Mac Game Programming Roadmap

Author: Mark Szymczyk
Last Update: October 10, 2007

Mac OS X does not have a set of libraries like Microsoft's DirectX specifically for writing games. The lack of game-specific libraries can be difficult for people new to Mac game development because they don't where to start. This article provides an overview of the technologies you need to learn to write Mac games.

Technology Paths

There are three technology paths you can take to write Mac OS X games. The first path is Cocoa, which is Apple's Objective C development framework. The second path is Carbon, which is Apple's C API for writing Mac applications. The third path is to use a game library like SDL or Allegro.

What path should you choose? The path to choose depends on whether you want your game to run only on Mac OS X or if you want your game to run on multiple operating systems. If you want your game to run on multiple operating systems, you should use a library like SDL or Allegro. Those libraries handle a lot of cross-platform issues for you so one set of code will run on Linux, Mac OS X, and Windows with minimal changes. I'm not familiar with Allegro, but I know SDL works well on multiple platforms. Learning SDL is going to be easier than learning multiple technologies to get cross-platform compatibility.

If you want to write a Mac-only game, you have a choice between the Cocoa and Carbon paths. One thing to keep in mind is that Cocoa and Carbon are frameworks for writing GUI applications. You're going to use a tiny portion of them in your games: creating a window to draw into, and running an event loop. Most of Apple's game technologies have a C interface, which means they consist of C functions. Both Cocoa and Carbon applications can call C functions so you can write games in both Cocoa and Carbon. Should you use Cocoa or Carbon?

Both Cocoa and Carbon are suitable for games so the path you choose depends on personal preference. One thing to consider is that on Mac OS X 10.5, you can write 64-bit Cocoa applications, but most of Carbon will not have 64-bit support. I can't imagine you needing to write a 64-bit game right now, but Cocoa's 64-bit support is an advantage. I would strongly recommend Cocoa for creating tools for your game, such as level editors. Cocoa is easier than Carbon for GUI programming.

One disadvantage of Cocoa for some people is it uses Objective C, which looks strange to C and C++ programmers. But you can mix Objective C and C++ code using Objective C++ so you can write most of your game in C++, writing only a small amount of code in Objective C.

Technologies to Learn

The good thing about game programming is that many aspects are operating-system independent, such as artificial intelligence, physics, and game logic. Information about these topics will be usable on a Mac. But there are some aspects of game programming that differ between operating systems. The rest of the article discusses the technologies to use for these aspects on a Mac.

Graphics and Windowing

Use OpenGL for your game's graphics. It's the way to take advantage of hardware acceleration for 2D and 3D applications on Mac OS X. It also runs on Linux and Windows, which means there is a lot of information on OpenGL available on the Internet. Mac OS X ships with the OpenGL framework so all you have to do to use OpenGL is link to the OpenGL framework. In Xcode you would add the OpenGL framework to your project.

What is operating system dependent is creating a window for OpenGL to draw in. Apple has Cocoa OpenGL for Cocoa programs, AGL for Carbon frameworks, and GLUT, which is a cross-platform window and event library. GLUT is good for learning OpenGL, but it's not suitable for a shipping game. SDL has support for creating OpenGL windows.

Reading Player Input

In Carbon and Cocoa you use event handling to read keyboard and mouse input. The HID Manager provides support for joysticks and gamepads. It's not well-documented, but Apple provides the HID Utilities source code that you can use in your games. SDL comes with support for keyboards, mice, and joysticks.

Sound

The technologies you need to learn for game audio depends on your game's needs. For positional and 3D audio, use OpenAL, which is a cross-platform 3D audio library. If you have simpler audio needs where you just want to play background music and sound effects, use QuickTime with Cocoa and Carbon. SDL games should use SDL_mixer, which is a library for playing audio in SDL games.

Loading Textures

OpenGL games make heavy use of texture maps, and loading the textures from disk is not part of OpenGL. Using an image loading library simplifies the loading of the texture files. Cocoa and Carbon programs can use QuickTime to load textures. SDL games can use SDL_image, which is a library to load image files in SDL games.

Networking

Mac games normally use Unix sockets for networking. You can use sockets directly, or you can use one of the many networking libraries that use sockets. Apple provides the CFNetwork class that sits on top of sockets. SDL programs can use SDL_net, which is a wrapper around sockets.

Apple also has an open-source network library called OpenPlay. OpenPlay is a higher-level library than CFNetwork and SDL_net. It runs on Linux, Mac OS X, and Windows. But it hasn't been updated much in the past few years so I don't know how well it runs on Intel Macs.

Conclusion

For more information on Mac game development, go to Apple's game development page. It has links to the OpenGL and OpenAL pages, as well as a link to iDevGames, a Mac game development site. iDevGames is a good place to go with your Mac game programming questions.

The Mac game development page also has articles, guides, and sample code. I recommend reading the Getting Started article as well as the OpenGL Programming Guide for Mac OS X.