LinuxWorld 2000 in San Jose

Tutorial HL: Linux 3D Hardware Acceleration

David Blythe `(blythe@bluevoid.com)`

July 2000

This whitepaper complements LinuxWorld 2000 San Jose Tutorial HL with an introduction to implementations of the OpenGL® API based on the sample implementation ("SI") from Silicon Graphics, OpenGL conformance, and the new "oglbase" Linux/OpenGL Base Standard, and ends with a brief discussion of the future directions OpenGL under Linux may take.

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.

3D Solutions for Linux

SGI’s Sample Implementation (SI)

SGI maintains a sample implementation of GLX, OpenGL, and GLU., usually refered to as ’the SI’. This implementation serves as a reference implementation for SGI’s OpenGL licensees. This implementation has been in development since 1992 and is the most mature implementation of OpenGL available. SGI made this sample implementation available as open source in January of 2000. The OpenGL implementation currently supports indirect rendering only. The GLX portion of the implementation has been integrated into XFree86 (4.0) by Precision Insight.

The SI consists of several pieces, the X server GLX protocol and OpenGL renderer implementation, the client side GLX and OpenGL bindings, the GLU implementation, header files, utility and sample code (sample code from the 'red book') and man pages.

More information on the SI implementation is available at http://oss.sgi.com/projects/ogl-sample/.

NVIDIA's Drivers

NVIDIA, SGI, and VA Linux have collaboratively developed XFree86 4.0 and OpenGL 1.2 drivers for the TNT, TNT2, TNT Ultra, GeForce, and Quadro chipsets including AGP4x support. These drivers support direct rendering. The drivers are not open source, but x86 Linux binaries are available for download from NVIDIA's website http://www.nvidia.com/Products/Drivers.nsf) and have been tested against the RedHat 6.1 and 6.2 Linux distributions.

The Linux NVIDIA driver shares the same source code base with the Microsoft Windows drivers so it has had a lot of application exposure and provides the full OpenGL 1.2 features set as well as all of the NVIDIA extensions. The driver consists of several parts, the 2D X driver, a loadable kernel module that provides the direct rendering support, and the client side code the application links to (`libGL.so`, `libGLU.so`). To allow the loadable kernel module to work with different versions of the Linux kernel, NVIDIA ships some glue code in source form that can be compiled against the kernel header files. This allows the module to be used with minor variations in the kernel, e.g., 2.2.13, 2.2.14, but a new release of the kernel module will likely be necessary for major change in the kernel.

The GLU implementation is from the OpenGL sample implementation.

NVIDIA also provides an open source OpenGL and XFree86 3.3.5 driver implementation on their website. The implementation supports the NV1, RIVA 128, RIVA 128ZX, RIVA TNT, RIVA TNT2, and GeForce 256 chipsets. This driver has lower performance than NVIDIA's proprietary driver but it does include source code.

Xi Graphics' 3D Accelerated-X

Xi Graphics, Inc. (http://www.xig.com/), sells an X server including OpenGL client libraries and accelerated rendering as part of the "3D Accelerated-X" package. They support at numerous architectures (ATI, Matrox, 3Dlabs, Number Nine, 3dfx, S3) in an "entertainment" version for commodity cards and a "professional" version which provides workstation features such as overlays and accelerated antialiased lines. They also support multiple heads and laptops.

Although Accelerated-X is not Open Source, it is an OpenGL 1.1 implementation which supports accelerated textured polygons, fast clear, and fast swap, and provides acceleration for cards not likely to be supported through Mesa either because of lack of availability or lack of chipset documentation. Many 3D companies provide their chipset specification to Xi Graphics in order to receive OpenGL support on Linux such as Evans and Sutherland, SiS, and Number Nine.

Developers looking for a fully compliant OpenGL implementation and a vendor to provide support may contact Xi Graphics for more information.

Metrolink's Metro OpenGL

MetroLink's (http://www.metrolink.com/) "Metro OpenGL" is another X server providing OpenGL acceleration for a number of hardware platforms, including ATI, NVIDIA, S3, 3Dlabs, Matrox, Evans & Sutherland, etc).

Like the Xi Graphics product, this is a combination X server and OpenGL implementation. The implementation supports OpenGL 1.2 functionality with indirect rendering. There is not support for direct rendering, but any card supported by the X server can be used with OpenGL. The implementation is not available as open source.

Workstation Vendors

Both SGI and Hewlett Packard provide Linux 3D solutions. At the time of writing SGI is offering the 230, 330, 550 line of X86-based machines with V3 and VR3 graphics (enhanced NVIDIA GeForce/Quadro graphics) (http://www.sgi.com/workstations/230/

HP is currently offering the Visualize series of X86-based workstations with NVIDIA TNT2 graphics (http://www.hp.com/visualize/products/linux/ and has an fx-based product in Beta. The fx5 and fx10-based products use an X11 server derived from HP's HPUX code base, rather than XFree86 and includes a separate open source kernel module (leveraging code from the DRM/DRI) to provide direct rendering support. Longer term HP is looking at moving to a standardized infrastructure such as the DRI.

Completeness and conformance issues for OpenGL and Mesa

Conformance & Testing

The behavior of the OpenGL API is defined by the OpenGL specification. A conformance test (called 'the conformance test') was developed by the OpenGL ARB for validating the correctness of OpenGL implementations. An implementation can only be called an OpenGL implementation and use the OpenGL logo if it passes the 'must pass' portion of the conformance test (and the implementation provider has a license from SGI). The 'must pass' part of the conformance test is not especially strict nor comprehensive, but provides basic testing that the OpenGL implementation works. There are additional tests in the conformance suite that licensed vendors must run, but they do not necessarily need to pass them to ship their implementation and still call it OpenGL.

Microsoft (yes, this has nothing to do with Linux), includes the 'must pass' test and several other tests in their Windows Hardware Quality Lab (WHQL) conformance suite, that vendors shipping on the Windows platform need to pass in order to receive WHQL certification.

Vendors may have their own internal test suites. SGI's ogtst was distributed to some of the SGI's licensees and its coverage has been increased over time. Allen Akin started glean, which is a an open source suite of tools for testing the quality of OpenGL implementations. glean can also be used to compare two OpenGL implementations and highlight the differences. More information on glean is available at http://glean.sourceforge.net/

Both the OpenGL sample implementation (SI) and Mesa have been tested using the OpenGL conformance test, and both pass most of the tests.

ABIs & Versioning

In order to be able to release a single version of an OpenGL application that runs on different releases of Linux, there must be agreement on sundry details such as the location in the file system of dynamic libraries, names of symbols, etc. For example, if Vendor A creates a library called `/usr/lib/libGL.so` and Vendor B creates one called `/usr/lib/libOpenGL.so` applications linked using Vendor A's libraries will fail on platforms that have only Vendor B's libraries installed.

Similarly, to be able to compile an OpenGL application on different Linux releases without messing with lots of compile time configuration information, the locations of include files and contents of these files must be agreed upon. Some of the rules come directly from the OpenGL specification (e.g, the names of functions, #defines, etc), but others such as the location of include files, dynamic libraries, etc require a separate effort to achieve agreement.

Before looking at the Linux/OpenGL Base standard, it is useful to understand the versioning theory behind OpenGL. The OpenGL specification was written with the intent of having new functionality added over time. The rules for adding new functionality are simple, new functionality can not break compatibility with existing functionality. Therefore, tokens and symbols are never removed from the specification, and the definition of new functionality can not change old functionality in such a way that existing applications would break. For example, it is legal to add a new texture environment mode to combine a texture color and fragment color together, but it would not be legal to redefine the behavior of an existing texture mode.

The rules are simple, and they make it simple to release new versions of the library without breaking existing applications. If the OpenGL vendor follows the rules correctly (and doesn't introduce any bugs into their new implementation) then there is no reason to have anything other than the latest version of the library installed and there is no need to use any *DSO versioning* mechanism since the new library is backward compatible with old libraries.

For OpenGL SDK users the OpenGL header files all include tokens that define the version of OpenGL available on the system. These tokens can be tested at compile time to and allow developers to write portable application code that at compile time can optionally use new functionality when it is available. OpenGL also provides some mechanisms for testing for functionality at runtime, these mechanisms will be discussed later.

Problems with applications requiring a particular version of the library, for example, an application needing OpenGL 1.2 functionality, is left largely unaddressed by the OpenGL specification. Package installation tools such as the Redhat Package Manager are capable of testing for pre-requisites before installing an application. OpenGL provides a mechanism for querying the version at runtime, but the application is like to fail with unresolved symbols before the query is executed.

Besides additions to the core functionality, new functionality can be added in the form of extensions. In fact, it is preferred to vet new functionality first as an optional extension before adding it as baseline functionality. Extension management is a complicated topic and will be addressed after looking at standardization.

The Linux/OpenGL Base Standard (oglbase)

Recently renamed to **OpenGL Application Binary Interface for Linux** The Linux/OpenGL Base standard attempts to address these remaining issues by defining both the application binary interface (ABI) and runtime environment for the Linux platform. The effort also defines an SDK for developing portable applications. The SDK definition includes the locations of header files and conventions for extension usage.

oglbase's scope is limited to just the OpenGL-related components (OpenGL, GLX, and GLU) on Linux under X11. Other Linux ABI and runtime standards are addressed by the Linux Standard Base (http://www.linuxbase.com/). Other OpenGL-related toolkits are left to their maintainers to decide on ABI issues.

A brief summary of Version 1.0 of the ABI is:

- Definition of the OpenGL data types for the IA32 platform (GLint, etc).
- Definition of libraries:
  - `/usr/lib/libGL.so (DT_SONAME = libGL.so.1)`
    standard OpenGL 1.2 (+ ARB_multitexture) and GLX 1.3 entry points
  - `/usr/lib/libGLU.so (DT_SONAME = libGLU.so.1)`
    standard GLU 1.3 entry points
- libGL.so must also include GLX_ARB_get_proc_address extension for extension query
  http://oss.sgi.com/projects/ogl-sample/registry/ARB/get_proc_address.txt
- Definition of header files in `/usr/include/GL`:
  - `<GL/gl.h>` OpenGL 1.2 + ARB_multitexture defintions
  - `<GL/glx.h>` GLX 1.3 + ARB_get_proc_address definitions
  - `<GL/glu.h>` GLU 1.3 definitions
  - `<GL/glext.h>` OpenGL extension definitions
  - `<GL/glxext.h>` GLX extension definitions
  - `gl.h` defines `GL_OGLBASE_VERSION`
- Libraries are thread-safe and support multi-threading using pthreads (`-lpthread`)

Since version 1.0 has only recently become available, most providers of OpenGL implementations have not had enough time to produce compliant releases and older releases of OpenGL and OpenGL-like libraries are likely to be non-conformant. XFree86 4.01 includes oglbase conformant OpenGL support. Unfortunately, there will be period of turmoil as end users update their platforms to include oglbase compliant releases.

More information on the OpenGL Application Binary Interface for Linux is available at http://oss.sgi.com/projects/ogl-sample/ABI/

Extensions

Many OpenGL vendors add new features to OpenGL as extensions. The OpenGL specification includes a set of rules for defining a new extensions. One of the main difficulties with managing extensions in a portable manner is that extensions are optional and two vendors may choose to implement different sets of extensions. For many applications it is beneficial to use the extensions when they are available since they will improve either the performance or the visual appearance of the application. The fact that there isn't a universally available set of extensions makes it difficult for an application developer to use extensions if they want to create a portable single binary for their application.

In order to use optional extensions at runtime the application developer includes multiple code paths in their application, a path that uses an extension and another that does not. When the platform does not include support for one of the extensions, the code path that does not use the extension is used instead. A problem remains in that the code that does use the extension, even though it is not executed, will result in references to unresolved symbols. There are several solutions to this problem. The first is to package the code path using the extension in a separate dynamic library (DSO) and only load it when the application will use that path. This can be rather cumbersome for the application developer. Another method is to allow applications to run with unresolved symbols. This works fine if the application never tries to execute code that uses those symbols, but it requires system support. The third option is to provide support in the library to bind symbols indirectly using runtime query functions to determine the addresses of the symbols rather than hard-coded references.

This third option has been included in the **OpenGL Application Binary Interface for Linux** as a GLX extension. The ARB_get_proc_address extension includes a new GLX query that looks up the address of the named function, similar to the `dlsym()` function for looking up symbols in dynamic libaries. Now an application that can take advantage of a particular extension just needs to query for the presence of the extension using the `glGetString` function and if the extension is present query for the corresponding function address using `glXGetProcAddressARB`. This provides an effective solution for deploying portable OpenGL Linux application binaries that are capable of using optionally using extensions.

Application developers may also be interested in making their source code portable as well. There are two aspects in making the use of extensions more portable in source code. The first is the specification of the extensions. SGI maintains a registry of OpenGL extensions (http://oss.sgi.com/projects/ogl-sample/registry/) that include a description of the behavior of the extension as well as the the definitions of the symbols and tokens. This enables vendors to work from a common definition of the extensions. The more significant extensions may be elevated to ARB status making them more likely to be implemented by multiple vendors. The second component is that SGI maintains header files (`glext.h`, `glxext.h`) that contain the function prototypes and token values for the extensions available in the registry. OpenGL vendors can include these files as part of their distribution and if necessary application developers can download (http://oss.sgi.com/projects/ogl-sample/ABI/glext.h) new versions of these files as extensions are added. These header files are included as part of the oglbase standard and are included automatically in `gl.h` and `glx.h` unless `GL_GLEXT_LEGACY` and `GLX_GLXEXT_LEGACY` are defined at compile time.

Futures

Direct Rendering Infrastructure

As time and energy permit, new architectures will be supported by baseline DRI drivers and the existing implementation will receive performance enhancements. The DRI will be ported to other hardware platforms and to other processors as part of XFree86.

A longer term goal for the DRI is to allow other OpenGL implementations to plug into the infrastructure and eliminate any need to have mutually incompatible versions of the OpenGL components on the same platforms. This would allow vendors to ship proprietary driver implementations and allow them to co-exist with open source implementations on the same platform. This will be benificial in the future for supporting multiple graphics cards simultaneously.

OpenGL Directions

OpenGL continues to evolve. Most of the effort is currently focused on developing and standardizing new extensions for using compressed textures, vertex array objects, morphing, etc. Information about extensions is available at http://oss.sgi.com/projects/ogl-sample/registry/

The OpenGL sample implementation has only recently been made available as open source. The implementation has been integrated with XFree86 3.3.6 and is in the process of being upgraded to XFree86 4.0.

SGI plans to make some additional software available as open source. Some parts under development include optimized geometry code for the SI implementation, the OpenGL Stream Codec (GLS), and the OpenGL Character Renderer (GLC).

oglbase Futures

The ABI definitions needs to be extended to include Linux on other platforms such as PowerPC and Alpha. This mostly involves mapping the OpenGL data types to the appropriate machine-specific types.

There are some small issues with the current specification that still need refinement or clarification. These include ensuring that dependencies on other libraries are either eliminated or full defined, e.g. in terms of the Linux Base ABI.

Finally, as new extensions become available and OpenGL evolves, these need to be considered for inclusion in future versions of the ABI.

OpenAL

Loki Entertainment Software and Creative Labs are driving the development of OpenAL, an audio API similar in concept to an OpenGL for audio hardware. OpenAL does not have a strict specification like OpenGL but does attempt to style its API like OpenGL in orthogonality and hardware encapsulation.

OpenAL provides 3D spatialized sound; it has the concept of a spatial listener and multiple sound sources; each has a velocity, orientation, and position. The following code sets up a listener and a simple source and plays a sample for a short period of time.

```
ALuint buf;
ALsizei size, bits, freq, format;
ALfloat back[] = {0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f};

alutinit(&argc, argv);

alGenBuffers( 1, &buf_id );
```

```
alutLoadWAV( "sound.wav", &wave, &format, &size, &bits, &freq);
alBufferData( buf_id, format, wave, size, freq );
free(wave); /* openal makes a local copy of wave data */

alGenSources( 1, &source_id);
alSource3f(source_id, AL_POSITION, 0.0, 0.0, -5.0);
alSourcefv(source_id, AL_ORIENTATION, backwards);
alSourcei (source_id, AL_BUFFER, buf_id);
alSourcePlay(source_id);

sleep(10);

alutExit();
```

OpenAL is available from the OpenAL web page at www.openal.org

OpenML

A number of companies including Discreet, Evans and Sutherland, IBM, Silicon Graphics, 3dfx, 3Dlabs, ATI, Intel, S3, and others have developed an initiative, called the Khronos Special Interest Group, to create a multimedia API for audio and video playback, recording, and synchronization, called OpenML. Not much information is yet available about OpenML at www.khronos.org, but Linux is in dire need of a DirectX-like API that provides audio and video services integrated together with accelerated 2D and 3D rendering.

OpenML is being modeled on the successful dmSDK digital media libraries on IRIX from SGI. Initial implementations of OpenML will contain software emulation of many features, with the intent (like OpenGL), that individual vendors will optimize paths through OpenML that correspond to their market segment and their hardware feature set. We don’t anticipate seeing mature OpenML implementations within the next year.

In Summary

OpenGL on Linux is maturing rapidly. Precision Insight’s Direct Rendering Infrastructure, XFree86 4.0, Mesa, and Silicon Graphics’ open source sample implementation of OpenGL 1.2 all provide a strong basis for vendors to provide open source Linux drivers for their hardware. The availability of hardware accelerated OpenGL for Linux strengthens both the OpenGL API for 3D rendering and Linux as a platform for 3D games, simulation, visualization, and design.

By the end of the year, we expect that drivers for the DRI and SI should bring robust production-quality OpenGL for Linux to the most popular commodity and high-end boards available.