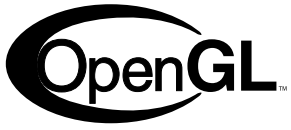

Programmer's Guide

For OpenGL For Macintosh, Version 1.0

Apple Computer, Inc.
Technical Publications
April, 1999

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Contents

About This Book

This book is designed and written for Mac OS developers who are working with or building applications using OpenGL. This guide provides some basic programming tips for writing an application that uses the features of the OpenGL API provided by OpenGL for Macintosh, version 1.0. (This release of the software corresponds to version 1.1 of OpenGL from Silicon Graphics, Inc.)

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

- *OpenGL for Macintosh Introduction*, which provides a conceptual overview of OpenGL features. This document is included as part of the OpenGL for Macintosh Software Developer Kit (SDK).
- *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 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

Setting Up Your Project

OpenGL for Mac OS supports any Mac OS-compatible compiler that can link to shared libraries, such as those from CodeWarrior, MPW, or Symantec. The programming concepts are the same for all three development environments.

As delivered, OpenGL comes with CodeWarrior-compatible libraries for TK, AUX, GLUT and MUI. If you are not using CodeWarrior, it may be necessary to recompile the TK, AUX, GLUT and MUI static libraries.

Creating a New Project

To get started, you must first create a new project. You will need at least 20Mb of free disk space to save the project files.

Adding OpenGL Libraries

Add the following OpenGL libraries:

- OpenGLLibrary
- OpenGLUtility

The following libraries are optional:

- OpenGLMemory
- tk.lib
- aux.lib

- glut.lib

The OpenGLMemory library is needed for modifying the memory functionality of the OpenGL system. Add the tk.lib, aux.lib, or glut.lib library if your application uses TK, AUX, or GLUT, respectively.

Setting Heap and Stack Size

Set the heap size and stack size as appropriate. It is important that you set these sizes large enough initially to run the application. A minimum 3500KB heap and 256KB stack are required for running with hardware acceleration. You can tune the memory size later by running the application and checking the memory usage from the Finder.

Writing an OpenGL Application

When you have completed all the setup steps, you are ready to begin writing your OpenGL application. This section provides some information about how you might want to structure your source files.

Selecting a Pixel Format

When writing an OpenGL application, the first thing to do is select a pixel format. The `aglChoosePixelFormat` selection function queries a set of installed OpenGL renderers for available buffer configurations and hardware capabilities. The results of these queries are scored, and the highest-scoring pixel format for each hardware accelerator and display device is returned. If a renderer is unable to support the requested display device, no pixel format will be returned. A renderer may support more than one display device, but is not required to do so.

You can pass a display device list to `aglChoosePixelFormat` to restrict the devices required to be managed by a pixel format. The default is all devices. A pixel format may specify more than one renderer to attach, depending on the input attributes and display devices to be used. Pixel formats are cached for fast repeated requests. You can adjust cache size by calling aglConfigure (`AGL_FORMAT_CACHE_SIZE`, size). The default value for `AGL_FORMAT_CACHE_SIZE` is 5.

Creating a Rendering Context

With a pixel format selected, you can call `aglCreateContext` to create a rendering context based on the selected pixel format. During the context creation process, one or more renderers are loaded and linked to an OpenGL procedure dispatch table. Once a renderer is loaded, all subsequent load requests reference that loaded copy. When all contexts referencing a rendering are destroyed, the renderer is unloaded. Renderer caching can be achieved by calling `aglConfigure (AGL_RETAIN_RENDERERS, GL_TRUE)`. This allows for fast context creation when accessing a previously used renderer. You can specify a shared context to allow two contexts to share display lists and texture maps. For `aglCreateContext` to succeed in creating shared contexts, both contexts must be created with an identical set of renderers.

Listing 1-1 shows a useful function for creating a double-buffered RGBA context with a z-buffer. The function will create a context that is capable of managing all display devices. Additional attributes could be added for more precise control over the selected renderer. The `AGL_ACCELERATED` attribute could be added to specify that only hardware accelerated renderers be chosen. `AGL_OFFSCREEN` or `AGL_FULLSCREEN` could be added to specify a drawable type other than a window.

Listing 1-1 Creating a rendering context

```
AGLContext CreateOpenGLContext(void)
{
    GLint attrib[5] = {AGL_RGBA,
        AGL_DOUBLEBUFFER,
        AGL_DEPTH_SIZE, 16,
        AGL_NONE    };

    AGLPixelFormat fmt;
    AGLContext     ctx;

    /* Choose pixel format */
    fmt = aglChoosePixelFormat(NULL, 0, attrib);
    if(fmt == NULL) return NULL;

    /* Create an AGL context */
    ctx = aglCreateContext(fmt, NULL);
```

```
        /* Destroy pixel format */
        aglDestroyPixelFormat(fmt);

        return ctx;
    }
```

Destroying a Pixel Format

After you have created the rendering context, you can retain the pixel format for creating additional contexts, or you can destroy it by calling `aglDestroyPixelFormat`.

Preparing and Using a Drawable

To prepare a rendering context for drawing, you must first attach a drawable to the context by calling `aglSetDrawable`. The attached drawable is evaluated for screen ownership information and a renderer capable of managing the current drawable display devices is selected. Once you have attached a drawable to the context, you must call either `aglUpdateContext` or `aglSetDrawable` each time before starting to render if any of the following events affecting the drawable have occurred since the last drawing sequence, thereby invalidating the window contents:

- window drag
- window grow
- window zoom
- Color table modification (only for 256 color RGB mode)

If the drawable is a full-screen or off-screen buffer, or if the window sub-rectangle is not occluded by other windows, the ownership area will be a simple rectangle. OpenGL buffers are created when `aglSetDrawable` is called with a valid drawable. If `aglSetDrawable` returns `GL_FALSE`, the renderer has failed to allocate the drawing buffers. Passing `NULL` for the drawable frees the buffers allocated to the context. This approach increases memory use efficiency, but is not recommended if performance is a priority.

Setting the Current Rendering Context

All OpenGL API commands are dependent on the current context. You call `aglSetCurrentContext` to set the current context—or to switch between contexts , if the application has more than one. Commands are dispatched to a renderer based on the display list mode, compile and / or execute. Although you can call `aglSetCurrentContext` with a valid context before or after a drawable has been attached, commands point to a non-operational function and will not be linked to the renderer dispatch table until a current context with an attached drawable is established.

Listing 1-2 shows a useful function for preparing the OpenGL context for drawing. This function should be called when an event has invalidated the window.

Listing 1-2 Preparing a context for drawing

```
GLboolean StartOpenGLDrawing(AGLContext ctx, CWindowPtr win)
{
    /* Attach the context to the window */
    no_err = aglSetDrawable(ctx, (AGLDrawable) win);
    if(!no_err) return GL_FALSE;

    /* Make context current */
    no_err = aglSetCurrentContext(ctx);
    if(!no_err) return GL_FALSE;

    return GL_TRUE;
```

Initiating Drawing

You initiate drawing by issuing the buffer swapping command `aglSwapBuffers`. Buffer swapping commands are dispatched to the context. It is left to the renderer to update the screen area based on attached drawable. The buffer rectangle or the swap rectangle can be specified by calling `aglSetInteger` and `aglEnable` with `AGL_BUFFER_RECT` or `AGL_SWAP_RECT`. The buffer rectangle and swap rectangle are both specified in OpenGL coordinates and default to the window size.

Listing 1-3 Drawing

```
void Draw(void)
{
    /* Clear buffers */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* Draw */
    glColor3f(1.0, 0.0, 0.0);

    glBegin(GL_POLYGON);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
        glVertex2f(-0.5, -0.5);
    glEnd();

    /* Swap buffer to window */
    aglSwapBuffers(ctx);
}
```

Destroying a Rendering Context

When a context is no longer needed, you should destroy it. Listing 1-4 shows an example of destroying a context.

Listing 1-4 Destroying a context

```
void DestroyOpenGLContext(AGLContext ctx)
{
    /* Destroy context */
    aglDestroyContext(ctx);
```

AGL Color Index Strategies

There are two basic strategies that you can use to control the RGB colors assigned to each color index when using AGL in color index mode: **direct color**

index mode and **translated color index mode** . This section describes the benefits of each mode.

Direct Color Index Mode

When the monitor is in 256-color mode, you can use direct color index mode (the default mode). This mode yields the highest performance because it requires no color translation, but it has two disadvantages:

- The monitor must be set to 256 colors—other color depths do not have an appropriate color table.
- Changes to the color table affect the entire monitor; other windows can be adversely affected.

When a window is attached to a color index rendering context with `aglSetDrawable`, the color table seed of the back buffer is matched to that of the window. With matching color table seeds, color index data is copied directly from the back buffer to the window by `aglSwapBuffers` or `glFlush` with no color translation. To change the RGB values associated with each index, simply alter the color table for the window and call `aglUpdateContext`.

Translated Color Index Mode

If the color table seeds of the back and front buffers are not matched, the color of each pixel is translated as the color indices are copied from the back buffer to the window. To control the color translation, the matching of color table seeds must be disabled and the desired RGB values for each index must be assigned to the color table of the back buffer with `aglSetInteger`. This approach is called translated color index mode.

This approach yields lower performance than direct color mode because each pixel is mapped from a color index to an RGB color with each call to `aglSwapBuffers`, `glFlush`, or `glFinish`, but it can be used without regard for the current monitor depth, and without affecting the contents of other windows.

For example, given three arrays containing the red, green, and blue values to be assigned to each color index, the following function could be used to prepare the color table:

Listing 1-5 Preparing the color table for translated color index mode

```
void setColorMap(GLushort red[256], GLushort green[256], GLushort
blue[256])
{
    GLint i, color[4];

    aglDisable(aglGetCurrentContext(), AGL_COLORMAP_TRACKING);
    for(i = 0; i < 256; i++)
    {
        color[0] = i;
        color[1] = red[i];
        color[2] = green[i];
        color[3] = blue[i];
        aglSetInteger(aglGetCurrentContext(),
            AGL_COLORMAP_ENTRY, color);
    }
}
```

Each component in a colormap entry is given as an unsigned 16-bit integer, with 0 representing minimum intensity and 65535 representing maximum intensity.

This technique offers lower performance than direct color mode because each pixel is mapped from a color index to an RGB color with each call to `aglSwapBuffers`, `glFlush`, or `glFinish`, but it can be used without regard for the current monitor depth, and without affecting the contents of other windows.

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