SDL and OpenGL

Author: Mark Szymczyk
Last Update: December 29, 2009

This article provides an introduction to the Simple DirectMedia Library (SDL) and using it for OpenGL programs. Although I focus on using SDL with Mac OS X in the article, there is also a lot of good information for SDL developers on any platform.

Updates

I removed the last three sentences of the first paragraph in the Setting Up SDL in Xcode section. SDL 1.2.14 removed the Project Builder templates so there was no need to explain the difference between the Xcode and Project Builder templates. SDL now has Xcode project templates for Mac OS X 10.4 (Tiger), 10.5 (Leopard), and 10.6 (Snow Leopard). (December 29, 2009)

I revised the material on including the OpenGL header files. The new material makes it very clear that including `SDL_opengl.h` is superior from a cross-platform perspective than including `gl.h` manually. (December 4, 2007)

The Mac OS X version of SDL 1.2.10 no longer includes an installer. The article has been updated to reflect that information. (June 9, 2006)

SDL 1.2.10 has been released. The Mac OS X version supports universal binaries. You no longer have to download the SDL source and build the framework to create universal binaries that run on Intel and PowerPC Macs. (May 17, 2006)

Introduction

SDL is an open source library to handle the operating system dependent parts of game development. It can handle things like creating a window, reading mouse, keyboard, and joystick input from the player, playing audio, and creating threads.

SDL shines for writing games on multiple operating system. Suppose you wanted to write an OpenGL game that ran on Linux, Mac OS X, and Windows. Without a library like SDL, you would have to learn the following technologies:

- WGL, GLX, and either AGL or Cocoa OpenGL to create OpenGL windows.
- DirectX for Windows audio and reading player input.
- QuickTime or Core Audio for Mac OS X audio.
- Either Carbon or Cocoa event handling to read the keyboard and mouse on Mac OS X.

SDL saves you the hassle of having to become intimately familiar with the details of programming multiple operating systems. Learn SDL and your code can run on multiple operating systems in the time it takes to write a game for one operating system.

SDL comes with functions for 2D graphics, but I'm going to be focusing on using SDL with OpenGL in this article. 2D drawing with OpenGL is faster than using SDL's functions because OpenGL takes advantage of graphics hardware acceleration. Plus, OpenGL can be used for both 2D and 3D graphics, which makes OpenGL more versatile than SDL's 2D drawing functions.

Setting Up SDL in Xcode

Xcode doesn't ship with SDL support so you must download the SDL libraries from the SDL website. At the download site you will see two types of libraries: runtime and development. You must download both libraries. The runtime library contains the SDL framework. The development libraries install documentation and SDL Xcode project templates on your computer. Copy the SDL.framework folder that is part of the runtime library download to `/Library/Frameworks`. The development library download contains instructions on how to install the documentation and project templates.

After installing SDL, launch Xcode and choose File > New Project to create a new project. In the Applications collection of project templates, you will see a SDL OpenGL Application project. Select that project and click the Next button. Give your project a name, choose a location to store the project, and click the Finish button.When you create a SDL OpenGL application project, the detail view looks like Figure 1. The project template provides files to create an application without having to write any code, but these files get in the way when writing your own programs. You can remove the GLUT framework from the project as well as the files `atlantis.c`, `atlantis.h`, `dolphin.c`, `shark.c`, `swim.c`, and `whale.c`. If you want to write your program in C++, you can also remove the `main.c` file from the project and add a file named `main.cpp` to the project.

The files `SDLMain.h` and `SDLMain.m` contain glue code that allow SDL applications to run on Mac OS X. Make sure you keep them in the project. If you decide to use a nib file for your application's menu bar, you must change the value of the following line of code in `SDLMain.m`:

```
#define             SDL_USE_NIB_FILE   0
```

To the following:

```
#define             SDL_USE_NIB_FILE   1
```

In your source code files you must include the header `SDL.h`.

```
#include "SDL.h"
```

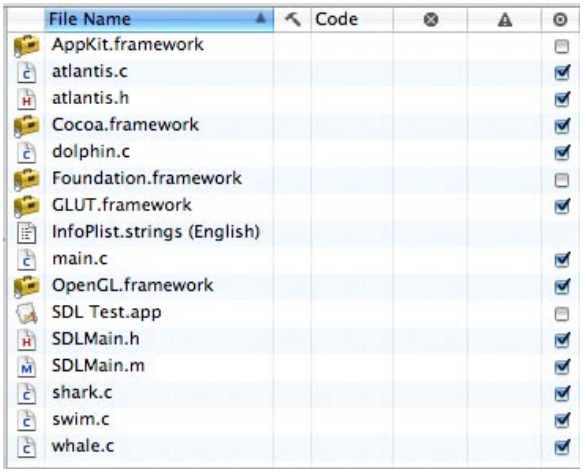

Figure 1

The files that are included when you create a SDL OpenGL Application Xcode project.

To use OpenGL with SDL, include the SDL header file `SDL_opengl.h`. `SDL_opengl.h` includes the OpenGL header files `gl.h` and `glu.h` for you. Including `SDL_opengl.h` is much easier for cross-platform games than manually including `gl.h` and `glu.h` because Mac OS X, Linux, and Windows have different syntax for including OpenGL header files.

```
#include "SDL_opengl.h"
```

Initializing SDL

When your application starts, you must initialize SDL in order to call any SDL functions in your program. Call the function `SDL_Init()` to initialize SDL. This function takes one argument: flags that tell SDL the subsystems to initialize. There are five subsystems: timer, audio, video, CD_ROM, and joystick. Use the flag `SDL_INIT_EVERYTHING` to initialize all five subsystems.

```
int error;
error = SDL_Init(SDL_INIT_EVERYTHING);
```

Setting the OpenGL Attributes

To use OpenGL in a SDL program, you must create a draw context. A draw context is where your program draws graphics. If you don't create a draw context, your program will crash when it makes an OpenGL function call.

Before creating an OpenGL draw context with SDL, you must specify the attributes you want the context to have. Call the function `SDL_GL_SetAttribute()` to set an attribute. This function takes two arguments. The first argument is the name of the attribute, and the second argument is the desired value for the attribute.

The OpenGL attributes you can set generally involve the size of OpenGL's frame, depth, stencil, and accumulation buffers. One attribute virtually every game needs is double buffering. When you use double buffering, drawing is done in an offscreen buffer and the image is moved from the offscreen buffer to the screen. Use the following call to use double buffering in OpenGL:

```
SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
```

If you're writing a 3D game, you must set the size of the depth buffer. The following code requests a 16-bit depth buffer:

```
SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 16);
```

On Mac OS X you don't need to set the size of the frame buffer's red, green, blue, and alpha components. If you create a draw context with 32 bit color, Mac OS X uses 8 bits for each of the four components. On other operating systems, you may need to set the size of the frame buffer's components. The following code sets 8 bits of red, green, blue, and alpha:

```
SDL_GL_SetAttribute(SDL_GL_RED_SIZE, 8);
SDL_GL_SetAttribute(SDL_GL_GREEN_SIZE, 8);
SDL_GL_SetAttribute(SDL_GL_BLUE_SIZE, 8);
SDL_GL_SetAttribute(SDL_GL_ALPHA_SIZE, 8);
```

Creating the OpenGL Draw Context

To create the OpenGL draw context, call the function `SDL_SetVideoMode()`. This function takes four arguments: width, height, color depth, and flags. If you specify a color depth of 0, SDL uses the current color depth.

For an OpenGL context you must use the flag `SDL_OPENGL`. If you want a fullscreen draw context, you must also include the flag `SDL_FULLSCREEN`. The following code creates a 1024 by 768 pixel fullscreen OpenGL draw context that uses the current color depth:

```
SDL_Surface* drawContext;
Uint32 flags;

flags = SDL_OPENGL | SDL_FULLSCREEN;
drawContext = SDL_SetVideoMode(1024, 768, 0, flags);
```

If you read the SDL documentation on `SDL_SetVideoMode()`, you will see that one of the flags you can use is `SDL_DOUBLEBUF`. This flag is for SDL programs that do not use OpenGL. If you're using OpenGL for your graphics, do not use the `SDL_DOUBLEBUF` flag. Enable OpenGL double buffering by calling `SDL_GL_SetAttribute()`.

Now that you've created the draw context, you can make any OpenGL function call. When you're finished drawing a frame, call the function `SDL_GL_SwapBuffers()` to display the newly drawn frame on the screen.

```
SDL_GL_SwapBuffers();
```

Timers

Timers are not something every game uses so you might be wondering why I'm covering them in this article. Trying to get SDL timers to work caused me a lot of aggravation and I spent hours trying to figure out what the problem was. I'm writing about SDL timers to save you the aggravation I went through.

A timer is a function the operating system calls periodically. You get to specify how often the timer fires, which is how often the operating system calls the timer. Use a timer when you want to do something repeatedly in your program at regular time intervals.

Writing a Timer

A timer function takes the form.

```
Uint32 TimerFunctionName(Uint32 interval, void* param)
```

The `interval` argument specifies the timer's firing rate in milliseconds. The `param` argument contains data you can supply to the timer when installing the timer.

C++ programs that want timer functions to be member functions of a class must declare the timer to be a static function in the header file.

```
static Uint32 TimerFunctionName(Uint32 interval, void* param);
```

If you read my article on Carbon event timers, you know that timers generally call functions in your program. Unfortunately you cannot directly call a function from a SDL timer. To call a function from a timer, you must create a SDL user event and add the event to the event queue. Your event handler calls the function when the user event occurs.

A SDL user event has four fields you must set.

- Type, which will be `SDL_USEREVENT` for a user event.
- Code, which is an integer value that uniquely defines the event type.
- Data1 and data 2, which let you attach data to the event. If you have no data to attach, set the fields to 0.

Call the function `SDL_PushEvent()` to add the event you created to the event queue. The following timer function creates an event to run a game loop:

```
const int RUN_GAME_LOOP = 1;

Uint32 GameApp::GameLoopTimer(Uint32 interval, void* param)
{
    // Create a user event to call the game loop.
    SDL_Event event;

    event.type = SDL_USEREVENT;
    event.user.code = RUN_GAME_LOOP;
    event.user.data1 = 0;
    event.user.data2 = 0;

    SDL_PushEvent(&event);

    return interval;
}
```

Notice how the timer returns the value of the `interval` argument. Doing this tells the timer to fire at the same rate next time. If you want the timer to stop firing, return 0.

Installing a Timer

Call the function `SDL_AddTimer()` to install the timer. This function takes three arguments. The first argument is the firing rate in milliseconds. The second argument is the name of the timer function. The third argument is any data you want to supply to the timer. The following function code installs the timer I wrote in the previous section and tells it to fire 50 times per second:

```
SDL_TimerID timer;
timer = SDL_AddTimer(20, GameLoopTimer, this);
```

SDL Event Loop

If your game doesn't read input from the player, you don't have much of a game. SDL supports keyboard, mouse, and joystick input as well as changing applications, quitting the application, resizing windows, and updating the contents of windows. You can also create your own events, which you may remember if you read the previous section of this article.

Mac OS X developers should keep in mind that closing a window generates a quit event. The Cmd-Q key combination, which is the standard way to quit Mac OS X applications, does not generate a quit event. If you want your game to support the Cmd-Q key combination on Mac OS X, you must handle Cmd-Q in your keyboard event handler.

What SDL programs do is check for events. When an event occurs, SDL provides you with the event's data structure. Use the event structure's type field to determine the type of event that occurred. Normally you write a `switch` statement with one case for each type of event your game handles.

There are two ways SDL can check for events: polling and waiting. Polling tells the operating system to constantly check for events. Waiting tells the operating system to do nothing until an event occurs. Waiting is more efficient than polling.

Call the function `SDL_WaitEvent()` to wait for events, and call the function `SDL_PollEvent()` to poll. The following code is an example of an event loop that waits for events:

```
bool done;

void GameApp::EventLoop(void)
{
    SDL_Event event;

    while((!done) && (SDL_WaitEvent(&event))) {
        switch(event.type) {
            case SDL_USEREVENT:
                HandleUserEvents(&event);
                break;

            case SDL_KEYDOWN:
                // Handle any key presses here.
                break;

            case SDL_MOUSEBUTTONDOWN:
                // Handle mouse clicks here.
                break;

            case SDL_QUIT:
                done = true;
                break;

            default:
                break;
        }   // End switch

    }   // End while

}

void GameApp::HandleUserEvents(SDL_Event* event)
{
    switch (event->user.code) {
        case RUN_GAME_LOOP:
            GameLoop();
            break;

        default:
            break;
    }
}
```

If you decide to wait for events, make sure you provide a way to exit the `while` loop. When you call `SDL_WaitEvent()`, nothing happens until an event occurs. If you provide no way out of the `while` loop, you won't be able to quit the program.

Cleaning Up

When the player quits your game, you must remove any timers you installed and shut down SDL. Call the function `SDL_RemoveTimer()` to remove a timer. Call the function `SDL_Quit()` to shut down SDL.

```
SDL_bool success;
SDL_TimerID timer;

success = SDL_RemoveTimer(timer);
SDL_Quit();
```

Conclusion

Included with this article is source code for a simple SDL application. It provides a nice starting point for people new to SDL and OpenGL development.

The SDL library is too large to completely cover in a single article. Mac OS X developers who download the SDL development libraries can read the SDL documentation at `/Developer/Documentation/SDL`. Documentation is also available at the SDL website.