non-issue (security of a direct-rendering machine). The side effects are reduced stability of the system (adding graphics applications logic inside the kernel), added cost to the graphics chip manufacturers of developing and maintaining graphics driver software, and vunerability of the entire system to frequent kernel changes emanating from the Linux group.

The idea of pulling more and more applications code into the kernel seems to be a foolish move. When things go wrong inside the kernel, bad things happen. Graphics code is large and complicated, so bad things will happen. When the bad things are happening in applications space, the kernel (at least a well behaved one such as those in Solaris OSs) can usually shut down the out-of-control applicaiton and protect the rest of the system. When those bad things are happening inside kernel space, really bad things happen, and the entire system is often lost.

Xi Graphics installs about 160KB of its graphcis software for Linux in the kernel, including tables and other static data. ATI and Nvidia install 2MB to 3MB or more in the kernel, a large amount of it executing code, and highly dependent upon the version of the particular kernel running. A slight change to the Linux kernel - a frequent occurrence - and the ATI or Nvidia kernel code will often require reworking. So is it any surprise that "Linux graphics drivers" seem to have such problems?

The basic design of the XFree86/X.org X server that the graphics chip manufacturers use with Linux is a structural mess, the code is written by many "contrubutors" with varying levels of architectural and development skill in graphics software, and the whole thing managed by ...?

Well, managed may be a wrong description. With Linus echewing the use of specifications for the Linux kernel(s) and his group producing kernel changes at a breakneck pace, with the XFree86/Xorg X servers in a such a sorry state, and with the graphics chip manufacturers designing ever more capable (and complicated) hardware for which they must write graphics "drivers" to hook up with the XFree86/X.org X servers and kernel modules that keep changing rapidly, maybe the situation is just not manageable.

Xi Graphics has probably designed more commercial UNIX/Linux graphics drivers and X servers for use with more graphics chips, than any other organization in the World, and we would not want to have to manage the mess made by the XFree86/X.org community. Instead, we develop a unified, coherent, commercial-quality, set of X Window System sub-system software products that operate on various UNIX kernels, and on numerous computer platforms, running graphics hardware manufactured by several graphics chip manufacturers.   We have been doing it for over ten years, and have licensed the software to countless individuals and organization for use in applications that cover the spectrum. All the while competing with that "free software." How do we do it, you ask? Well, Xi Graphics exists because some folks have learned that "free software" can be very costly. And others have learned that "expensive, licensed proprietary software" is actually very economical software when one puts a value on easy installation, stable operation, speedy performance, free customer support, lack of stalled production lines caused by obscure graphics software bugs, and a vendor who must satisfy customers to stay in business.