New Evolutions in the X Window System

Matthieu Herrb* and Matthias Hopf†

October 2005

Abstract

This paper presents an overview of recent and on-going evolutions in the X window system. First, the state of some features will be presented that are already available for several months in the X server, but not yet widely used in applications. Then some ongoing and future evolutions will be shown: on the short term, the new EXA acceleration framework and the new modularized build system. The last part will focus on a longer term project: the new Xgl server architecture, based on the OpenGL technology for both 2D and 3D acceleration.

Introduction

The X window system celebrated its twentieth birthday last year. After some quick evolution in its early years, its development slowed down during the nineties, because the system had acquired a level of maturity that made it fit most of the needs of the users of graphical work stations. But after all this time, pushed by the competition with other systems (Mac OS X and Microsoft Windows) the need for more advanced graphics features triggered new developments.

The first part of this paper is going to describe some of these features that are already available (and have been used) for a couple of years but not necessarily known by users of the X window system. A second part will address some on-going work that will be part of the X11R7 release: a new 2D acceleration architecture and the modularization of the source tree.

In the third part, a complete redesign of the device dependent layer of the X server, based on OpenGL, will be presented. It will allow a better integration of accelerated 2D and 3D graphics and make it possible to take advantage of the powerful 3D acceleration engines available today even in low end graphics adapters.

*CNRS-LAAS
†SUSE Labs

1 Already available new features

This section aims to remind a couple of the already available features, used by some toolkits to get better user experience with X: client-side font rendering, including anti-aliasing using Xft2 and fontconfig as well as rendering improvements based on the Render and Damage extensions.

It also presents the Composite extension and explains how the composite manager can be used to achieve various effects (transparency, shadows,...), taking advantage of the Render code already present in the existing X server.

1.1 The Render extension

The original X protocol provides a display model based on traditional boolean operations between source and destination. The Render extension was designed to enhance this model by adding image compositing operations. Image compositing was formalized by T. Porter and T. Duff [6], and implemented in the Plan 9 window system by R. Pike and R. Cox. Keith Packard designed and implemented the Render extension in XFree86 [3]. Porter-Duff compositing adds a pixel opacity value called "alpha" to its color attributes. This opacity value can be used to represent two different effects: translucency and anti-aliasing. The effect of translucency is created when all pixels of an object have their color computed as a combination of the intrinsic color of the object and the existing background values. Anti-aliasing is achieved by taking into account partial occlusion of the background by the boundaries of an object.

The Render extension implements new primitives for the display of images and polygons, as well as the basis for a new font rendering system that takes advantage of the image compositing features to render anti-aliased text.

Some of the core X applications have been extended to be able to use the X render extension: for instance `xclock` in analog mode now draws anti-aliased and translucent clock hands.

The Render extension was developed initially in XFree86 (now in X.org). It has been adopted by many commercial X providers too and thus can be assumed as a standard for modern applications.

1.2 Client-side font rendering

When X was originally designed, more than twenty years ago, it was decided that text display would be done by the X server. In the traditional X world, fonts are a server-side resource and applications depend on the fonts available in the server.

This approach had the advantage of limiting the amount of data that text-based applications have to send to the server, but it also caused lots of frustration among application developers. PDF or Postscript viewers for instance need to be able to render fonts that are embedded in the document they are displaying.

Moreover applications need access to more than just bitmaps in the fonts specifications for precise rendering. Attempts to extend the server-based font rendering model have all failed to solve all problems.

So, together with the introduction of Render, a radical decision has been taken to move font rendering from the server to client applications. To achieve this, a new text-rendering library has been designed; it is now at its second revision: Xft2. A companion library, fontconfig provides support to all font naming, installation and caching issues.

Fontconfig can, by the way, be used on a broader spectrum of applications than just X. It could be used by TEX like publishing applications, printer drivers and so on. Fontconfig uses XML formatted configuration files, located in the `/etc/fonts` directory.

Xft2 is based on the Freetype library. It can render several font formats: the traditional bitmap-based PCF format from the legacy server-side font system, Postscript Type 1 and True Type fonts.

Xft2 also provides some enhancements in the font encoding management, among others it is possible to use UTF-8 encoded text directly with Xft2.

Measurements have shown that the new client-side font rendering scheme has little to no impact on the overall performance. In many cases, it reduces the number of round-trips between the application and the server and thus even greatly improves application startup times.

Like Render, the Xft and fontconfig libraries have been embraced by more than XFree86 and X.Org. They are now the standard way of displaying text for X toolkits and applications. The legacy server-side mechanism is obsolete and should not be used by new developments anymore.

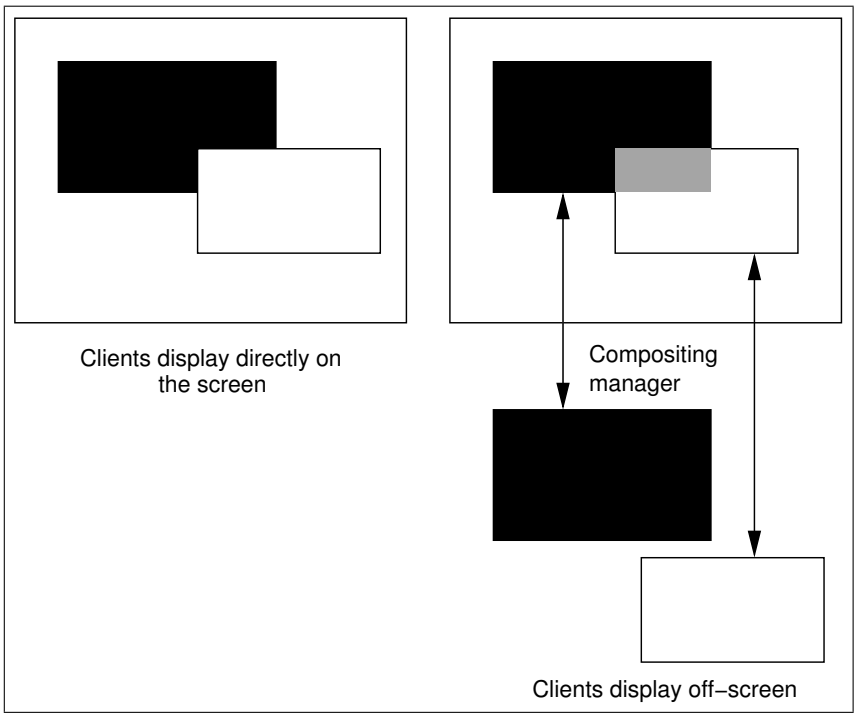
1.3 Composite, Damage, Xfixes extensions and the composite manager

To take the full advantage of the image compositing model provided by the X render extension, for example to provide translucent windows or to have a window manager add drop shadows to the windows, there are some bits missing. In the traditional X model, each application draws its window independently and doesn't take care of underlying or overlaying windows.

To be able to implement those eye candies, applications should be redirected to use off-screen windows, and a specific application, the *composite manager*, will work with the window manager and the new Composite extension to compute the screen's contents, doing compositing operations to produce translucency, shadows and anti-aliased polygons and texts.

To make this work, this application needs a bit more information than before about what is happening on the screen, and it needs this information in an efficient manner. This is the goal of the the Damage extension: it provides an efficient way to notify an application of damage done to a region of the screen by another application.

Figure 1: Composite manager principle: *traditional direct on-screen drawing on the left, vs off-screen drawing & compositing on the right.*



The Xfixes extension provides a general framework to extend the X protocol in order to work around some limitations in the core protocol. It currently contains five fixes. The more important ones allow better manipulation of the application cursor and export the region objects from the server to the clients.

`xcompmgr` is a sample composite manager that can be used with any existing window manager to provide some eye candy. Figure 2 shows the default OpenBSD desktop enhanced with xcompmgr and anti-aliased fonts in xterm and firefox.

KDE provides its own composite manager, `kompmgr`, while some window managers (Luminocity for instance) are integrating this functionality.

1.4 Cairo

Another important evolution in graphical user interfaces is the growth of vector-based graphics, as opposed to existing bitmap-based graphics. Vectors offer a better representation of screen contents, independent of the actual resolution, allow producing a perfect-looking printed version of an on-screen document, use less space for storage, and provide a better base for anti-aliased graphics.

With the introduction of the Render extension, X now has the ability of producing high-quality graphics based on vector representation.

Figure 2: Adding eye-candy to the default OpenBSD desktop

Cairo[1] is a library that implements vector based graphics with support for multiple output devices. Existing back-ends include X with the Render extension, and image buffers [4]. Experimental drivers include OpenGL (through the glitz library) and PDF files.

The Gtk+ toolkit as well as some existing applications already started to base most of their graphics on the Cairo library.

The OpenGL back-end offers some interesting features: on systems with accelerated OpenGL it provides the toolkits and application with a way to do accelerated 2D graphics that almost completely bypass the X libraries and server. However, this doesn't provide a solution for the global desktop compositing acceleration mentioned above.

2 Ongoing work

The current Xserver is divided in an architecture independent (DIX) and an architecture dependent (DDX) layer, which in turn loads the relevant hardware driver for rendering into the framebuffer. In order to accelerate drawing operations, the hardware drivers offer functions that implement certain operations using the graphics

[1] `http://www.cairographics.org`

hardware. The traditional interface for this is the Xserver Acceleration Architecture (XAA), which mainly focuses on accelerating core X protocol requests.

In contrast to the features described in the previous section most ongoing and future developments focus on the Xserver framework. These changes will not affect the programmer's API, as e.g. the Render and Composite extensions did, so all applications can immediately benefit from their implementation.

This section describes on-going work, which is available in the X11R6.9 / X11R7 release: the new 2D acceleration architecture EXA and the modularization effort. The EXA architecture is aimed at replacing XAA in drivers, focusing on accelerating primitives used by modern applications based on the render extension. The modularization effort will be described from an architectural point of view.

2.1 A new acceleration architecture for Render: EXA

Without composite manager, the performance of the Render extension is decent on reasonably recent hardware, that doesn't even deserve the "fast" qualifier. However, most of of Render computations are done on the main CPU and take little advantage of GPU acceleration. Render takes advantage of MMX or SSE instructions when they're available, and there have been some work done to add basic hardware acceleration for Render in the radeon driver.

When the composite manager is involved, things get worse, performance-wise. Even today's "fast" hardware can feel slow with compositing enabled. It is thus mandatory to rework the acceleration framework so that Render and Composite can be accelerated much better.

The currently used acceleration architecture in Xorg (XAA) is unsuitable for modern desktop usage. As a result of heavily using the card's 2D engine to accelerate mostly rarely used operations (like pattern fills and Bresenham lines) it invalidates any backing store that the X server might have on a region. Furthermore accelerating the Render extension using XAA is rather complicated and severely limited by its memory manager.

EXA (for EXcellent Architecture or Ex-kaa aXeleration Architecture or whatever) aims to extend the life of the venerable XFree86 video drivers by introducing hooks that they can implement to more efficiently accelerate the X Render extension: solid fills, blits within screen memory and to and from system memory, and Porter-Duff compositing and transform operations. It has been implemented by Zack Rusin in X.org.

A couple of existing drivers have already been converted to use the new EXA acceleration framework if requested: the i810 driver for Intel graphics card adapters, the radeon driver for ATI Radeon cards, the sis driver and the i128 drivers.

2.2 Source tree modularization

One of the big tasks in the lastest X release has been a complete rework of the X build system. The existing source tree, built using the imake build system, was considered as a big monolithic thing in which most developers found themselves uncomfortable. The need for global releases, updating all drivers at once every six month or so doesn't really fit the market of graphics cards that can produce new models more often than that timeframe.

Based on the experiences of other software projects, it was decided to switch to a more modular organization of the project, with more or less independent components [8]. This new organization will allow drivers maintainers (or others) to make independent releases, whenever they are needed.

It was decided that the best tools to manage the build of this new modularized source tree are the GNU auto-tools. They have an existing large user and developer base, and thus feel easier to use by the majority of developers. Being maintained outside of the X.Org project is supposed to lower the maintenance burden on the X developers which are now free to concentrate on their code.

The existing source tree has been split in several components, and each of them is composed of independent packages. The main components are:

- *xproto* which holds all the header files describing the actual X protocol and extensions. There is one package for the core X protocol and one package per X extension (Shape, MIT-SHM, Render, etc.),
- *libs* which holds all the libraries, one package per library (X11, Xext, Xrender, etc.),
- *data* which holds several data files (bitmaps and icons, XKB data files, X cursors),
- *apps* which holds the sample applications provided by X.Org (twm, xcalc, xedit, xlogo, xman, xwd, etc),
- *xserver* which provides the different X servers (Xorg, Xnest, Xprint, Xvfb),
- *drivers* which provides the graphics cards drivers, each one in an independent package,
- *fonts* which provides several fonts packages,
- *doc* for the existing documentation that doesn't fit a a specific package,
- *utils* various utilities that help the modular infrastructure, including an autotoolized version of `imake`, for use with third party applications that still depend on it.

Dependencies and configuration of the new packages is heavily based on the `pkgconfig`[2] tool.

To make the transition smoother, X11R6.9 and X11R7 share the same source base, and as far as possible produce the same set of binaries. X11R6.9 is the last version of the monolithic tree, while X11R7 is the first version based on the new modular tree. Both releases should be equivalent feature-wise.

Future work will be done in the modularized tree only. Only patches and bug fixes will be done in the X11R6.9 branch.

3 The future: Xgl

This last section will present the ideas, the rationale and the work already done to move to a new X server rendering model, based on OpenGL and glitz. This Xserver, called Xgl[3], is mainly developed by David Reveman. Currently it has to be run on top of a regular Xserver, comparable to Xnest, but first steps have been made to use Embedded OpenGL (EGL) extensions to make it run stand-alone [7].

3.1 Why use OpenGL

When looking at the current state of the Xserver architecture, several shortcomings are getting obvious, which we will analyze in detail now:

- XAA does not match current rendering use and is difficult to extend,
- the X server is a mix of high level code (window management etc.) and low level code (drivers),
- there are little to no ideas how to support modern graphics hardware features like pixel shaders,
- the driver API is used by Xserver only,
- the driver API is basically 2D only,
- drivers are difficult to maintain outside of main development tree,
- future graphics hardware won't have a 2D acceleration core any more.

The current acceleration architecture, XAA, has pretty much reached the end of its productive life, as it is difficult to implement and maintain, and modern applications don't use many core X requests for rendering any more. Many new features like the Render extension have to be implemented and tested for each driver, which is a tedious and troublesome work.

[2] `http://pkgconfig.freedesktop.org/wiki/`

[3] `http://http://www.freedesktop.org/Software/Xgl`

EXA is one alternative that has already been discussed in the previous section, but it still has the disadvantage that it keeps the driver code inside the Xserver, while it would be a worthy goal to have a real driver abstraction layer.

Both acceleration architectures are 2D only APIs, that are used by the Xserver alone and not by other programs. APIs that are used by only a small number of programs tend to be less stable and flexible than APIs used by many programs. While using 2D for a windowing system makes basically sense, there are several ideas how 3D user interfaces could enhance productivity in the long-term future, for instance with the project Looking Glass[4].

On the X.Org developer's conference 2005 all attendants agreed that using the industry standard 3D graphics interface OpenGL is a worthy investigation for a driver abstraction layer. David Reveman showed an early version of his Xgl prototype, which since then has matured and supports OpenGL based implementations of most important drawing operations in the Xserver. Additional features have been contributed by the community, for instance Xegl (Dave Airlie, Adam Jackson, John Smirl) and XVideo (Matthias Hopf).

Basically, Xgl has shown even in its early state, that using OpenGL for the drawing operations needed for an Xserver is a viable option, which additionally allows for more advanced compositing operations as it will be shown in Subsect. 3.3. It also gives easy access to modern features of graphics hardware like vertex and pixel shaders, and as the API continues to evolve we will see future capabilites exposed as well. Furthermore, hardware vendors use a lot more transistors and invest more in the design for the 3D core, so it is very likely to be faster than the 2D acceleration core.

The most important advantage is, however, that finally the Xserver has got rid of its hardware drivers, which can now be maintained outside the Xserver tree. E.g. Render is accelerated on every graphics hardware with OpenGL drivers, not just on the ones that actually implement the required acceleration interface. Having drivers removed from the Xserver core is especially important with future graphics hardware, which won't have 2D acceleration any more, and for which only closed-source OpenGL drivers exist. While vendors can (and do) implement their 2D drivers themselves as well, having a stable interface abstraction using a standard API will certainly improve the driver quality.

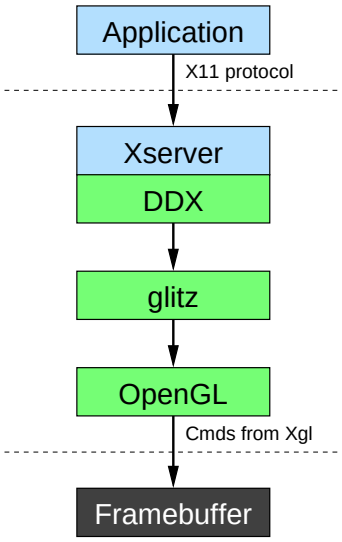
3.2 The architecture of Xgl

Figure 3 shows an overview over the Xgl architecture. At the moment Xgl is one additional DDX in the kdrive Xserver, which is an experimental Xserver mostly written by Keith Packard. After the Xorg modularization is finished, Xgl will slowly be integrated into the main stream Xorg server as an additional DDX as well.

OpenGL is still a relatively low-level API, so it made sense to create an abstrac-

[4] `https://lg3d.dev.java.net/`

Figure 3: Xgl architecture overview



tion layer that covers the most common graphics operations. As many X operations are pixmap oriented, texture handling is of particular importance.

Before working on Xgl David Reveman implemented an OpenGL based backend for Cairo [2]. The semantic of this backend, named glitz, closely resembles the Render protocol, and thus was the perfect abstraction layer for Xgl. Basically, glitz is an OpenGL image compositing library, which provides Porter-Duff compositing of images with implicit mask generation for geometric primitives. This includes, but is not limited to, alpha blending and affine transformations, and it has support for additional features like convolution filters and color gradients, which are not needed for Cairo. It also abstracts general texture use and the different sorts of OpenGL buffers.

There are no software fallbacks in glitz, if the hardware isn't capable of implementing a certain operation, glitz will just report the failure.

Certain operations of glitz require modern OpenGL features, for instance convolution filters or color space conversion and resampling for YUV textures both need pixel shaders. If the hardware isn't capable of these operations, a general software fallback inside glitz would result in poor performance, while the upper layer can easily implement this particular feature (e.g. color conversion) in software in an optimized way.

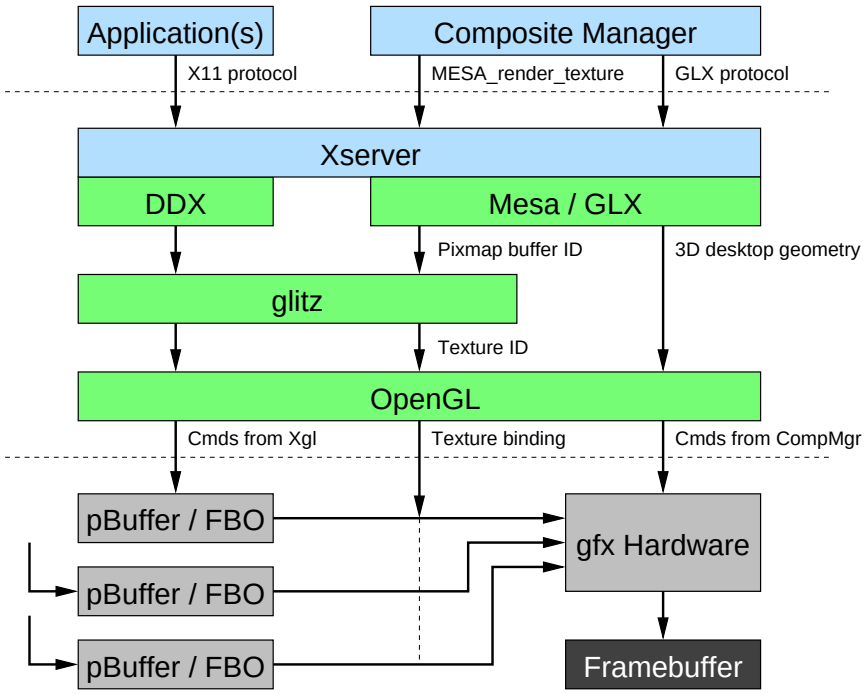
Applications that want to use OpenGL for drawing have to share the drawing space with the Xserver. As currently there is no way to share textures or framebuffers between applications, they currently have to use indirect rendering, i.e. the Xserver is doing the actual OpenGL calls it gets via the GLX protocol from the application. On one hand, this can be significantly slower for applications doing a

lot of memory transfer (video textures or geometry with high primitive count), on the other hand Xgl is now one of the few X servers capable of doing hardware accelerated indirect rendering, for example for running OpenGL programs remotely, which isn't implemented in Xorg yet.

3.3 Composite managers using OpenGL

As already described in Subsect. 1.3, all windows are rendered to off-screen pixmaps when the Composite extension is active. In the OpenGL case, this means the Xserver must render to an off-screen framebuffer, which can be provided by either the pBuffer or the more modern Frame Buffer Object (FBO) extension. Unfortunately, pBuffers are not yet widely supported, and implementation of FBOs is even less common and unstable. In these cases Xgl has to do all rendering to client windows in software and download the window contents to textures afterwards, which surprisingly is still quite usable.

Figure 4: Xgl in combination with a composite manager

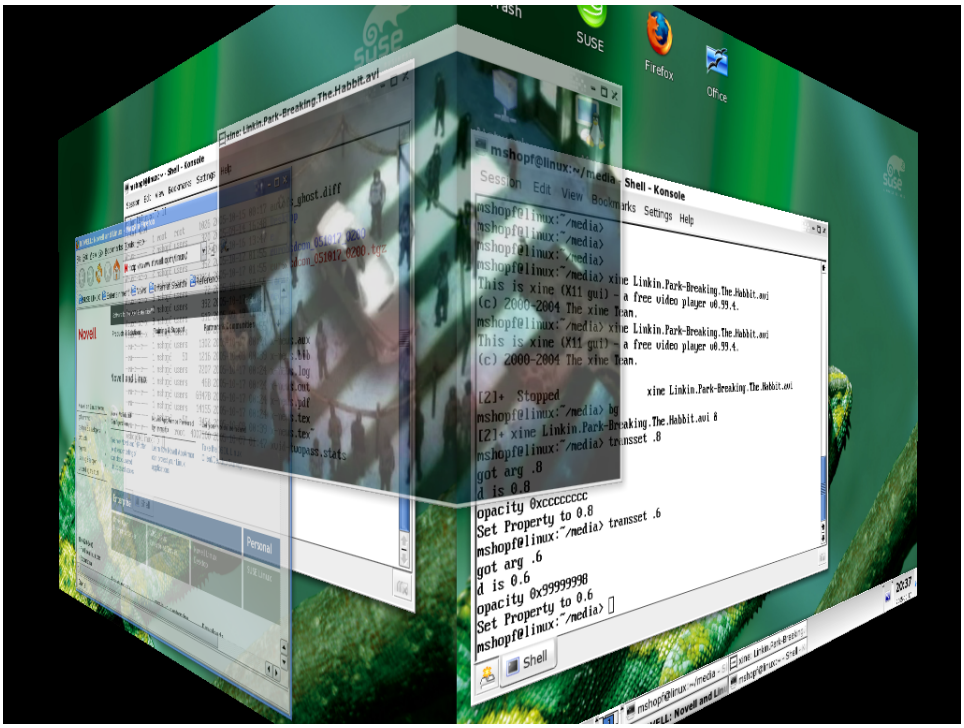


The pixmaps with the window contents can afterwards be composed using the Composite extension. An alternative to this is to use GLX to do the compositing with indirect OpenGL rendering. For this the composite manager has to be able to bind off-screen pixmaps to textures, which is done with the `GLX_MESA_render_texture` extension from Xgl. Figure 4 provides the com-

plete architectural overview over a session using an OpenGL based composite manager.

With this type of composite manager windows can be arbitrarily placed in 3D, which leads to pretty exciting rendering possibilities (see Fig. 5). Note that the pixmaps stay on the graphics hardware all the time, and only the geometry to be rendered has to be transfered from the composite manager to Xgl.

Figure 5: Fancy desktop switching with GLX based composite manager



3.4 Caveats and pitfalls

Currently Xgl is working best when run on top of a regular Xserver, comparable to Xnest. OpenGL provides neither facilities for creating a displayable framebuffer, nor for changing display modes. Both issues are addressed by the experimental `EGL_MESA_screen_surface` extension, which uses the buffer management ideas incorporated in Embedded OpenGL (EGL). The extension is close to being submitted to the Embedded OpenGL ARB for review. Right now there exists an early implementation in Mesa, named Xegl [5], for the R100 and R200 based Radeon chips from ATI, but several hardware vendors want to provide all extensions needed for Xgl to run in the future.

[5] `http://www.freedesktop.org/wiki/Xegl`

However, for full functionality, extensions for creating a hardware mouse pointer, getting monitor information, and setting drivers for different output plugs are needed as well. These extensions are not specified yet.

One major drawback of Xgl right now is that applications cannot do direct OpenGL rendering at all. For this an extension for sharing textures between address spaces is needed, as the application, Xgl, and the composite manager are all running in different address spaces. This is the subject of current discussions, but nothing is specified yet.

During the implementation phase of Xgl, several pitfalls have been encountered, but for most of them a reasonable solution has been found. First, with many OpenGL drivers one can easily get namespace collisions, as Xgl needs to be linked against a software rendering Mesa library for fallback and GLX handling as well as against the current OpenGL library on the host system. This can be solved by loading the OpenGL backend dynamically, which also allows for the Xegl backend to be loaded upon availability automatically. Then, frame buffer objects have turned out to be pretty unstable for many operations, so the code path using pBuffers will stay around longer than anticipated.

One source for major headaches in the open source community is of course the lack of open source drivers for modern graphics hardware, which are often only covered by binary only OpenGL drivers. One notable exception here is Intel, which has committed itself to providing open source drivers for future chips as well. Currently, their drivers are not yet equivalent to their competitors with respect to implemented features, but they are advancing steadily.

3.5 Implementation on BSD systems

Xgl currently runs on any system providing OpenGL, but it is unusable without hardware acceleration (i.e. without DRI support).

For the longer term, Xegl needs a console driver that provides a graphical mode with EGL drivers. The first implementations will be done on the Linux framebuffer driver. NetBSD and OpenBSD share the *wscons* console driver, on which some level of support for graphical console is already available. FreeBSD has his own console driver, *syscons*, that doesn't provides a graphical mode yet, as far as we know.

The integration of these graphical modes with hardware OpenGL acceleration (and DRI) is required to provide an EGL capable console with support for the necessary hardware setup extensions.

BSD developers will have to work with Linux DRI developers to make sure that the direct rendering infrastructure is kept in sync with the Linux DRI with respect to features like the proposed EGL extensions. A good way to help here would be to discuss the new extensions on the `dri` and `dri-egl` mailing lists so that no requirements are missed.

In the short term, with these extensions not completely specified, some more low-level hardware access might be necessary inside the Xserver, and BSD and

X.org should work closely together as soon as more development efforts are concentrated upon Xegl.

Conclusion

After a couple of years of relative stagnation in the world of the X window system, development has resumed, with the goal of providing rich enough functionalities for desktop environments which want to provide eye-candy on par with other Desktop OSs. The wide availability of cheap OpenGL-capable graphics cards makes such a new project realistic, although the lack of support for open source systems by most of the hardware vendors darkens the bright sky of this new technology.

References

[1] S. Nickell. Design fu: Xshots. `http://www.gnome.org/~seth/blog/xshots`, March 2005.

[2] P. Nilsson and D. Reveman. Glitz: Hardware Accelerated Image Compositing Using Opengl. In *Usenix 2004 Annual Technical Conference, Freenix Track*, pages 29–40, June 2004.

[3] K. Packard. Design and Implementation of the X Rendering Extension. In *Usenix Technical Conference, Boston*, June 2001.

[4] K. Packard. Cairo status. `http://keithp.com/~keithp/talks/cairo-exdc2005/`, June 2005. European X.Org developpers Meeting, Karlsruhe.

[5] K. Packard. X Status Report. `http://keithp.com/~keithp/talks/x-rearch-lca2005`, April 2005. Linux.conf.au.

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

[7] J. Smirl. The state of linux graphics. `http://www.freedesktop.org/~jonsmirl/graphics.html`, September 2005.

[8] D. Stone. X.org modularization. "Where to from here?". `http://people.freedesktop.org/~daniels/exdctalk/`, June 2005. European X.Org developpers Meeting, Karlsruhe.