

Submitted: September 23, 2002



OpenGL Fullscreen Graphics with AGL

by Mark Szymczyk

Abstract: Many games need to take over the entire screen. For Macintosh OpenGL programmers, it can be difficult to find information on how to do this. The majority of the OpenGL tutorials and examples on the Internet have been written for Windows. Most of Apple's OpenGL sample code does its drawing into a regular window. This tutorial will show you how to take over the screen and do it on the Macintosh.

Copyright © 2002 iDevGames

Notice of Rights

All rights reserved. No part of this tutorial may be reproduced or transmitted in any form or by any means, electronic, mechanical, photocopying, recoding, or otherwise, without the prior written permission of iDevGames.

Notice of Liability

The information in this tutorial is distributed on an "As is" basis, without warranty. While every precaution has been taken in the preparation of this tutorial, neither the author nor iDevGames shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instruction contained in this tutorial or by the computer software and hardware products describe herein.

What is AGL?

As you are probably aware, OpenGL is a cross-platform library for drawing 3D graphics, which means that OpenGL code works on any operating system that supports OpenGL. What gets in the way of cross-platform 3D graphics is the fact that every operating system has their own unique way of implementing windows and drawing graphics to the screen. If OpenGL had to worry about how to get graphics to the screen, it would become a huge mess, with functions for each operating system with an OpenGL implementation. Writing cross-platform code would be very difficult.

OpenGL's solution was to create extensions for each operating system that dealt with drawing OpenGL graphics to the screen. By doing this, developers' base OpenGL code would work on all operating systems, and the only platform-specific graphics code they would have to write would be the extension for that particular platform. On the Macintosh, AGL is the extension for drawing into offscreen **GWorlds**, windows, and the screen with OpenGL. In this tutorial, I'm going to show you how to draw fullscreen graphics with AGL as well as show you how to use AGL with **DrawSprocket**.

AGL is not the only way to make OpenGL graphics appear on the screen on the Mac. **GLUT** (OpenGL Utility Toolkit) is a cross-platform library for creating windows, drawing into them, and reading the keyboard and mouse. GLUT does a lot more than AGL; it functions as a substitute for the standard Macintosh window and event handlers as well as doing the things that AGL does. The advantage of using GLUT is that the same set of code works on multiple platforms. The disadvantage of GLUT is that GLUT code runs slower than AGL code. Most OpenGL examples on the Internet use GLUT so you can see how GLUT works by looking at those examples. The availability of GLUT examples on the Internet is the reason I chose to focus on AGL for this tutorial.

If you're using Mac OS X, another solution is to use **CGL**, a low level library that uses Apple's Core Graphics technology. CGL runs only on Mac OS X, and it only runs in fullscreen mode.

Before You Begin

If you are writing a Mach-O Carbon program, you must include the following frameworks in your AGL project:

- `OpenGL.framework`
- `AGL.framework`

If you are writing a CFM Carbon program or a Classic program, you must include the following libraries in your AGL project:

- `OpenGLLibraryStub`
- `OpenGLUtilityStub`

No matter what type of Mac program you write, you must include the following header files to use AGL:

```
#include <gl.h>
#include <agl.h>
```

Going Fullscreen with AGL

You must perform the following steps to draw with AGL:

1. Choose an AGL pixel format.
2. Create an AGL draw context.
3. Set the current draw context to the draw context you just created.

To choose an AGL pixel format, you first create an array of type `GLint` that contains all the features you want your draw context to have. If you look at the AGL documentation, you will notice tons of features that an AGL context can have.

OpenGL Fullscreen Graphics with AGL

by Mark Szymczyk

For a fullscreen AGL draw context, you will want the following features at a minimum:

`AGL_FULLSCREEN`

Tells OpenGL you want a fullscreen context. An important thing to keep in mind is that only Mac OS X supports the `AGL_FULLSCREEN` pixel format.

`AGL_DOUBLEBUFFER`

Tells OpenGL you want a double-buffered context. For a game you want a double-buffered context to avoid flickering. In a double-buffered context, you do your drawing in the offscreen buffer, then transfer it to the screen.

`AGL_RGBA`

Tells OpenGL you want the red, green, blue, alpha color format, which is the color format that Mac 3D accelerators support.

`AGL_ACCELERATED`

Tells OpenGL you want hardware-accelerated drawing. `AGL_NONE` tells OpenGL you have no more features to request. `AGL_NONE` must be the last entry in your array of features. After you fill your array of features, call the function `aglChoosePixelFormat()` to select the pixel format.

The function `aglChoosePixelFormat()` takes three parameters:

`aglChoosePixelFormat()`

A pointer to an array of Macintosh graphics devices. You should pass nil unless you have multiple monitors.

`ndev`

The number of graphics devices you passed in the first parameter. You should pass 0 unless you have multiple monitors.

`attribs`

The feature list we discussed earlier in the section.

Here's an example of choosing a pixel format:

```
GLint attributeList[] = { AGL_RGBA, AGL_ACCELERATED,  
    AGL_DOUBLEBUFFER, AGL_FULLSCREEN, AGL_NONE };  
AGLPixelFormat pixelFormat;  
  
pixelFormat = aglChoosePixelFormat(nil, 0, attributeList);
```

Now that we selected a pixel format, we can use it to create an AGL draw context. The function to call is `aglCreateContext()`, and it takes two parameters. The first parameter is the pixel format we created by calling `aglChoosePixelFormat()`. The second parameter is an AGL context with which you want to share OpenGL display lists. Normally, you will use only one AGL context so you can pass `nil` in the second parameter. The following code shows an example of creating an AGL draw context:

```
AGLContext drawContext;  
drawContext = aglCreateContext (pixelFormat, nil);
```

After creating the context, you can make it the current context by calling the function `aglSetCurrentContext()`, as you can see in the code below:

```
GLboolean success;  
success = aglSetCurrentContext(drawContext);
```

Going in and out of Fullscreen Mode

Once you create a fullscreen AGL context, you can go into fullscreen mode by calling the function `aglSetFullScreen()`. This function takes five parameters:

`ctx`
The AGL draw context.

`width`
The width of the screen in pixels.

`height`

The height of the screen in pixels.

`frequency`

The frequency (refresh rate) of the monitor in hertz.

`device`

The index of the graphics device. For a single monitor system, you can pass 0.

Here's an example of going into fullscreen mode:

```
GLboolean success;  
success = aglSetFullScreen(drawContext, 1024, 768, 60, 0);
```

When the player wants to pause the game, you want to go out of fullscreen mode so the player can access the menu bar. To go out of fullscreen mode, call the function `aglSetDrawable()`. Pass the draw context in the first parameter and nil in the second parameter, and you will be out of fullscreen mode. Here's how the code looks:

```
aglSetDrawable(drawContext, nil);
```

Using AGL with DrawSprocket

Although you can certainly use AGL alone to take over the entire screen, you will probably want to use DrawSprocket with AGL. You cannot create a fullscreen AGL draw context in Mac OS 8 and 9 so if you want your AGL code to run on Mac OS 8 and 9, you must use DrawSprocket. DrawSprocket also makes it easy to do things like change the screen resolution and color depth so you will find it useful even if you are developing a Mac OS X-only game. Using DrawSprocket and AGL together requires the following steps:

1. Create a single-buffered DrawSprocket draw context.
2. Choose an AGL pixel format.
3. Create an AGL draw context.
4. Set the current draw context to the draw context you just created.

When using AGL and DrawSprocket together, you normally attach the AGL draw context to the front buffer of the DrawSprocket context. That's why you should create a single-buffered DrawSprocket draw context; OpenGL does not use DrawSprocket's back buffer. Refer to the DrawSprocket tutorial on iDevGames for more information on creating a DrawSprocket draw context.

Choosing the AGL pixel format when using DrawSprocket is very similar. The main difference is that you may not want a fullscreen AGL draw context. The DrawSprocket draw context is fullscreen so you do not need a fullscreen AGL draw context. On Mac OS 8 and 9, you cannot specify fullscreen AGL draw contexts. If you want your code to run on Mac OS 8, 9, and X, do not use `AGL_FULLSCREEN`. If your game will run only Mac OS X, you have the option to use a fullscreen AGL context, and you probably will want to do so because drawing is faster with a fullscreen AGL context. Creating the AGL draw context and setting the current AGL draw context are exactly the same when using DrawSprocket.

To go into fullscreen mode with AGL and DrawSprocket, you must perform the following steps:

1. Activate the DrawSprocket context.
2. Retrieve the DrawSprocket front buffer.
3. Attach the AGL draw context to the DrawSprocket front buffer.

To activate the DrawSprocket context, call the function `DSPContext_SetState ()`, passing the value `kDSPContextState_Active` in the second parameter, as you can see in the function below:

```
void DrawContext::Activate(void)
{
    OSStatus error;

    error = DSPContext_Reserve(contextRef, contextData);
    error = DSPContext_FadeGammaOut(kDSPEveryContext, nil);
    error = DSPContext_SetState(contextRef,
        kDSPContextState_Active);
    error = DSPContext_FadeGammaIn(kDSPEveryContext, nil);
}
```

```
}
```

To retrieve the DrawSprocket front buffer, call the function `DSPContext_GetFrontBuffer()`, as you can see in the following function:

```
CGrafPtr DrawContext::GetFrontBuffer(void)
{
    CGrafPtr frontBuffer;
    OSStatus error;

    error = DSPContext_GetFrontBuffer(contextRef, &frontBuffer);
    return frontBuffer;
}
```

To attach the AGL draw context to the DrawSprocket front buffer, call the function `aglSetDrawable()` as in the following code:

```
DrawContext context;
AGLContext drawContext;
AGLDrawable window = context.GetFrontBuffer();
aglSetDrawable(drawContext, window);
```

When the player pauses the game, call `aglSetDrawable()` with nil as the second parameter, just like you do when using AGL alone.

Conclusion

I included two sample programs in this tutorial. Both programs take over the screen and draw a colored rectangle. One program does it using only AGL, and the other one uses AGL and DrawSprocket. You should be able to use the programs as a starting point for writing your own OpenGL programs. The programs also use Carbon events and Carbon event timers so you can use them as a guide to using these new Apple technologies.

To learn more about AGL, you have two references. The first is the AGL Reference that comes with the OpenGL SDK that you can download from Apple's Website. The full SDK is a big download (over 40 MB), but you can download a smaller version that contains only the OpenGL libraries, header files, and documentation. The second reference is the OpenGL documentation that comes as part of the Carbon developer documentation. If you have Project Builder, you already have this documentation, if not, you can view it from Apple's developer page (<http://www.apple.com/developer>).

For more information on DrawSprocket, refer to the DrawSprocket tutorial on iDevGames or read Chapter 14 of the book "Mac Game Programming." For more OpenGL programming information, refer to the following sites:

- Apple's OpenGL site (<http://developer.apple.com/opengl/>)
- The OpenGL site (<http://www.opengl.org>)
- Nehe's OpenGL tutorials (<http://nehe.gamedev.net>)