Introduction

For OpenGL For Macintosh, Version 1.0

Apple Computer, Inc.
Technical Publications
April, 1999

Contents

About This Book

This reference book is designed and written for Mac OS developers who are working with or building applications using OpenGL. This guide introduces the features of OpenGL for Macintosh, version 1.0. This release of the software corresponds to version 1.1 of OpenGL from Silicon Graphics, Inc. (SGI).

This book is intended for use in conjunction with the following documents:

- *OpenGL for Macintosh AGL Reference*, which describes AGL, Apple's implementation of the main OpenGL library. This document is included as part of the OpenGL for Macintosh Software Developer Kit (SDK).
- *OpenGL for Macintosh Programmer's Guide*, which provides an overview of how to set up an OpenGL programming project. This document is included as part of the OpenGL for Macintosh Software Developer Kit (SDK).
- *OpenGL Reference*, which describes GL, the main OpenGL library. This document is available at www.opengl.org.
- *OpenGL GLU Reference*, which describes the OpenGL Utility Library, containing graphical extensions based entirely on GL functions. This document is available at www.opengl.org.
- OpenGL GLUT Reference, which describes the OpenGL Utility Toolkit, a standard API for performing operations associated with a windowing environment. This document is available at www.opengl.org.

Conventions Used in This Book

This book provides various conventions to present information. Words that require special treatment appear in specific fonts or font styles. Certain types of information, such as parameter blocks, use special fonts so that you can scan them quickly.

Special Fonts

All code listings, reserved words, and the names of actual data structures, constants, fields, parameters, and functions are shown in Letter Gothic (`this is Letter Gothic`).

Words that appear in **boldface** are key terms or concepts that are defined in the glossary.

Types of Notes

There are several types of notes used in this book.

Note
A note like this contains information that is interesting but not essential to an understanding of the main text. ◆

IMPORTANT
A note like this contains information that is essential for an understanding of the main text. ▲

▲ **WARNING**
A warning like this indicates potential problems that you should be aware of as you design your software. Failure to heed these warnings could result in system crashes or loss of data. ▲

Development Environment

OpenGL for Macintosh is implemented as a set of shared libraries. As such, it can be used by any compiler for PowerPC that is compatible with Mac OS.

Code listings in this book are shown in ANSI C. They suggest methods of using various functions and illustrate techniques for accomplishing particular tasks. Although most code listings have been compiled and tested, Apple Computer Inc., does not intend for you to use these code samples unmodified or untested in your application.

System Requirements

OpenGL for Macintosh supports the ATI RAGE-2, RAGE Pro, and RAGE 128 graphics cards shipped in iMac computers and 1999 Power Macintosh G3 minitower computers. At this release no other computers or graphics cards are supported

Software Development Kit (SDK)

The OpenGL for Macintosh Software Development Kit (SDK) includes the OpenGL libraries, API documentation, and example source code. It is available for download from http://developer.apple.com/opengl/ and provided on CD-ROM in the Apple Developer Connection monthly mailing program (see http://www.apple.com/developer/programs/ for membership information).

OpenGL Overview

OpenGL is an open, cross-platform three-dimensional (3D) graphics standard with broad industry support. OpenGL was developed by Silicon Graphics, Inc. (SGI), based on SGI's IRIS GL (Graphics Library), first released in 1992. As an open standard, it is now controlled by the OpenGL Architecture Review Board (ARB), a consortium whose members represent many of the significant companies in the computer graphics industry.

OpenGL has a number of benefits for you as a developer:

- True standard: Each implementation of OpenGL must adhere to the OpenGL specification and pass a set of conformance tests.
- Platform independence: It is a cross-platform system, allowing you to leverage your Mac OS development efforts for other platforms. OpenGL drivers free you from designing for specific hardware features and ensure consistent presentation on any compliant hardware/software configuration.
- Industry acceptance: In addition to OpenGL for Macintosh, there's already an OpenGL implementation for OS/2, Windows 95, Windows NT, Linux, OPENStep, and BeOS.
- Controlled evolution: The OpenGL GLU extension mechanism allows you to take advantage of hardware innovations early. Successful innovations are then incorporated into the core OpenGL API as appropriate. And the standard enforces backward compatibility to extend the usable life of your application.
- Performance: OpenGL uses available 3D acceleration hardware features effectively to improve rendering speeds.
- Full feature set: The OpenGL API includes over 250 graphics routines, providing geometric and raster primitives, RGBA or color index mode, display list or immediate mode rendering, and viewing and modeling transformations. In addition, it provides capabilities like lighting and shading, hidden surface removal (for depth buffering), alpha blending (for

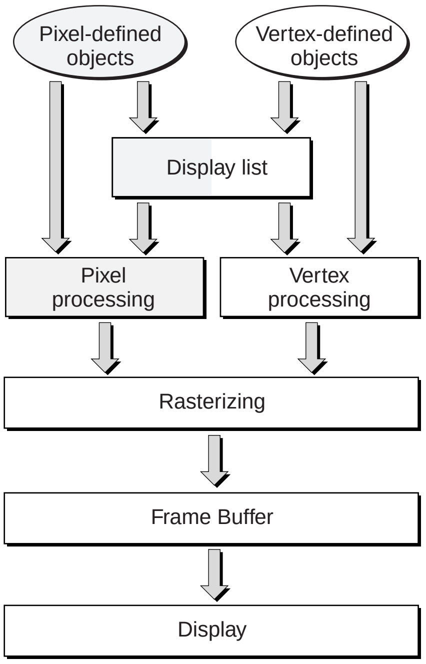
translucency), anti-aliasing, texture mapping, atmospheric effects (for example, fog, smoke, and haze), feedback and selection, stencil planes and accumulation buffering.

- Efficiency: OpenGL routines help to keep your application small.

The OpenGL Model

The block diagram in Figure 1-1 shows the basic components of OpenGL and their relation to your application and to each other.

Figure 1-1 OpenGL Architecture



Your application will typically interface directly with the OpenGL library (GL), the OpenGL Utility Library (GLU), and the OpenGL Utility Toolkit (GLUT).

The GL is a low-level modular API that allows you to define graphical objects. It includes the core functions that are common to all OpenGL implementations, as mandated by the OpenGL specification. It provides support for two fundamental types of graphics primitives: objects defined by a set of vertices, such as line segments and simple polygons, and objects that are pixel-based images, such as filled rectangles and bitmaps. Support for complex custom graphical objects is not provided by GL.

Here are some of the features provided by the GL:

- 3D transformation
- **accumulation buffering**
- **alpha blending**
- **antialiasing**
- **atmospheric effects** (haze, fog, and smoke)
- clipping and **culling** (hidden surface removal)
- **color index mode**
- **edge flags**
- geometric (vertex-defined) and raster (pixel-defined) primitives
- **Gouraud shading**
- **material lighting**
- **Phong lighting**
- RGBA and **color index modes**
- **stencil planes**
- **texture mapping**
- viewing and modeling transformations

The OpenGL Utility Library (GLU) provides advanced features by combining functions in the GL library. Consequently, you can be sure that the GLU functions are supported on all conforming implementations of OpenGL. Some GLU features are complex polygon (including quartics) creation and handling, b-spline curve (NURBS) processing, image scaling, and tesselation.

The Graphics Library Utility Toolkit (GLUT) provides a standard API for performing operations associated with a windowing environment such as display, redraw, and event handling.

In addition to the GL, GLU, and GLUT libraries, the OpenGL graphics processing system consists of the run-time engine and the rendering (or rasterizing) software and firmware required to create high-quality display images of the 3-dimensional objects you define.

OpenGL for Macintosh

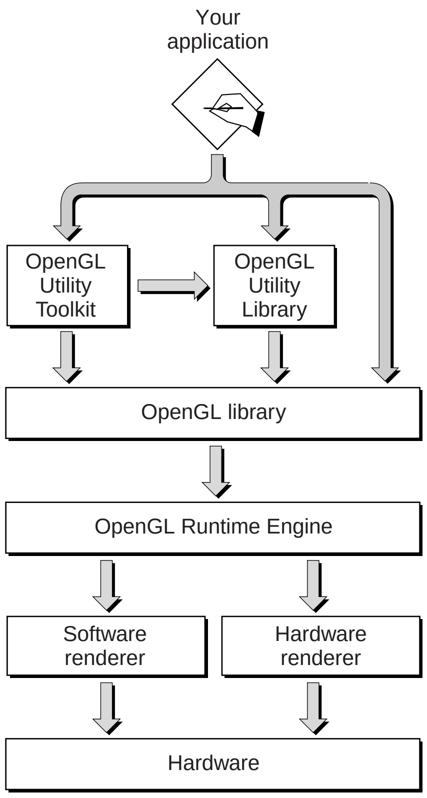
The OpenGL for Macintosh runtime engine is implemented in a set of shared libraries that reside in the Extensions folder. These libraries include the rendering software for Mac OS computers. You can also license the runtime engine to install with your application. Rendering firmware is specific to the target acceleration hardware.

OpenGL Operation

The main purpose of OpenGL is to accept objects defined as three-dimensional and process them for realistic display in two dimensions, typically on a computer screen. To display an object with OpenGL, your application first creates a drawable (an entity into which objects can be drawn, such as a window or a screen image buffer). The application then establishes a rendering context and associates the context with the drawable. Upon confirming that association, OpenGL sets up required graphics buffers (depth buffer, alpha buffer, stencil buffer, and accumulation buffer). The application can then issue OpenGL commands to be interpreted and drawn into the drawable.

To maximize performance, OpenGL processes each graphical object in an ordered sequence of operations much like a Unix pipeline. Figure 1-2 illustrates this processing pipeline.

Figure 1-2 OpenGL graphics processing pipeline



Graphic objects and any operations you have specified can be processed immediately upon entering the pipeline or added to a display list for later use. You might think of a display list as an OpenGL macro or subroutine; it stores a set of operations that can be run at any time, can be reused, and can be combined with other display lists to create complex graphical objects not provided in the core library. Display list operations are precompiled so that they are ready for use when needed; consequently, they are stored as pixel data.

Operations not destined for a display list undergo processing immediately. Vertex processing starts with modeling objects from their 3D vertices and applying transformations such as scaling, positioning, and orientation to those objects. The effects of color, lighting, camera angle, and texture are also included in processing at this stage. Another part of the vertex processing is projection—the process by which a 3D object is mapped onto a 2D display space.

The rasterizing process converts the 2D image data (including objects originally pixel-defined) into an image fragment: a set of values in a format capable of being loaded into the frame buffer. Prior to being placed into the frame buffer, however, image fragments can be modified by a variety of logical operations such as masking and color blending.

Finally, loading an image fragment into the frame buffer initiates a series of tests and processes to ensure that the fragment is presented appropriately relative to the other data already in the frame buffer. For example, a fragment may represent a translucent object, and would thus be adjusted so that objects behind it are visible. When all the tests and data conversions are performed, the pixel data is loaded into the frame buffer and is ready for display.

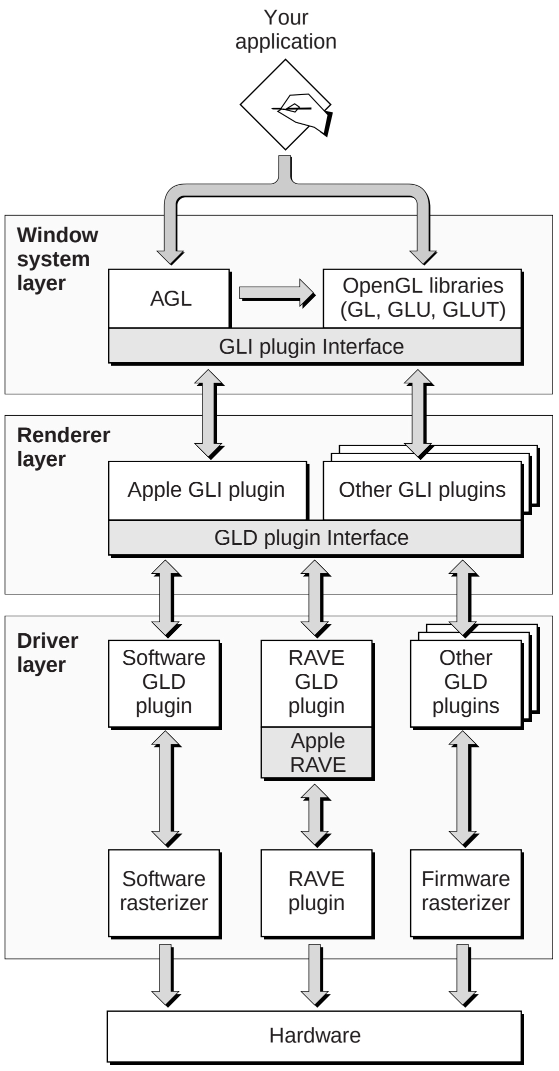
Architecture of the Runtime Engine

OpenGL for Macintosh is a set of shared libraries (installed in the Extensions folder) that implement the OpenGL runtime engine.

The Mac OS display model is more complicated than most, in that it supports a display space that can consist of multiple dissimilar monitors. Furthermore, those monitors can be driven by different graphics cards with different capabilities. Adding to this complexity is the possibility of multiple rendering libraries that can drive the same graphics cards.

To accommodate this complexity, OpenGL for Macintosh is segmented into three well-defined layers: a window system layer, a renderer layer, and a driver layer (as shown in Figure 1-3). This segmentation allows for plug-in interfaces to both the window system level and the renderer level. These plugin interfaces offer flexibility in software and hardware configuration while adhering to the OpenGL standard.

Figure 1-3 OpenGL for Macintosh layers



Window System Layer

Most applications will need to interact with only the window system layer. This layer includes GL (the main OpenGL library) and the Apple GL library (AGL), which contains the implementation of OpenGL functions and commands specific to the Mac OS windowing system. Operations performed in this layer include pixel format selection, context creation / destruction, and drawable and buffer swapping operations.

Rendering Environment Support

As described above, the Mac OS supports a variety of graphics accelerator cards with varying capabilities, while on some Mac OS systems there is no graphics acceleration hardware. It is also possible for multiple renderers, each with different capabilities or features, to be used to drive the same graphics hardware. The GLI plugin interface is provided to manage these complications. It allows easy installation and removal of multiple rendering libraries (without extensive system rebuilding) by moving them (respectively) into and out of the Extensions folder.

For peak performance, the GLI plugin interface enables AGL to dynamically select the best rendering library for the current rendering task—transparently to your application.

When your application starts, AGL registers all OpenGL-compliant rendering libraries it finds in the Extensions folder. Then, when the application specifies a pixel format, AGL loads the best renderer for that format and passes to it all OpenGL rendering commands issued by the application. If the application then specifies a different pixel format for another window, AGL will repeat the selection process and use the most capable renderer for the new window. In this way, your application may benefit from the use of multiple OpenGL renderers without even being aware that more than one renderer is available.

If your application requires more control of renderer selection, AGL helps in several ways. Your application can specify preferences as to which capabilities are most important, forcing selection of renderers with those capabilities.

To ensure use of a specific renderer, your application can request that renderer by its unique renderer ID. Moreover, applications with very specific rendering requirements can use the AGL mechanism for feeding back detailed information about the full capabilities of all renderers on the system.

Multiple Monitor Support

AGL also transparently manages renderers across multiple monitors. As mentioned above, the Mac OS windowing system supports a large virtual screen comprising multiple monitors. For example, a user can drag a window from one monitor to another despite the fact that their display capabilities may be entirely different and that they may be driven by dissimilar graphics cards with dissimilar resolutions and color depths.

When an application chooses a pixel format on a multi-monitor system, AGL attempts to find for each monitor a renderer that provides the requested pixel format at the highest performance on that monitor. A particular renderer may be able to drive multiple monitors with a single OpenGL rendering context, or it may have to create separate contexts to support separate graphics cards. AGL queries each renderer about its capabilities and attempts to use it as efficiently as possible. If necessary, AGL then creates a multiple-monitor pixel format that defines the relationship between the renderer and the monitor it will drive. Your application uses only this one pixel format. AGL dynamically switches between renderers when part of an OpenGL window crosses the screen boundary between one monitor and another. There is no performance impact from multiple monitor support while the entire graphics window is displayed on one monitor.

Renderer Layer

The renderer layer implements the OpenGL command compilation and execution mechanism as well as the renderer control commands for pixel format selection, context creation / destruction, and drawable and buffer swapping operations. This layer consists of the GLI plugin interface and one or more GLI plugin renderers (which may have different software and hardware support capabilities). The GLI plugin interface supports third party plugin renderers, potentially easing the task of porting existing drivers (for example, Windows 98 / NT OpenGL ICD drivers) to the Power Macintosh platform. Other possible plug-in renderers include ray tracers, file IO managers, debuggers, and print drivers.

GLI plugins should be installed in the Extensions folder.

Driver Layer

The OpenGL driver layer implements the hardware-specific functions for performing pixel format selection, context creation / destruction, drawable, buffer access, texture management and buffer swapping operations. This layer consists of the GLD plugin interface and one or more GLD plugin drivers (which may have different software and hardware support capabilities). The

GLD plugin interface supports third party plugin drivers, allowing third party hardware vendors to take advantage of current driver technology.

Glossary

2D Two-dimensional. See also planar.

3D Three-dimensional. See also spatial.

accelerator See graphics accelerator.

accumulation buffer A buffer in which multiple rendered frames can be composited to produce a single image.

aliasing The jagged edges (or staircasing) that result from drawing an image on a raster device such as a computer screen. Compare **antialiasing.**

alpha blending A process fo using alpha information to create transparent objects.

alpha channel A color component in some color spaces whose value represents the opacity of the color defined in the other components. Compare **ARGB color structure.**

antialiasing The smoothing of jagged edges on a displayed shape by modifying the transparencies of individual pixels along the shape's edge. Compare **aliasing.**

API See **application programming interface.**

application programming interface (API) The total set of constants, data structures, routines, and other programming elements that allow developers to use some part of the system software.

Architecture Review Board (ARB) An independent consortium that controls the evolution of OpenGL. Member s currently include Digital Equipment Corporation, Evans and Sutherlin, Hewlett-Packard, IBM, Integraph, Intel, Microsoft, and Silicon Graphics.

B-spline curve A curve that passes smoothly through a series of control points.

bitmap A two-dimensional array of values, each of which represents the state of one pixel.

constant shading A method of shading surfaces in which the incident light color and intensity are calculated for a single point on a polygon and then applied to the entire polygon. Compare **Gouraud shading, Phong shading.**

culling Ignoring hidden image datato reduce the amount of time required to render a model.

depth buffer TBD.

display list A named list of OpenGL commands that can be precompiled for faster execution and possible reuse.

double buffering Building an image in an off-screen buffer prior to display. Used to provide smooth animation of objects.

feedback mode A mode in which OpenGL returns the processed geometric information (colors, pixel positions, and so on) to the application instead of rendering them into the frame buffer.

drawable An entity into which pixel data can be drawn, such as a window, a full-screen buffer, or an off-screen buffer.

frame buffer The buffer in which the final image is prepared and staged for display.

geometric primitive Any of the basic geometric objects defined by OpenGL in the GL library.

Gouraud shading A method of shading surfaces in which the incident light color and intensity are calculated for each vertex of a polygon and then interpolated linearly across the entire polygon. Compare **constant shading, Phong shading.**

graphics accelerator Any hardware device used to increase rendering speed.

image The two-dimensional product of rendering.

material lighting A process by which the color of a point on a surface is computed using the properties of the surface material.

modeling The process of creating a representation of real or abstract objects.

nonuniform rational B-spline (NURB or NURBS) A curve defined by nonuniform parametric ratios of B-spline polynomials. NURB curves can be used to define very complex curves and surfaces, as well as very common geometric objects (for instance, the conic sections).

NURB See **nonuniform rational B-spline.**

NURB curve A three-dimensional curve represented by a NURB equation.

Phong shading A method of shading surfaces in which the incident light color and intensity are calculated for a series of points along each edge of a polygon and then interpolated across the entire polygon. Compare **constant shading, Gouraud shading.**

planar Contained completely in two dimensions (as, for example, a circle). See also **spatial.**

polygon A closed plane figure. See **general polygon, simple polygon.**

projection A method of mapping three-dimensional objects into two dimensions.

rasterization The process of determining values for the pixels in a rendered image. Also called scan conversion.

render To create an image (on the screen or some other medium) of a model.

renderer Software or firmware used to create an image from a view and a model.

rendering The process of creating an image (on the screen or some other medium) of a model. See also **rasterization.**

scale To reposition and resize an object by multiplying the x, y, and z coordinates of each of its points by values dx, dy, and dz.

simple polygon A closed plane figure defined by a list of vertices (that is, defined by a single contour).

stencil buffer A buffer used to mask individual pixels.

tessellate To decompose a curve or surface into polygonal faces.

texture mapping A technique wherein a predefined image (the texture) is mapped onto the surface of an object in a model.

transparency The ability of an object to allow light to pass through it.

vertex A dimensionless position in three- or four-dimensional space at which two or more lines (for instance, edges) intersect, with an optional set of vertex attributes.

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITERS
Robert Beretta, Michael Hinkson, John Stauffer

ILLUSTRATOR
Dave Arrigoni

PRODUCTION EDITOR
Lorraine Findlay