Introduction to the Direct Rendering Infrastructure

Brian Paul `(brianp@valinux.com)`

10 August 2000

This document is an introduction and high-level user guide for the Direct Rendering Infrastructure (DRI). It describes the goals of the DRI project, some of its history, the current state, the development environment and future plans.

---

1. Copyright

2. Trademarks

OpenGL is a registered trademark and SGI is a trademark of Silicon Graphics, Inc. Unix is a registered trademark of The Open Group. The 'X' device and X Window System are trademarks of The Open Group. XFree86 is a trademark of The XFree86 Project. Linux is a registered trademark of Linus Torvalds. Intel is a registered trademark of Intel Corporation. 3Dlabs, GLINT, and Oxygen are either registered trademarks or trademarks of 3Dlabs Inc. Ltd. 3dfx, Voodoo3, Voodoo4, and Voodoo5 are registered trademarks of 3dfx Interactive, Incorporated. Matrox is a registered trademark of Matrox Electronic Systems Ltd. Rage is a registered trademark of ATI Technologies, Inc. All other trademarks mentioned are the property of their respective owners.

3. Introduction

In simple terms, the DRI enables hardware-accelerated 3D graphics on Linux. More specifically, it's a software architecture for coordinating the Linux kernel, X window system, 3D graphics hardware and an OpenGL-based rendering engine.

Among the design goals for the DRI were:

- Allow high performance utilization of graphics hardware
- Support of a variety of different graphics hardware designs
- Support of multiple, simultaneous rendering by many client programs

- Security to prevent malicious misuse of the system
- Reliability to prevent hardware lockups or system deadlocks
- Portability to allow implementations on other operating systems and system architectures
- Compliance with the OpenGL and GLX specifications
- Integration with the XFree86 project
- Open-source implementation

4. History

Hardware-accelerated 3D computer graphics is not new - workstation vendors developed this technology over a decade ago. But now, inexpensive and powerful graphics hardware is commonplace. The technology is now available to anyone with a personal computer, not just professional engineers and researchers.

Anyone familiar with interactive graphics who was watching the development of Linux could see the need for (and indeed the potential of) implementing 3D graphics acceleration on Linux. The problem is that implemention of this technology is very complex. It would take a number of talented people working together to solve the problem.

4.1 OpenGL and Mesa

In 1993 Brian Paul began writing an implementation of the OpenGL specification, both for practical and personal reasons. Brian had been using an IRIS GL subset emulator called VOGL for a scientific visualization project when the OpenGL 1.0 specification was announced. As a graphics hobbyist, Brian thought it would be fun to implement a simple 3D graphics library using the OpenGL API, which he might then use instead of VOGL. After eighteen months of part-time development and checking with Silicon Graphic's legal department, Brian released his project, named Mesa, on the Internet. People quickly latched onto it and began contributing to its development.

Designed with correctness as a higher priority than performance, users soon recognized Mesa as a reliable and useful substitute for OpenGL on systems which did not otherwise have an OpenGL implementation.

In 1997 the first graphics hardware support was added to Mesa in the form of a Glide driver for the new 3dfx Voodoo graphics card. Suddenly, high performance 3D graphics was available at a modest price. This caught the attention of people who wanted to use inexpensive Linux systems instead of proprietary workstations for their daily work. But at this stage there was clearly a long way to go before Linux could replace or even compete with the graphics workstations of the day.

Both the hardware and the software had to be improved. The 3dfx Voodoo card was limited to full-screen operation at relatively low resolution, both in terms of pixels and color palette. The Mesa/Glide driver was not integrated with the window system and only a subset of OpenGL applications could benefit from the hardware acceleration.

In time the hardware would shed these limitations but on the software front, there was a tremendous challenge ahead. A real 3D workstation solution would require work in the Linux kernel, the X server, hardware drivers and the rendering core.

4.2 Precision Insight and the DRI

Precision Insight, Inc. (PI) was founded to develop hardware drivers for XFree86 through contracts with various IHVs and ISVs. PI had also set themselves a goal: to expand into 3D graphics in the future. Over time, PI was able to secure both funding and the personel to begin development of the Direct Rendering Infrastructure- the foundation for 3D graphics support on Linux.

The DRI would provide the infrastructure needed to integrate 3D hardware support into the X server and Linux kernel. Furthermore, it would provide a framework for building modular device drivers.

As of mid-2000 the DRI has been incorporated into XFree86 4.0 and at least five distinct hardware device drivers have been developed.

Nevertheless, the DRI is not "finished". The DRI was designed to be flexible in order to accomodate a variety of hardware architectures. Since the hardware designs are changing and growing more sophisticated the DRI will also evolve to accomodate these changes.

The initial development and ongoing evolution of the DRI is a prime example of the power of open-source development. A cooperative effort by the XFree86, Mesa, GLX and Linux kernel developers took several individual projects and combined them into a powerful new system.

5. DRI Components

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.

5.1 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. The DRI kernel modules reside in the `/lib/modules/`*kernel-version*`/misc/` directory. DRI kernel modules are named *device*.o where *device* is a name such as tdfx, mga, r128, etc.

Normally, the X server automatically loads whatever DRI kernel modules are needed.

5.2 The 2D XFree86 Driver

For each type of graphics card there is an XFree86 2D (or DDX) driver which does initialization, manages the display and performs 2D rendering. XFree86 4.0 introduced a new device driver interface called XAA which should allow XFree86 drivers to be backward compatible with future versions of the Xserver.

XFree86 drivers usually reside in the `/usr/X11R6/lib/modules/drivers/` directory with names of the form *device*_drv.o.

Each 2D driver has a bit of code to bootstrap the 3D/DRI features.

5.3 The 3D DRI Driver

The 3D capabilities of a graphics card are accessed through the 3D DRI driver. This driver essentially converts OpenGL command sequences into hardware commands. It then uses the kernel module to transmit the commands to the hardware. The 3D driver and kernel module basically implement the entire OpenGL rendering pipeline. The 3D driver implements as much of it as possible in user space while the kernel module does whatever is needed in kernel space.

3D DRI drivers usually reside in the `/usr/X11R6/lib/modules/dri/` directory with names of the form *device*_dri.so.

These drivers are loaded by the `libGL.so` library which implements the OpenGL API.

Most DRI 3D drivers today are based on Mesa. However, there is no DRI requirement to use Mesa. Driver developers may implement their 3D DRI drivers using any OpenGL-compatible software framework.

5.4 The libGL Library

OpenGL-based programs must link with the `libGL` library. `libGL` implements the GLX interface as well as the main OpenGL API entrypoints. When using indirect rendering, `libGL` creates GLX protocol messages and sends them to the X server via a socket. When using direct rendering, `libGL` loads the appropriate 3D DRI driver then dispatches OpenGL library calls directly to that driver.

libGL also has the ability to support heterogeneous, multi-head configurations. That means one could have two or more graphics cards (of different types) in one system and libGL would allow an application program to use all of them simultaneously.

Normally `libGL` loads 3D DRI drivers from the `/usr/X11R6/lib/modules/dri` directory but the search patch can be overriden by setting the `LIBGL_DRIVERS_PATH` environment variable.

5.5 XFree86 DRI Extension

The XFree86-DRI X server extension is used basically used for communication between the other DRI components (the X server, the kernel module, libGL.so and the 3D DRI drivers).

The DRI module maintains DRI-specific data structures related to screens, windows, and rendering contexts. When the user moves a window, for example, the other DRI components need to be informed so that rendering appears in the right place.

The XFree86-DRI module usually resides at `/usr/X11R6/lib/modules/extensions/libdri.a`.

5.6 XFree86 GLX Extension

The GLX extension to the X server handles the server-side tasks of the GLX protocol. This involves setup of GLX-enhanced visuals, GLX context creation, context binding and context destruction.

When using indirect rendering, the GLX extension decodes GLX command packets and dispatches them to the core rendering engine.

The extension usually resides at `/usr/X11R6/lib/modules/extensions/libglx.a`.

5.7 XFree86 GLcore Extension

The GLcore module takes care of rendering for indirect or non-local clients. Currently, Mesa is used as a software renderer. In the future, indirect rendering will also be hardware accelerated.

The extension usually resides at `/usr/X11R6/lib/modules/extensions/libGLcore.a`.

5.8 Modularity and Version Checking

Each of the DRI components has version numbers which are checked by all dependents. If there are any version mismatches between the DRI components the system will recover gracefully and provide diagnostic messages.

Errors detected on the server side will be logged with the normal X server messages.

Errors detected on the client side by libGL will be printed to stderr if the `LIBGL_DEBUG` environment variable is set.

Finally, thanks to the modularity of XFree86 and the DRI it is also possible to change the graphics hardware in the system, power-up, and resume using the system without having to install any new software.

6. Using the DRI

When everything is installed and configured correctly, no special actions are needed to enable 3D rendering. Simply start the X server as usual.

Applications linked with libGL will automatically be hardware accelerated. If 3D hardware is not installed, OpenGL-based applications will still run correctly using GLX and the software renderer in the server.

At this time, most of the DRI drivers are based on Mesa and cannot be called official OpenGL implementations. Nevertheless, conformance is reasonable now and improving all the time. The majority of OpenGL applications work just as expected. In some cases Mesa-based drivers have proven to be "more correct" than other OpenGL implementations, exposing portability problems or bad assumptions made in applications.

The DRI is now being used to run all kinds of OpenGL applications. Games (the 3D killer app) are very popular on Linux while the Linux workstation is now also a reality. There are numerous 3D modelers, simulators, visualization and 3D content creation tools running on the Linux DRI today. It is interesting to note that a single hardware/OS platform can run both the most popular 3D games and a large variety of 3D productivity applications just as well.

7. Supported Hardware

The original DRI and 3D driver work was done for the Intel x86 architecture. Support for Alpha-based systems is underway and support for other CPU architectures will probably arrive in time.

With the release of XFree86 4.0.1 there is support for the following 3D graphics hardware:

- 3dfx Voodoo3 and Voodoo5 series
- Matrox G200 and G400
- Intel i810
- ATI Rage 128
- 3Dlabs Oxygen

At this point in time, the drivers have not been fully optimized for performance. The limited development resources have been focused on dependability, correctness and functionality. In the long term, performance will optimized.

8. Supported Operating Systems

The initial DRI development was targetted for Linux, but with an eye toward portability to other Unix-like operating systems. FreeBSD is the first non-Linux port of the DRI. It is included in XFree86 4.0.1 with support for 3dfx and Matrox graphics hardware.

Since most hardware drivers require AGP support, a recent Linux kernel is needed. Until Linux 2.4.0 is released, the recent 2.3.x kernels must be used. Older kernels such as 2.2.14, patched with AGP support may also work.

9. Open Source Development

The DRI is open source, copyrighted under the XFree86 license terms. The benefits of open source development, which have been well-documented elsewhere, have certainly applied to the DRI project. Some particular benefits to the DRI project follow.

9.1 Quality Assurance

Since the latest DRI source code can always be downloaded at any time, users are able to closely track new DRI development. Enthusiastic users often update their DRI installation on a daily basis and provide constant feedback to the developers.

The user community is able to test the software on a far wider variety of systems than the developers could themselves. There are innumerable combinations of motherboards, processors, graphics cards and software installations. Thanks to the users, the developers can be sure of substantial coverage testing.

9.2 New Development Projects

There are several instances of new DRI 3D drivers being developed by parties outside the core DRI development team. In a closed development environment the core DRI team would have to develop all drivers, which clearly would be a bottleneck. With open-source anyone is free to implement new drivers.

In another instance, the DRI infrastructure has been ported to FreeBSD by an independent developer. In many cases, porting software to new a hardware or OS environment can help to improve the overall design and quality of that software.

9.3 Shared Code Base

Since many of the DRI drivers are build using Mesa as their rendering core, all those drivers can simultaneously benefit from Mesa development. For example, Mesa bug fixes, optimizations and extensions can generally benefit all drivers, not just one. Similarly, improvements to the core DRI modules will be of benefit to all hardware drivers.

Contrast that to proprietary development: none of the development effort invested in one vendor’s driver will benenfit another vendor’s.

9.4 Community Communication

The communication between the developers and end users is very important. In closed/proprietary development the software developers are seldom known by name nor develop a rapport with the end users.

It’s encouraging to the DRI developers to get positive feedback from users about the latest features, bug fixes and performance improvements. The developers know how their work is being used and, indeed, how well it’s working.

An on-going dialog with end users can also bring about new innovations. For example, when users have a problem that can’t be solved by "off the shelf" software, it’s possible for the users and developers to collaborate on solutions to those problems. This might result in new rendering extensions, for example. This level of cooperation is exciting for both parties.

10. Future Work

The term *direct rendering infrastructure* does not only imply 3D graphics support. From the beginning, the DRI was designed with flexibility such that it may also be used in other areas such as video I/O.

That’s a potential future project.

As stated before, the immediate goal of the DRI drivers has been stability and correctness, not maximum performance. With maturity, the drivers will also become faster. Some of that work is now underway.

There are always new 3D cards coming to market. The DRI team expects to develop drivers for many of them.

OpenGL is always growing with new extensions. In order to remain competitive with other operating systems, the Linux 3D driver writers need to implement those new features. Thanks to open source and shared code bases many new extensions may be made available even in hardware drivers which may have been considered to be finished.

In the near future major Linux vendors will be incorporating Linux 2.4.x and XFree86 4.0.x into their products. At that time, 3D hardware acceleration will be widely available without need to manually install 3D support piece-by-piece. Hardware 3D will finally become a staple of the Linux experience.

11. References

1. A Multipipe Direct Rendering Architecture for 3D, Precision Insight, Inc. (http://www.precisioninsight.com/dr/dr.html)
2. Direct Rendering Infrastructure, Low-Level Design Document, Precision Insight, Inc. (http://www.precisioninsight.com/dr/drill.html)
3. The Direct Rendering Manager: Kernel Support for the Direct Rendering Infrastructure Precision Insight, Inc. (http://www.precisioninsight.com/dr/drm.html)
4. Hardware Locking for the Direct Rendering Infrastructure Precision Insight, Inc. (http://www.precisioninsight.com/dr/locking.html)
5. A Security Analysis of the Direct Rendering Infrastructure Precision Insight, Inc. (http://www.precisioninsight.com/dr/security.html)
6. DRI Extension for supporting Direct Rendering Protocol Specification Precision Insight, Inc. (http://www.precisioninsight.com/dr/XFree86-DRI.txt)
7. DRI Home Page (http://dri.sourceforge.net)
8. DRI User Guide Precision Insight, Inc. (http://dri.sourceforge.net/DRI.html)
9. DRI Compilation Guide Precision Insight, Inc. (http://dri.sourceforge.net/DRIcompile.html)