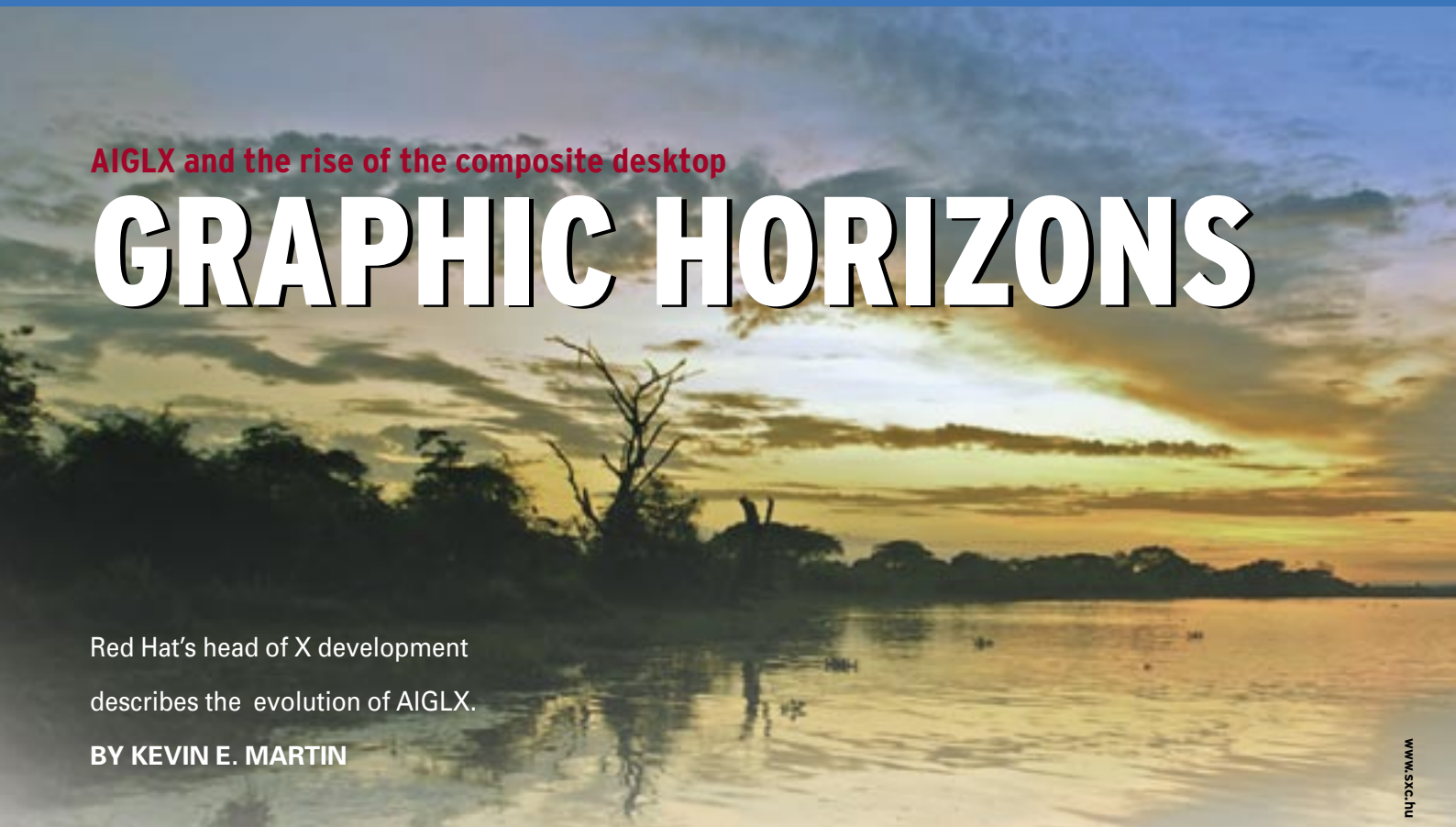

AIGLX and the rise of the composite desktop

GRAPHIC HORIZONS

Red Hat's head of X development describes the evolution of AIGLX.

BY KEVIN E. MARTIN

www.sxc.hu

Red Hat developers have been working on an alternative to Xgl, which they have dubbed AIGLX. AIGLX promises hardware accelerated desktop effects, while at the same time facilitating integration with the Xorg infrastructure.

The Fedora Rendering Project describes AIGLX as "a project that aims to enable GL-accelerated effects on the standard desktop. They add, "We have a lightly modified X server (that includes a couple of extensions), an updated Mesa package that adds some new protocol support, and a version of Metacity with a composite manager. The end result is that you can use GL effects on your desktop with very few changes, you can turn it on and off at will, and you don't have to replace your X server in the process."

AIGLX is part of an ecosystem of graphics components that interact to deliver better performance and more vivid images. How did all this happen and what does it mean for the user? This article describes the context for the evolution of AIGLX and a new generation of graphics technologies.

The State of Computer Graphics

The current desktop design is starting to show its age in several areas. First, the current desktop is designed around a 2D display device, while the graphics card hardware has shifted dramatically to emphasize 3D. Integrating 3D into the desktop has long been the goal, but until recently, it has not been possible. Other operating systems have also recognized this paradigm shift – Apple is using OpenGL through its Quartz [1] compositor architecture and Sun has a research project called Looking Glass [2] to experiment with using Java3D on their desktop.

A second problematic area is that all of our drawing is rendered directly into the front buffer. What this means is that users can see rendering artifacts while the desktop scene is being constructed. Most drawing is very fast, so it usually appears as a visually displeasing blur before the final image is visible, but sometimes is it much worse and you can see individual elements being drawn. Traditionally, toolkits have had to work around this problem by drawing directly to host-memory pixmaps and then copying the finished image to the screen.

The third problem is the static nature of the windowing states, and the fact that the transitions between those states are either instantaneous or have very primitive transition animations. For example, minimizing or unminimizing windows either simply pops windows into or out of existence, or very simple window outlines are drawn in sequence from the window to the icon in the Gnome panel to show the transition.

Building on the Past

The current vision for an OpenGL-based composited desktop is built on several key technologies that have been developed over the past several years.

Throughout most of the 1990s, the only open source implementation of OpenGL was Mesa [3], which was, at that time, a software-only client-side library that implemented the OpenGL interface. Then, in late 1998, Precision Insight began developing the Direct Rendering Infrastructure (DRI) [4]. This project brought hardware-accelerated 3D graphics to Linux, but as the name implies, it was only for direct rendering applications. Indirect rendering was implemented using the software Mesa code.

The DRI allowed applications that wanted to take advantage of the 3D hardware to do so through the OpenGL library (libGL). Traditionally, only specialized 3D applications wrote directly to OpenGL or one of its toolkits. These apps had no (or very little) direct knowledge of the 3D hardware underlying libGL because the DRI itself is a general purpose infrastructure that loads a card-specific driver to handle all accelerated OpenGL functions. The card-specific drivers have a well-defined set of entry points to handle initialization

and hook themselves directly into the libGL library.

With the relatively recent development of the Composite extension [5], we now have the ability to redirect 2D pixel data into host-memory or off-screen pixmaps. This pixel data can then be copied to the display buffer as needed to update what the user sees on their screen as their desktop. This effectively gives us the ability to double-buffer 2D data, an ability that has long been used by OpenGL applications to eliminate visual artifacts.

This ability is not new. The double-buffer extension (DBE) allowed individual apps to double buffer their output. What makes Composite unique is that individual application do not need to have direct knowledge of the DBE. Instead, an external application known as the composite manager controls when windows are redirected and how the pixel data is copied to the display buffer.

Luminocity

In late 2004, we began experimenting with using OpenGL to render redirected windows in our Luminocity project [6]. The example composite managers that had been developed previously used the Render extension to copy the window data to the screen (e.g., xcompmgr). As noted earlier, Apple was using OpenGL to achieve similar effects, and Sun had been experimenting with using Java3D in their Looking Glass project.

The basic idea behind Luminocity was to create OpenGL textures using the pixel data from each redirected window and then draw textured rectangles to the front buffer using each of those textures. Since the only open source hardware-accelerated OpenGL available at that time was through the DRI, Luminocity was developed to use direct rendering. The problem with this approach was that the pixel data had to be copied from the X server to the client before it could be used as a texture. These extra copy steps hurt performance.

Another difference was that previous composite managers were separated from the window manager, whereas Luminocity was a combined composite and window manager. By combining the two, Luminocity could not only copy window data to the screen and render static effects like window shadows, but it could also animate various state transitions. For example, we created the wobbly window effect, where windows were modeled with a simple spring system, so that dragging a window around would distort it as if you were pulling on one of the springs.

The New GI-based Composite Desktop

One of the primary performance problems with Luminocity was the number of data copies required to get the redirected window pixel data into a texture that could be used by the hardware. Ultimately, we want to get to the point where no copying of redirected window pixel data is necessary, i.e., a redirected window could be drawn to an off-screen pixmap in the correct format to be used directly by the hardware's 3D engine. To this end, we implemented two new infrastructure-level technologies: Accelerated Indirect GLX (AIGLX) and the GLX EXT_texture_from_pixmap extension (TFP). On top of this new infrastructure, we built a new scene-graph-based compositing library that is used by the Metacity window manager to implement the OpenGL animation effects.

Accelerated Indirect Rendering

As noted earlier, indirect rendering was completely unaccelerated in the initial DRI project. The plan had always been to implement accelerated indirect rendering using the same card-specific driver code that is loaded on the client-side by libGL; however, it was not a simple task, and the driving issue to make this happen did not occur until we came to the GL-based composited desktop.

The software Mesa driver used in the initial DRI work was based on the client-side version of Mesa, which translated OpenGL requests into X11 drawing commands. We modified this code, which was previously called libX11, to instead call the equivalent internal X server functions. We named this version GLcore. The interfaces used to initialize and call into GLcore were the *__GLinterface* and *__GLdrawablePrivate* structs, which are part of the OpenGL sample implementation (SI) [7].

However, the DRI project used Mesa instead of the SI, so the interface to the card-specific drivers were Mesa-based, which is quite different from the GLcore interface. This impedance mismatch was one of the reasons that it took so long to implement AIGLX. So, to take advantage of the DRI/Mesa card-specific drivers within the X server for AIGLX, we had to reconcile these two interfaces.

We developed a new abstraction layer [8], which is heavily based on the DRI interface, to provide the glue logic between the server-side GLX extension code and the card-specific driver. The new interface provides three objects: *__GLXscreen*, *__GLXcontext* and

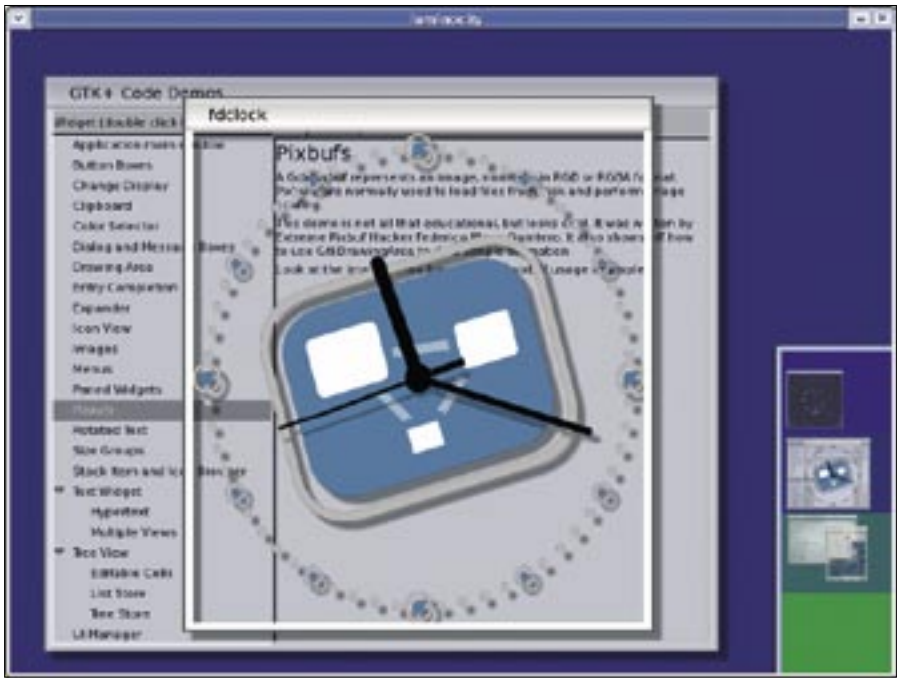

Figure 1: Luminocity supports transparent windows.

__GLXdrawable. Methods for allocating the DRI-specific objects and calling into the card-specific driver are contained entirely within the abstraction layer, which we called the DRI provider.

Since not all graphics cards have card-specific 3D drivers, and since several other servers (e.g., Xnest) that provide GLX support cannot use hardware drivers, the *GLcore* module must remain available, so we rewrote the top level of *GLcore* to use the new interface. This allows it to be used in place of the card-specific drivers when needed or desired, and it is called the GLcore provider.

To initialize the GL module for each screen, a stack of GL providers are called and the first provider that returns a non-NULL. *__GLXscreen* claims that screen. This mechanism allows for future GL modules to implement their own *__GLXprovider* and hook into the provider stack.

GLX_EXT_texture_from_pixmap

With accelerated indirect OpenGL, we can now render directly from within the X server process; however, we still need to be able to use the window pixel data that was redirected to a pixmap with the Composite extension as a texture. This is what the texture from pixmap GLX extension provides.

The simple approach would be to copy the data either through the protocol via *XGetImage* or through a shared-memory pixmap into the client's address space, and then the direct-rendered composite manager could use that data as the source for a *glTexImage2D* or *glDrawPixels* call. This does not work in practice due to the high overhead of copying pixel data to and from video memory.

THE AUTHOR

Kevin E. Martin has been working on the X Window System for the past 17 years. He was the principle architect of the Direct Rendering Infrastructure (DRI) and Distributed Multihead X (DMX) system. He has also developed many 2D and 3D drivers. He is the head of X and OpenGL development at Red Hat, serves on the X.Org Foundation Board of Directors, chairs the X.Org Foundation's Modularization Working Group and was the overall release manager for the last Xorg releases.

A better approach is to keep the pixmap data in the X server address space where it was rendered and use it directly as the source for a texture operation. *GLX_EXT_texture_from_pixmap* provides the interface to make that happen. As noted above, the ideal solution is to have the graphics card render the window contents into an off-screen buffer, which would then be used directly (i.e., with no copying or conversion) as the input to the hardware texture engine. To implement this solution, we will need additional infrastructure work (e.g., memory management) as well as additional card-specific driver work, both of which are currently in development.

Our intermediate TFP solution redirects window data into host-memory pixmaps and calls the texture operations directly through the new AIGLX abstraction layer interface to the Mesa/DRI card-specific driver. By rendering directly to host-memory pixmaps, we bypass the "read from framebuffer" operation, which can be quite slow.

Metacity

Luminocity was a toy window and composite manager that allowed us to rapidly prototype various technologies and experiment with using OpenGL in a composited desktop. We could have expanded Luminocity into a fully functional window manager, but this would have involved recreating the years of work that went into our standard desktop window manager, Metacity. Instead, we took what we learned from Luminocity and reimplemented it in Metacity.

Our approach was to create a new OpenGL scene-graph-based compositing library, called libcm, that would encapsulate the methods used by the rest of the window manager to draw the desktop. Metacity could then hook various state transition animations into the scene-graph as needed.

By making the full OpenGL interface available, we can create arbitrarily complex animations that are only limited by what we can dream and what the hardware is capable of. Some common effects that we already have or are in the process of implementing include minimization, maximization, menu fade in/out, drop shadows, window transparency, and workspace switching. Many others will be developed as the need arises.

It should be noted that, while AIGLX and TFP are critical to our GL-based composited desktop, we have developed them so they can be used independently by application developers. For example, Compiz [9] is another window/composite manager that takes a different approach but works well using technologies we have developed [10].

Technology Preview

We currently have a technology preview that redirects windows to host-memory pixmaps in the X server and then uses the texture from pixmap extension to use the host-memory pixmap as a texture source. This eliminates all but the last data copy and provides reasonable performance. You'll find more information, demos, and status about our GL-based composited desktop project at [11].

There is still much to do. We and others in the open source development community are in the process of adding several new technologies. These include input transformation, advanced memory management, redirecting extensions (e.g., Xv, GL, DRI), frame buffer objects, FBconfigs, and full GLX 1.3 support. As these and other technologies are developed, we will continue to update our GL-based composited desktop solution to take advantage of the new features. ■

INFO

[1] Quartz compositor architecture: *http://www.apple.com/macosx/features/quartzextreme*

[2] Looking glass desktop: *http://www.sun.com/software/looking_glass*

[3] Mesa: *http://www.mesa3d.org*

[4] DRI: *http://dri.freedesktop.org/wiki*

[5] Composite extension: *http://cvs.freedesktop.org/xlibs/CompositeExt/protocol?view=markup*

[6] Luminocity: *http://live.gnome.org/Luminocity*

[7] OpenGL sample implementation: *http://oss.sgi.com/projects/ogl-sample*

[8] GLX abstraction layer: *http://lists.freedesktop.org/archives/xorg/2006-February/013326.html*

[9] Compiz: *http://en.opensuse.org/Compiz*

[10] Compiz with AIGLX: *http://lists.freedesktop.org/archives/xorg/2006-March/013577.html*

[11] Fedora rendering project: *http://fedoraproject.org/wiki/RenderingProject/aiglx*