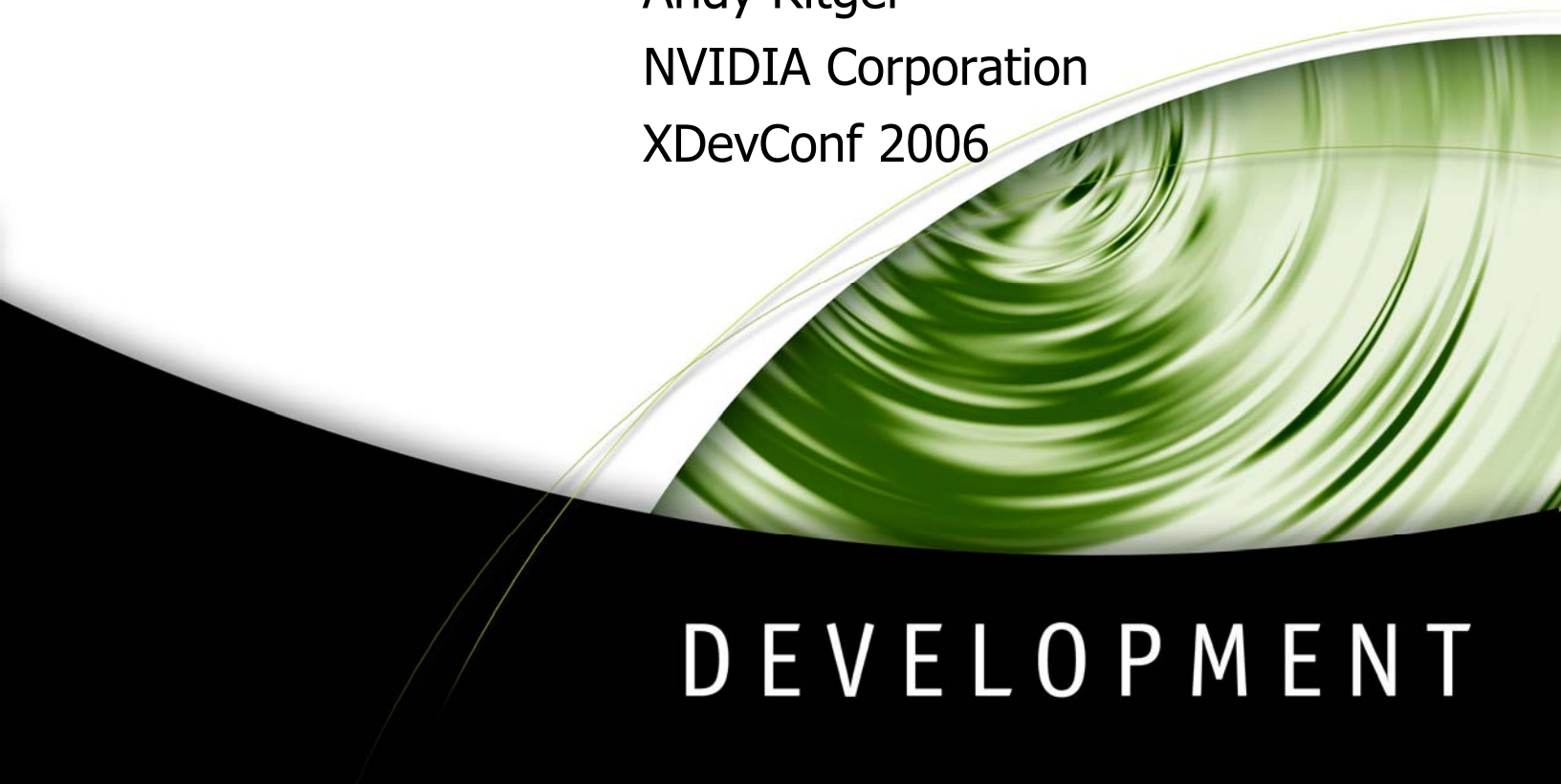
Whitepaper

Using the Existing XFree86/X.Org Loadable Driver Framework to Achieve a Composited X Desktop

Andy Ritger
NVIDIA Corporation
XDevConf 2006



Abstract

In this paper, we make the case for using the existing XFree86/X.Org DDX loadable driver framework to achieve a production-quality composited X desktop, as opposed to the X-on-OpenGL model. While the X-on-OpenGL model demonstrates what the graphics hardware is capable of, everything that the X-on-OpenGL model can achieve is equally possible with the current framework. Furthermore, the current framework offers flexibility to driver developers to expose vendor-specific features that may not be possible through the X-on-OpenGL model.

Andy Ritger
aritger@nvidia.com

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050

February 8, 2006

Using the XFree86/X.Org Loadable Driver Framework to Achieve a Composited X Desktop

There is increasing interest in delivering a composited X desktop, and debate about what driver model can best accelerate the compositing operations. Two candidate models are: 1) the existing XFree86/X.Org DDX loadable driver framework, and 2) a replacement DDX that uses OpenGL for its rendering and modesetting.

Candidate 1, the XFree86/X.Org DDX loadable driver framework, has been in use since the release of XFree86 4.0 in March 2000, and is what both the XFree86 and X.Org X servers use in shipping operating systems. In this framework, a hardware-specific driver is loaded and performs the tasks of rendering and modesetting. The framework does not dictate how the loaded driver performs these tasks.

Candidate 2, the X-on-OpenGL model, eliminates hardware-specific X drivers, and instead uses OpenGL to perform all X rendering. The X server, as proposed in this model, also uses the OpenGL driver to perform modesetting, through the EGL_MESA_screen_surface [1] extension.

In this paper we provide background on the technology that enables a composited X desktop, and identify goals related to a composited desktop for the X windows community. We then compare the two driver models, and make the case that the existing loadable driver model is better suited for achieving our goals. Finally, we end with a roadmap for how to achieve a production-quality composited X desktop.

Background

A composited desktop is a window system where windows are rendered unclipped off-screen and composited together with alpha blending to give the effect of translucency. Such a system has been a goal of the X windows community for some time. The Render[2], Damage[3], and Composite[4] X extensions provide the tools necessary to create such a composited X desktop.

The Render extension introduces Porter-Duff-style image compositing operations [5] to the X window system. In modern X desktops it is used, for example, to render anti-aliased text.

The Damage extension gives X clients a mechanism to request notification when a window or windows have been modified.

The Composite extension introduces the ability to "redirect" an X window subtree to off-screen memory. This means that all rendering that is done to that window and its children are done, unclipped, to an off-screen location.

A "composite manager", which may or may not be the same application as the X window manager, can choose to redirect certain window subtrees to off-screen memory using the Composite extension. It can use the Damage extension to be notified when that subtree is rendered to, and then use whatever means it prefers, typically the Render extension, to merge that subtree back into the visible desktop, performing blending and transformations along the way. The details are explained more fully in Keith Packard's description [6].

An important benefit of this desktop compositing architecture is that it is an incremental enhancement to X windows, and will be transparent to existing X clients. Another important benefit is that the compositing is optional. Users can choose between the benefits of a composited desktop, and the existing window system without the overhead of compositing. This flexibility is particularly important for users of direct-rendering OpenGL where the compositing overhead may be most noticeable. The compositing overhead is due to the extra copy that must be performed to blend the redirected windows back into the visible desktop. This may be more noticeable for direct-rendering OpenGL applications due to the needed additional synchronization.

Currently, the Render extension is in production, and has been for several years. The Damage extension has been in production since X11R6.8.0 (09/09/2004). The Composite extension implementation was deemed too immature by the X.Org group at the time of X11R6.8.0 release and was turned off by default, though it can be optionally enabled as an experimental feature. While much work has been done recently to stabilize Composite, it has not yet been enabled by default in the X.Org X server, as of the X11R7.0 release (12/21/2005).

Goals

For purposes of this discussion, we have identified the following goals for the X window system -- some are drawn from Keith Packard's *Getting X Off The Hardware* [7], some are based on the interests of independent hardware vendors (IHVs) such as NVIDIA, and some appear to be general trends in the X windows community. We will consider these goals when comparing candidate driver models for use in accelerating a composited X desktop throughout this paper.

The goals are:

1. Bring a compelling composited X window system to the UNIX desktop.

2. Give window manager and composite manager authors the power and flexibility to explore new realms of user interfaces.
3. Maintain application backward compatibility.
4. Improve interaction between X and UNIX kernels, particularly in the area of PCI device management.
5. Make relevant X rendering perform optimally and make best possible use of the available graphics hardware.
6. Continue to support existing advanced functionality, such as hardware-accelerated direct-rendering OpenGL.
7. Grant vendors the flexibility to expose vendor-specific features such as TwinView/MergedFB, Quad-Buffered Stereo, SLI, and FrameLock.
8. Give users the flexibility to choose for themselves when they want a composited desktop and when they want full performance OpenGL or features like Workstation Overlays that may not be compatible with Composite.
9. Bring the functionality of Damage and Composite to production quality in the near future, so that it can be shipped and enabled by default by operating system vendors.

Comparison of Driver Models

The current loadable driver framework present in X.Org provides a high level, flexible interface between the DDX and the loadable hardware-specific drivers, described in detail in [8]. It imposes minimal restrictions on how a driver performs rendering and modesetting.

With this flexible infrastructure, vendors have been able to deliver a wide range of powerful features including direct-rendering hardware-accelerated OpenGL (e.g., DRI [9], NVIDIA [10], and ATI [11]), TwinView/MergedFB (e.g., NVIDIA [10], ATI [11], SiS Driver [12], Radeon Driver [13], Matrox Driver [14]), Quad-Buffered Stereo (e.g., NVIDIA [10], ATI [11]), Workstation Overlays (e.g., NVIDIA [10], TechSource [15]), and SLI (NVIDIA [10]). For more examples, see the list of features that the NVIDIA driver provides described in [16].

While the loadable driver framework is very flexible, one notable complaint has been the lack of hardware-acceleration of the Render extension. Due in large part to frustration with poor Render hardware-acceleration, some have advocated replacing the existing X.Org DDX loadable driver infrastructure with a DDX that implements all of X's rendering operations with calls into a stand-alone OpenGL driver ([7] and [17]). Arguments for an X server running on top of OpenGL, hereafter referred to as "X-on-OpenGL", have included "it will make Render faster", and "this will make vendors' lives easier." The arguments for the X-on-OpenGL model will be addressed more thoroughly below in the section *Arguments for, and Rebuttals against, X-on-OpenGL.*

An implementation of the X-on-OpenGL model, Xgl [18], is in progress and has been successfully demonstrated on several occasions [19]. Additionally, a sort of forerunner to Xgl, glitz [20], demonstrated that Render can be well accelerated using the OpenGL API.

In the X-on-OpenGL model, there would be a hardware-specific OpenGL implementation, but no hardware-specific X driver. The traditional tasks of the X driver, rendering and modesetting, would be handled largely by the common X-on-OpenGL DDX. The X-on-OpenGL DDX would call into the hardware-specific OpenGL driver to perform rendering; on hardware for which no accelerated OpenGL is available, Mesa software OpenGL rendering would be used. X-on-OpenGL advocates propose that OpenGL implementations provide the EGL_MESA_screen_surface [1] extension, which the X-on-OpenGL DDX would call to perform modesetting.

In the following we evaluate how well each of the aforementioned goals can be achieved by each of the two driver models.

Goals 1, 2, and 3

Goals 1, 2, and 3 are already implicitly accomplished with either driver model, thanks to the design of the Damage and Composite extensions: these extensions give us the tools to create a composited X desktop (Goal 1), they place the policy decisions of how compositing should be done in the hands of composite manager authors (Goal 2), and they are transparent to current X applications, achieving backwards compatibility (Goal 3).

Goal 4

Improving PCI configuration interaction between the X server and the UNIX kernels (Goal 4) is independent of either driver model, and work is already in progress in this area [21]. This work will support Goal 4 equally for both driver models.

Goal 5

For the purposes of highest performance and making best possible use of the available graphics hardware, both an X driver within the existing framework and an OpenGL driver have access to the same hardware capabilities. Additionally, an X driver has more context and information about the requested X rendering than an OpenGL driver used by X-on-OpenGL, and thus should be in a better position to make intelligent tradeoffs when necessary for optimal performance.

It has been argued that X-on-OpenGL will lead to dramatic performance improvements over the existing loadable driver framework [7]. This might be true in the short term. However, this is not because the existing driver framework has prevented better hardware-acceleration. Rather, Render support has historically been poorly accelerated due to lack of attention and an out-dated acceleration architecture within the X server (XAA [22]). This is changing, though:

- The new EXA [23] acceleration architecture provides a path for drivers to support acceleration of Render.
- NVIDIA is taking Render acceleration seriously: in our upcoming nvr85-series drivers, many improvements and stability issues have been resolved in our Render acceleration

support, and our Render acceleration has been enabled by default. Much more tuning is possible and will be phased in over the next several releases.

The existing loadable driver framework is at least as capable of achieving goal 5 as is the X-on-OpenGL model.

Goals 6 and 7

Goals 6 and 7 can be considered together; they both focus on giving vendors the flexibility to support features like OpenGL, TwinView/MergedFB, Quad-Buffered Stereo, Workstation Overlays, etc. Features such as these are possible within the current loadable driver framework; this is evident from the earlier mentioned examples of vendors who have provided these features. However, the X-on-OpenGL model poses some difficult problems for supporting existing and future advanced features.

For example, consider hardware-accelerated direct-rendering OpenGL. To support a direct-rendering client within X, a server-side component must coordinate with the direct-rendering client library to:

- Propagate data from the server to the client about the drawable's geometry, clips, and other attributes.
- Manage synchronization between the server and the client such that the client's rendering arrives in the correct place at the correct time.

Please refer to [16] for more details on how a direct-rendering client interacts with the X window system.

With the X-on-OpenGL model there would be no X driver, just a stand-alone OpenGL implementation. It is not clear in this model how data propagation or synchronization would be managed, or who would manage it.

Using the same OpenGL library from both OpenGL clients and the X-on-OpenGL server may impose undesirable requirements on the OpenGL implementer. For example, special communication between the X-on-OpenGL DDX and the OpenGL implementation would be necessary for the following reasons:

- The OpenGL implementation must determine if it is running within the X server or within the OpenGL client.
- If the OpenGL implementation is running within the X server, the OpenGL implementation must manage the data propagation to and synchronization with the instance of the OpenGL library running in the direct-rendering OpenGL client applications.

How would vendors provide vendor-specific features like TwinView, FrameLock, Quad-Buffered Stereo, and Workstation Overlays, within the X-on-OpenGL model? Features such as these largely depend on coordination between the X server and the OpenGL client library.

In the best case, providing the same level of functionality as is available with current NVIDIA drivers will require a major effort to re-implement large portions of the support directly in Xgl, and negotiate numerous backdoors between Xgl and the stand-alone OpenGL driver. In the worst case, some of the features provided by existing drivers, like Workstation Overlays and Quad-Buffered Stereo, may not be possible at all with Xgl.

We believe that the existing loadable driver model can achieve Goals 6 and 7; we believe that the X-on-OpenGL model cannot adequately achieve Goals 6 and 7.

Goal 8

Goal 8 is to give users the flexibility to choose for themselves when they want a composited desktop and when they want features that may not be compatible with a composited desktop. This flexibility is contingent on the ability to have those features in the first place (Goals 6 and 7). Because the X-on-OpenGL model is not conducive to vendors providing these advanced features, Goal 8 is not achievable with the X-on-OpenGL model. However, the architecture of Damage and Composite is flexible, and we believe that within the existing loadable driver framework we can achieve Goal 8: users should be able to disable Composite and have the same performance and functionality as they have today.

Goal 9

Goal 9 is to bring the composited X desktop technology to market in a timely fashion. To achieve feature parity with the current available drivers, the X-on-OpenGL model would require a huge investment of time and engineering resources. And even then, not all features would be possible. However the existing loadable driver framework requires only minor incremental work to achieve our goals. This seems to be the simplest path, and the one to most quickly yield a production-quality solution. In the below section *Future Directions for the Loadable Driver Framework*, we present a roadmap for how to achieve this.

We feel that everything that can be done with X-on-OpenGL can be done with the existing loadable driver framework. The existing framework is the better suited driver model to achieve a composited X desktop because it is an incremental enhancement to the existing system, yet it is flexible enough to allow vendors to provide the additional features that are important to many UNIX users.

Arguments for, and Rebuttals against, X-on-OpenGL

The below is a collection of arguments in favor of the X-on-OpenGL model and misconceptions that have surrounded the discussion of how to best achieve a fully-accelerated composited X desktop. Each is followed by a rebuttal.

Argument: "Using accelerated OpenGL drivers will provide dramatic performance improvements for important operations now ill-supported in existing X drivers." [7]

Rebuttal: This is true, today. However, as noted earlier in our discussion of Goal 5, an X driver within the existing loadable driver framework has access to all of the hardware capabilities that an OpenGL driver has, and therefore should be able to perform at least as well as X-on-OpenGL on the same hardware.

Argument: Having X use OpenGL for rendering and thus eliminating the custom 2D acceleration code will reduce the development burden, and ease maintainability and code reuse.

Rebuttal: Perhaps. However, this could be done with the existing framework. If there is acceleration code that makes sense to be leveraged across both an X driver and the corresponding OpenGL driver for the hardware in question, it would be wise to share that code across the two drivers components. Nothing about the existing driver framework precludes that.

Argument: Bringing up new hardware will be easier because X can just leverage the OpenGL driver for all of its rendering.

Rebuttal: This, however, requires that you have an OpenGL driver for the new hardware before you can start an X server. It is generally much easier to do hardware bring-up with just an X server than a full OpenGL driver.

Argument: Proper hardware-acceleration and improved robustness of X requires kernel interaction/support to setup DMA, process interrupts, etc.

Rebuttal: Agreed. However, nothing about the existing framework precludes this. The NVIDIA X driver, for example, uses a kernel module for low-level resource management.

Argument: OpenGL applications cannot work with Composite unless the X server is also using OpenGL for its rendering.

Rebuttal: The OpenGL implementation must closely coordinate with the X server in order to properly render to the redirected window and for Damage notification to be correctly propagated. The coordination is important for this to work, but how the X server performs its rendering is not relevant to this coordination. The NVIDIA driver will support direct-rendering of OpenGL to redirected windows in a future driver release.

How direct-rendering OpenGL accomplishes its rendering is not dependent on how X implements its rendering.

Argument: Using OpenGL for compositing the X desktop requires that the X server use OpenGL for its rendering.

Rebuttal: The composite manager has its choice of what rendering mechanism to use to assemble the desktop from redirected subtrees. Render is one choice and is what xcompmgr uses; core X primitives is another choice. OpenGL is another choice, and has been demonstrated with Luminocity [24], glxcompmgr [25], and Looking Glass [26].

For a composite manager to use OpenGL to perform the compositing today, it could use XGetImage(3X11) to copy the data from the redirected window, and then use that data as an OpenGL texture. In the future, it will be possible to do this much more efficiently through the GLX_EXT_texture_from_pixmap extension -- the X pixmap for the redirected window subtree will be usable directly as an OpenGL texture, avoiding the copy of pixmap data from the X server to OpenGL.

How the composite manager accomplishes its rendering is not dependent on how X implements its rendering.

Argument: "...3D is simply faster than 2D." [17]

Rebuttal: Presumably the author meant that 3D hardware is faster than 2D hardware. This is not true in all cases. It depends on what you are trying to do. Typically, the implementer of an X driver would assess, on a case-by-case basis, how best to use the available hardware to accomplish the requested rendering.

Argument: "At some future point the graphics chip vendors are going to remove that dot labeled 2D and only leave us only [sic] with 3D hardware." [17]

Rebuttal: Perhaps; perhaps not. Graphics chip vendors are going to build hardware that most effectively accomplishes the tasks at hand. One of those tasks is accelerating a modern X desktop. It is the role of the X driver implementer to assess how to optimally perform the necessary operations with the hardware in question.

Argument: X-on-OpenGL will be easier for IHVs, because now an IHV will only need to provide a stand-alone OpenGL driver, rather than both an OpenGL driver and an X driver.

Rebuttal: If the goal were simply to accelerate X rendering, then this might be true. However, the X-on-OpenGL model poses significant challenges for supporting existing and future advanced features, as discussed above in the discussion of Goals 6 and 7. From this perspective, X-on-OpenGL will not make things easier for IHVs.

The work needed to achieve feature parity with existing drivers, if even possible, will also greatly postpone the time to market.

Argument: X-on-OpenGL will be easier for the open source community; if the documentation for modern GPUs is not available then the X community must depend on IHVs to perform the work to build a good Render implementation. With X-on-OpenGL the open source community can implement their own Render driver and the X community need only depend on IHVs to provide a good OpenGL implementation.

Rebuttal: Relying on an IHV for a good OpenGL implementation is a substantial dependence. Implementing a full OpenGL driver is an enormous undertaking; implementing Render acceleration is a much smaller task, especially as the EXA acceleration architecture matures. It seems that if an IHV is willing to put forth the large effort to provide a suitable OpenGL implementation, then the IHV would also be willing to put forth the comparatively small effort to provide Render acceleration.

Furthermore, an IHV committed enough to Linux to provide an OpenGL implementation is going to be interested in having that OpenGL implementation exposed to direct-rendering OpenGL clients, and will likely be interested in also exposing vendor-specific features. As described earlier, X-on-OpenGL impedes those additional goals of an IHV. These impediments may even discourage IHVs from providing an OpenGL driver at all.

Argument: "...graphics vendors really only care about MS Windows so they do the minimum driver support they can get away with for Linux." [17]

Rebuttal: That is certainly not true in the case of NVIDIA.

While X-on-OpenGL may offer better Render performance in the short term, this can and is being addressed within the existing driver framework. X-on-OpenGL may encourage code reuse between the OpenGL driver and the X driver, but this same reuse is possible within the existing framework. Furthermore, the X-on-OpenGL model limits a vendor's ability to provide extensions and additional functionality, effectively reducing all vendors to the lowest common denominator.

For these reasons, we feel that X-on-OpenGL is not an appealing solution to the stated goals, especially given that all of our goals can be achieved with the existing driver framework.

Future Directions for the Loadable Driver Framework

The trend in composite managers seems to be to use OpenGL to perform their compositing, rather than Render. This makes sense because OpenGL provides much more flexibility in how the compositing is performed than Render. OpenGL also exposes GPU programmability and other advanced rendering features of the GPU that a composite manager may wish to use, which Render does not expose.

It is our belief that the best route to deliver a production-quality composited X desktop is to use the existing driver framework for X rendering and, where possible, OpenGL for the composite manager's rendering. If no OpenGL implementation is available, then Render is a suitable fallback.

In order for the composite manager to cleanly use OpenGL for its compositing, there are several issues to resolve:

- We need to resolve how OpenGL can render to the "output window" without being clipped by redirected subtrees; this has been discussed in a recent X.Org email thread [27], and we are confident that this can be resolved.
- We need an efficient means for X pixmaps to be used as textures in OpenGL; Xgl provides an experimental GLX_MESA_render_texture extension to allow indirect-rendering GLX clients to use X pixmaps as textures. A more fleshed out variation on that specification, GLX_EXT_texture_from_pixmap, will subsume that functionality and make it possible for direct-rendering OpenGL clients to use X pixmaps as textures. The GLX_EXT_texture_from_pixmap specification is currently in review. We intend to

provide an implementation of this GLX extension in a future driver release, and expect that other OpenGL implementers will do the same.

In order to bring Composite to the mainstream:

- OpenGL implementers need to add support for direct-rendering OpenGL to redirected windows; we are actively implementing this and expect that other OpenGL implementers will do the same.
- We need to address Xv + Composite within the current X driver architecture -- the XF86VideoAdaptor API needs to give drivers a way to get the drawable that they are rendering to.
- X driver developers need to continue to improve their Render implementations; EXA is a great step in that direction for the drivers that use EXA. NVIDIA will continue to improve the performance of its Render implementation.
- We need to establish industry standard benchmarks to measure the usage cases we feel are important for a composited X desktop. Does x11perf measure all the Render cases that are important? As OpenGL-based composite manager technology matures, we will need to construct benchmarks to measure the relevant OpenGL performance.

  Encouraging healthy competition, through the posting of industry standard benchmark results on hardware review websites and perhaps even www.spec.org will ensure that driver developers and IHVs take this seriously.
- We need to establish industry standard conformance tests to exercise all the relevant functionality. rendercheck [28] appears to be the defacto Render conformance test. Is rendercheck exhaustive? Should it be folded into the VSW suit?
- We need to fix the remaining Composite bugs in the X.Org Composite implementation, and finally enable the Composite X extension without a special X configuration option.

By focusing our efforts on these incremental improvements to the existing model, we believe we can achieve the best possible composited X desktop in the shortest period of time.

Conclusion

An X server implemented using the existing loadable driver framework can provide high performance Render acceleration and can fully support a composited desktop while still providing vendors with the flexibility to enable users with additional features. Alternatively, the X-on-OpenGL model, and the Xgl implementation of that model, is a significant technical achievement on the part of its authors but is not flexible enough to be an appealing solution to NVIDIA. Because of its flexibility and the manageable, incremental nature of the needed changes, we believe the existing X.Org DDX loadable driver framework is the best means to achieve our goals.

NVIDIA is committed to doing its part to make this a reality.

References

[1] Brian Paul. Online Resources: EGL_MESA_screen_surface, February 2006. http://www.freedesktop.org/wiki/EGL

[2] Keith Packard. Online Resources: X Render extension, February 2006. http://cvs.freedesktop.org/xorg/xc/doc/specs/Render/protocol?rev=1.1.1.1&view=markup

[3] Keith Packard. Online Resources: X Damage extension, February 2006. http://cvs.freedesktop.org/xlibs/DamageExt/protocol?rev=HEAD&view=markup

[4] Keith Packard. Online Resources: X Composite extension, February 2006. http://cvs.freedesktop.org/xlibs/CompositeExt/protocol?view=markup

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

[6] Keith Packard. Online Resources: TranslucentWindows, February 2006. http://www.freedesktop.org/wiki/Software_2fTranslucentWindows

[7] Keith Packard. Getting X Off The Hardware. June 2004. http://keithp.com/~keithp/talks/xserver_ols2004/

[8] The XFree86 Project. XFree86 server 4.x Design. December 2003. http://cvs.freedesktop.org/*checkout*/xorg/xc/programs/Xserver/hw/xfree86/doc/DESIGN

[9] Online Resources: DRI, February 2006. http://dri.freedesktop.org/

[10] Online Resources: NVIDIA, February 2006. www.nvidia.com

[11] Online Resources: ATI, February 2006. www.ati.com

[12] Thomas Winischhofer. Online Resources: SIS XFree86/X.Org Driver, February 2006. http://www.winischhofer.at/linuxsisvga.shtml

[13] Online Resources: Radeon Driver MergedFB Support, February 2006. http://dri.freedesktop.org/wiki/MergedFB

[14] Online Resources: Matrox DRI Driver, February 2006. http://dri.freedesktop.org/wiki/Matrox

[15] Online Resources: TechSource, February 2006. http://www.techsource.com/

[16] Andy Ritger. *An Overview of the NVIDIA UNIX Graphics Driver.* XDevConf 2006.

[17] Jon Smirl. Online Resources: *The State of Linux Graphics.* August 2005. http://www.freedesktop.org/~jonsmirl/graphics.html

[18] David Reveman, et al. Online Resources: Xgl, February 2006. http://www.freedesktop.org/wiki/Software_2fXgl

[19] Online Resources: Novell Brainshare 2005 Friday Keynote. http://www.novell.com/img/flash/load_stream.html?temp=1&id=bs2005_friday_keynote

[20] Peter Nilsson and David Reveman, Umeå University. *Glitz: Hardware Accelerated Image Compositing Using OpenGL.* In FREENIX Track, 2004 Usenix Annual Technical Conference, pages 29-40, Boston, MA, June 2004. USENIX. http://www.usenix.org/events/usenix04/tech/freenix/nilsson.html

[21] X.Org Foundation. Online Resources: PciReworkProposal, February 2006. http://wiki.x.org/wiki/PciReworkProposal

[22] Mark Vojkovich and Marc Aurele La France. XAA.HOWTO. Technical report, The XFree86 Project Inc., 2000. http://cvsweb.xfree86.org/cvsweb/xc/programs/Xserver/hw/xfree86/xaa/XAA.HOWTO?rev=1.13&content-type=text/vnd.viewcvs-markup

[23] Zack Rusin. EXA. X.Org email thread: *New acceleration architecture.* June 2005. http://lists.freedesktop.org/archives/xorg/2005-June/008356.html

[24] Online Resources: Luminocity, February 2006. http://live.gnome.org/Luminocity

[25] Zack Rusin. Online Resources: glxcompmgr, February 2006. http://cvs.freedesktop.org/xorg/app/glxcompmgr/

[26] Sun Microsystems. Online Resources: Project Looking Glass, February 2006. http://www.sun.com/software/looking_glass/

[27] Keith Packard, et al. X.Org email thread: *Finishing Composite to handle transformed windows.* January 2006. http://lists.freedesktop.org/archives/xorg/2006-January/011974.html

[28] Eric Anholt. Online Resources: rendercheck, February 2006. http://cvs.freedesktop.org/xapps/rendercheck/

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com