AIX

Programming with OpenGL

By Linas Vepstas and Chandrasekhar Narayanaswami

OpenGL provides an easy-to-use, full-featured, network-transparent Application Programming Interface (API) for developing applications using 3-D graphics. This article provides a brief overview of OpenGL, focusing on how OpenGL differs from its predecessor GL, how it is supported on IBM platforms, and how it is integrated with the X Window System.[1]

The OpenGL Architecture Review Board (ARB) controls OpenGL™, which is licensed from Silicon Graphics®, Inc. OpenGL has a broad base of support in the computer industry, including DEC, Intel, and Microsoft. IBM is a founding member of the ARB, and OpenGL is a strategic API for AIX.

What Is OpenGL?

OpenGL consists of several pieces: a rendering library, a utility toolkit, an X11 Window System integration suite, and a networking protocol. The rendering library has the following capabilities:

- Basic 3-D rendering features
  - Gouraud shaded points, lines, and polygons
  - Matrix and attribute stacks
  - Per-vertex Phong lighting
  - Depth (Z) buffering
  - Display list creation and manipulation facilities
- Anti-aliased points, lines, and polygons
- Subpixel accurate point, line, and polygon rasterization
- User-defined clipping planes
- Texture mapping support
  - Nearest, linear, bilinear and trilinear MIPMAP interpolation
  - Texture decaling, modulation, and blending
  - Automatic texture coordinate generation for environment mapping
- Fog and atmospheric effects
- Stencil buffer (useful for constructive solid geometry, interference, mirror reflection, and shadow algorithms)
- Accumulation buffer (useful for image post-processing, motion blur, and depth-of-field effects)
- Alpha buffer (useful for transparency algorithms)

The OpenGL Utility GLU toolkit provides the following additional features:

- Non-Uniform Rational B-Spline (NURBS) support
- Basic disk, cylinder, sphere, and quadric capabilities
- Concave and multicontour polygon tessellation capabilities

The OpenGL rendering library is integrated with the X11 Window System using the standard X11 Extension mechanism. This means OpenGL has a network protocol that makes it network transparent. The GLX library provides the following functions that integrate the rendering library with X11:

Aug. '92
p.4

Linas Vepstas

Chandrasekhar Narayanaswami

[1] Some familiarity with GL, the X Window System®, and Motif would be useful in reading this article. Books about OpenGL—the *OpenGL Reference Manual* (ISBN 0-201-63276-4) and the *OpenGL Programming Guide* (ISBN 0-201-63274-8)—are available from the Addison-Wesley Publishing Company. An active USENET newsgroup, `comp.graphics.opengl`, handles questions and discussions about OpenGL. The Frequently Asked Questions (FAQ) posting on that newsgroup provides a solid introduction to OpenGL.

- Creation and binding of rendering contexts to windows
- Window creation aids through the Extended Visual types mechanism
- Synchronization of OpenGL with X11 data stream
- Renderer access to X11 fonts
- 3-D widgets, subclassed from Core and XmPrimitive

Both C language and FORTRAN bindings contain these libraries.

OpenGL—A Descendent of GL

OpenGL differs from GL in several ways. The API has been cleaned up, with obsolete, poorly architected, and conflicting functions removed. All subroutines now start with the two-letter prefix "gl", which avoids name-space clashes with other APIs. Several new features are supported in OpenGL that were not available in the AIX implementation of GL, including texture mapping, antialiased polygons, and stencil and accumulation buffers. But possibly the biggest advance is that the level of support for each of these functions has been standardized by the OpenGL ARB. Furthermore, the use of the OpenGL trademark is contingent upon passage of a conformance test suite.

OpenGL retreats from some of the positions taken by GL. OpenGL does not have any of the windowing functions of GL; modern applications are expected to use X11 and Motif, or other windowing systems as they become available. In recognition of the difficulty of using these window systems for developing simple prototype code, OpenGL ships with an auxiliary toolkit library, detailed below. To facilitate Motif programming, an OpenGL widget has been defined.

GLX—Integration with X11

Special care has been taken to integrate OpenGL with the X Window System in a manner that is suitable and appropriate for both systems. Little in the way of what is familiar to X11 programmers was changed. The notion of X11 visuals was extended to accommodate the more complex concept of a frame buffer, which is used in OpenGL.

Extended Visual Types

OpenGL is integrated with the X Window System by extended X11 visuals. X11 is not normally affected by the sophisticated frame buffer types, such as double-buffers, depth buffers, stencils, alpha, accumulation buffers, and stereo windows,

A Visual and a Window

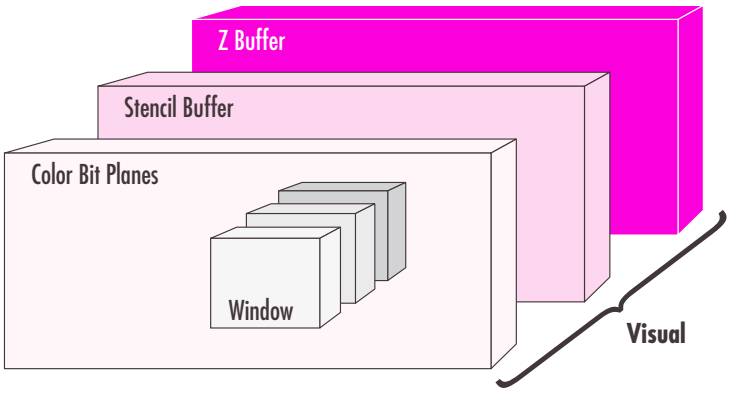

Figure 1. A visual is a class of which a window is an instance

that are interesting to OpenGL applications and needed by OpenGL to be able to draw.

The notion of the X11 visual was extended to allow the application to choose the frame buffer configuration interesting to it. X11 requires that a visual be specified when a window is created (a *visual* is a class of which a window is an instance); the visual type of a window cannot be changed after the window has been created (see Figure 1). Since X11 does not recognize extended visual types, the `glXGetConfig()` routine can provide access to the additional information (numbers and types of associated buffers).

The first step in an OpenGL program is to select the appropriate visual. To choose a visual manually, list all visuals with `XGetVisualInfo()`, and then browse through them with `glXGetConfig()`. Another way is to use the utility `glXChooseVisual()`, which attempts to find the best match to a list of desired buffer attributes. After selecting a visual, create a window with the `XCreateWindow()` subroutine.

Overlay Levels

One notable attribute of extended visuals is `GLX_LEVEL`.The *level* of a visual indicates where a window will be created: in the overlay, the main frame buffer, or the underlay. The usefulness of overlays to avoid exposure and redraw events is well known to video game manufacturers as well as traditional GL programmers. Introducing the level concept adds new flexibility.

Unlike overlays in GL, creating a window at a particular level does not automatically reserve pixels in other layers above and below. Independent windows can be created separately in different layers; can be moved without affecting one

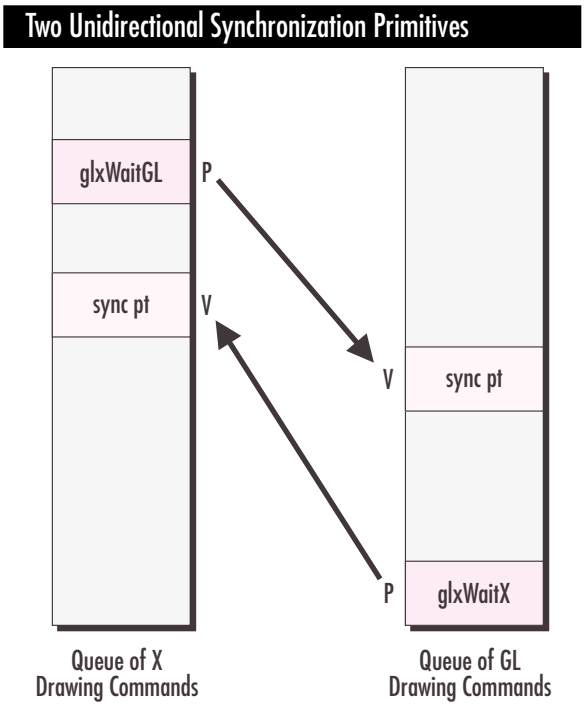

Figure 2. Two unidirectional synchronization primitives

w, with the new
els of processor
formance, it is
sible to imple-
nt 3-D function
irely in software.

another; and can receive mouse, keyboard, and exposure events just as expected for distinct windows. Not only can pop-ups and dialogs be placed in the overlays, but potentially the root window of the X Server (and thus the rest of the X Server) can be placed in the overlays. On a midrange or high-end machine supporting overlays, this makes it possible to drag desk items, such as the `xclock` client, across a 3-D window without destroying the contents of the 3-D window (and causing an expose event to be issued).

Graphics Contexts

The GL notion of graphics state or context has been crystallized into the OpenGL concept of the *GLXContext*. Contexts can be created and manipulated independently of a window; all the OpenGL state is contained entirely in and travels with the context, not the window. OpenGL contexts are created with `glXCreateContext()` and must be bound to a window with `glXMakeCurrent()` before OpenGL can be used to draw. A different window/context pair can be made current with `glXMakeCurrent()` at any time (except between a `glBegin()/glEnd()` pair).

Contexts are not permanently associated with windows as they were in GL, but they can be attached to and detached from different windows at the programmer's will. The only restriction on using the contexts is that a context must have been created with the same visual type as the window with which it will be made current. Not all X11 visual types support OpenGL contexts; some base X11 visuals may not. The `glXGetConfig()` subroutine, together with the `GLX_USE_GL` token, will indicate whether an OpenGL context can be created for the indicated visual type.

Synchronization

Using extended visuals with traditional X11 for creating and manipulating windows implies that it is possible to mix X11 drawing with OpenGL drawing in the same window, an operation that was previously considered dangerous and was unsupported in GL. Two ordering primitives, `glXWaitGL()` and `glXWaitX()`, are provided to enforce a relative ordering between the OpenGL and X11 drawing streams, as shown in Figure 2. There are also queue flushing primitives for each individual stream—`glFlush()` and `XFlush()`—as well as queue flushing primitives—`glFinish()` and `XSync()`—that block until the queues are drained.

Soft3D on IBM Platforms

Traditionally, 3-D graphics requires hardware acceleration for an acceptable level of performance. Now, with new levels of processor performance being set, it has finally become feasible to implement 3-D function entirely in software. As a result of IBM implementing Soft3D, OpenGL is available in low-end configurations with remarkable performance where 3-D has not been previously possible.

In particular, Soft3D has made OpenGL possible on the RISC System/6000 Models 220 and 230 native graphics frame buffers, Gt1 and Gt1x, and also on the PowerPC-based Model 250 native frame buffers GXT100 and GXT150. As with AIX 3.2.5, these platforms expose one PseudoColor (color index) and one TrueColor (RGB) visual that support OpenGL.

These platforms also highlight one of the difficulties of providing 3-D on low-cost, low-end machines: most GL programmers expect hardware overlay planes and multiple hardware colormaps. None of these graphics adapters have overlay planes, and only the GXT150 has more than one hardware colormap (it has four). The absence of overlay planes forces applications to perform all drawing in the main frame-buffer bitplanes. The absence of multiple colormaps can potentially lead to a more severe aesthetic problem, explained below.

Most 3-D applications, and GL applications in particular, generally perform better in RGB mode (for example, to draw into a TrueColor visual) because that is the natural colorspace for lighting and shading. By default, the AIX X11 server (and Motif Window Manager) runs in color index mode. On adapters with only one hardware colormap, there is competition for that colormap.

Whenever an OpenGL TrueColor (RGB) window is on the screen and the X Server was started in its default PseudoColor mode, color flashing will occur. Either the Motif Window Manager (mwm) and most of X11 will look wrong (have the wrong colors) when the RGB colormap is installed, or the OpenGL TrueColor window will look wrong. Although the colormap "focus" policy is configurable in mwm, most users set up their systems so that a window's colormap is installed when the cursor is in the window; thus, the screen will "flash" colors when the cursor is moved through an RGB window.

There are two ways to solve this problem. The X Server can be started with a TrueColor root window, making TrueColor the default visual for most child windows, and therefore for most applications. This can be done by starting the X server with the `-cc` and `-d` flags. (Do not confuse the `-d` flag for X with `-d` for `startx`; the former flag indicates the depth, the latter indicates the display.) The solution does have one problem: many simple X11 clients inherit the visual from the root window and assume that it is PseudoColor. These clients will not run well or look good when the root window visual has been changed—this is particularly true of freebie and demo clients that are found on the net. A more solid resolution is to avoid colormap starvation altogether: use the GXT150.

The Soft3D implementation of OpenGL is also available on the PowerGt3 and PowerGt4 family of graphics adapters. OpenGL is not supported on the GTO and the 3DCGP, primarily because these adapters lack TrueColor visuals. Although these adapters have overlay planes, these planes are not exposed through X11, which makes them unavailable to OpenGL.

The Auxiliary Library or Toolkit

The OpenGL library does not include routines to create and manipulate windows, making it independent of any window system or operating system. Similarly, routines for reading events from the keyboard or mouse are also not included. Since these routines are necessary to write any useful program that interacts with the user and draws on the screen, routines to manipulate windows and process events are provided in a separate auxiliary library. These include routines to initialize and destroy a window, to specify the type of visual needed for the window, to size and position the window on the screen, and to process input events from the keyboard and the mouse.

The auxiliary library also contains routines that are frequently used in computer graphics and geometric modeling to draw some simple three-dimensional objects, such as a sphere, a cube, a tetrahedron, other polyhedra, a torus, and a teapot. In addition, there are two functions to control program flow. The `auxIdleFunc()` routine specifies the function to be executed if no other events are to be processed. Similarly, the `auxMainLoop()` routine specifies the function to be invoked when the window needs to be updated.

For details about the auxiliary library, see *OpenGL Programming Guide* (ISBN 0-201-63274-8).

The OpenGL Widget

The Xt Intrinsics, along with the Motif toolkit, provide one of the most modern window and input management programming interfaces for AIX. Because the glX routines alone are hard to use with the Intrinsics, two OpenGL widgets have been defined to serve as Xt-based wrappers for the glX functions. One widget, the `glwMDrawingAreaWidgetClass`, derives from the Motif Primitive class. The other widget (whose class name `glwDrawingAreaWidgetClass` differs from the other by only the letter M) is almost identical, except that it derives from the Intrinsics Core class. This widget is provided for X toolkit users who are not using Motif.

Both widgets provide a resource `GLwNvisualInfo` that can be used to specify the visual to use when realizing the widget. In addition, these widgets provide a set of resources that closely resemble the attributes found in the extended visual type: the desired sizes of the color, alpha, stencil, depth and accumulation buffers, and so on. If the `VisualInfo` resource is not specified, the widget will use the latter resources together with `glXChooseVisual()` to find the closest matching visual to use in the widget.

Figure 3 shows an example program in OpenGL.

The Utility Library

The OpenGL Utility (GLU) library contains functions for the following tasks:

```
/*
* This program can be compiled as follows:
* cc -o example example.c -lGL -lGLU -laux -lXm -lXt -lX11
-lm
*/
#include <GL/gl.h>
#include <GL/aux.h>
/*This subroutine draws a red square and a blue wire frame
cube. */
void DrawStuff (void) {
  /* Define the background color to be black */
  glClearColor (0.0, 0.0, 0.0, 0.0);
  /* Clear the window to the background color */
  glClear (GL_COLOR_BUFFER_BIT);
  /* Initialize the modeling matrix */
  glLoadIdentity();
  /* Set the default color of primitives to red */
  glColor3f (1.0, 0.0, 0.0);
  /* Draw a red square */
  glBegin (GL_LINE_LOOP);
         glVertex3f (-0.5, -0.5, 0.0);
         glVertex3f (-0.5, 0.5, 0.0);
         glVertex3f (0.5, 0.5, 0.0);
         glVertex3f (0.5, -0.5, 0.0);
  glEnd ();
  /* Rotate by 60 degrees around axis (0.2, 0.3, 0.3) */
  glRotatef(60.0,0.2, 0.3, 0.3);
  /* Set the default color of primitives to blue */
  glColor3f (0.0, 0.0, 1.0);
  /* Draw the wire frame outline of a cube of side 0.5 in
blue */
  auxWireCube (0.5);
  /* Bring the back buffer to the front, making the drawing
visible */
  auxSwapBuffers ();
}
static void Reshape(int width, int height)
{
   glViewport(0, 0, (GLint)width, (GLint)height);
}
int main (int argc, char *argv[])
{
    /* Window size is 400X600, located at (0, 0) */
    auxInitPosition(0, 0, 400, 600);
    /* Ask for a double-buffered, TrueColor window */
    /* Rendering commands go indirectly through the X
protocol */
    auxInitDisplayMode (AUX_DOUBLE|AUX_RGBA|AUX_INDIRECT);
    /* Open the window */
    if (auxInitWindow ("Example Program") == GL_FALSE)
```

Figure 3. A simple OpenGL program

- Create several types of matrices, such as picking, perspective projection, and orthographic projection matrices
- Convert between world and window coordinates
- Manipulate images for texture mapping, including scaling textures and building MIPMAPs (texture images at varying resolutions)
- Tessellate polygons
- Render disks, cylinders, and spheres
- Handle NURBS curves and surfaces
- Convert numeric error codes into more meaningful textual error messages

OpenGL Tips for GL Programmers

The following are tips and hints for OpenGL programmers:

- The Soft3D implementation of OpenGL exposes only double-buffered visuals. Since `glXChooseVisual()` and the widgets search for single-buffered visuals by default, the search will fail unless double buffering was explicitly requested. Single-buffered visuals can be emulated by calling `glDrawBuffer(GL_FRONT)`.
- When porting GL code that uses `window()`, `ortho()`, or `perspective()` to the OpenGL calls `glFrustrum()`, `glOrtho()`, and `gluPerspective()`, be sure to call `glMatrixMode(GL_PROJECTION)` and `glLoadIdentity()` first. The GL routines implicitly load the projection matrix; the OpenGL routines multiply the current matrix on the current matrix stack. Therefore, the projection stack must be explicitly specified and initialized in OpenGL.
- When porting GL code that uses `scale()`, be sure to enable normal vector renormalization with `glEnable(GL_NORMALIZE)`. In OpenGL, unlike GL, renormalization is not automatically enabled.
- Watch for `#define` statements that are similar, such as `GL_LINE` and `GL_LINES`. The first is used with `glPolygonMode()`, while the second is intended for `glBegin()`. One way to catch such mistakes is to check the error code with `glGetError()` after each call.
- The Soft3D implementation makes extensive use of buffering. Performance can be considerably improved by specifying as many vertices as possible between `glBegin()/glEnd()` pairs. Rather than drawing independent lines and triangles with distinct `glBegin()/glEnd()` pairs, use the `GL_LINES`, `GL_TRIANGLES`, and `GL_QUADS` tokens.

Commands that change state will cause buffers to flush, and indirectly can lead to greater cache

misses. Performance can be improved by avoiding unnecessary state changes. Redundant state change commands should also be avoided. Beware of the object-oriented system that attempts to support very low-level primitives, such as lines and polygons. These systems tend to redundantly re-specify the color or material for each line and polygon, despite the fact that it has not changed since the last line or polygon.

- Neither Feedback mode nor Select mode examine, use, or consider the state of most attributes. Not setting or changing those attributes while in Feedback or Select mode will save CPU cycles.
- It is important to look for invalid data. Zero-length vectors and NaN (Not a Number) colors and coordinates will spring traps that can slow processing significantly.
- In the Soft3D implementation, certain common code paths have been tuned to give maximum performance. In particular, the depth test function `GL_LESS` will provide better performance than any other depth comparison function. GL code that uses `LEQUAL` comparisons should probably use `GL_LESS`.

Avoiding the alpha channel will significantly improve performance. This can be done by masking out the alpha channel with `glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_FALSE)`. Neither of the most commonly used blend functions, `glBlendFunc (GL_SRC_ALPHA` and `GL_ONE_MINUS_SRC_ALPHA)`, require or use the alpha buffer.

Care has been taken to optimize multiple light sources. Since the performance penalty for multiple light sources is slight, this should not discourage their use.

Summary

OpenGL is here to stay. It provides a dramatically improved set of features over IBM GL 3.2, and, with Soft3D, provides a new low-cost implementation. Having evolved over many years, it has become a mature API. Since it is governed by the standards body OpenGL ARB, it is stable. The OpenGL Conformance Test suite provides a way to test the uniformity and quality of implementations. Finally, the ARB has linked the use of the OpenGL trademark to passing the Conformance suite, guaranteeing the rich and stable feature set across all implementations.

OpenGL has dramatically improved feature compared to IBM GL 3.2, and with Soft3D, provides a new low-cost implementation.

***Linas Vepstas**, IBM Corporation, RISC System/6000 Division, 11400 Burnet Road, Austin, TX 78758. Internet: linas@inner-door.austin.ibm.com. Dr. Vepstas is one of the architects supporting graphics in the IBM Advanced Workstations and Servers group. He has been working with the GL 3-D graphics programming interface since 1989, contributing to software, microcode, and hardware design. He currently represents IBM on the OpenGL Architecture Review Board. He holds an AB degree from the University of Chicago and a PhD in Physics from the State University of New York (SUNY) at Stonybrook.*

***Chandrasekhar Narayanaswami**, IBM Corporation, RISC System/6000 Division, 11400 Burnet Road, Austin, TX 78758. Internet: chandra@austin.ibm.com. Dr. Narayanaswami works on the design of flexible architectures for high-performance graphics and multimedia. He has a BTech in Electrical Engi-*

The IBM International Seminar

The IBM International Seminar is scheduled for March 1 and 2, 1994 at the Santa Clara Marriott, Santa Clara, California. The target audience includes software engineers and technical managers who design and develop software for international markets. The seminar will cover both National Language Support (NLS) and Double-Byte Character Set (DBCS) considerations.

A host of technical speakers from the major IBM laboratories and several independent consultants will provide detailed programming design guidelines for internationalizing software. Topics range from a description of worldwide writing systems and national language-dependent functions to translation and NLS specifications. Platform-specific information about AIX, OS/2, DOS, AS/400, and System/390® is given in separate sessions. Four demonstrations enhance the presentations, and the focus is on current topics such as Unicode and POSIX™.

The $500 fee includes meals, as well as the opportunity for roundtable discussion groups in the evening. Ample time is allowed for individual dialog with the instructors.

For further information and to receive an invitation, contact Dory Miller at (203) 262-4034 or fax (203) 262-2141.