Next

Lesson 01 – Setup & Windowing

Home

Introduction

This section deals with using the SDL2 (Simple DirectMedia 2) library. It's a (relatively) easy to use library that adds multimedia functionality without relying on operating-specific features.

What SDL provides:

- Window management
- Software (CPU) and hardware (GPU) rendered 2D graphics
- Input event system
- Timing management
- Audio Processing
- File IO and library loading
- Threading
- An OpenGL API for 3D graphics

There are extensions to SDL that provide many more features—networking, a better audio system, image loading, etc.

Example Program

Download

```
#include <iostream>
#include <SDL.h>

// You shouldn't really use this statement, but it's fine for small programs
using namespace std;

// You must include the command line parameters for your main function to be recognized by SDL
int main(int argc, char** args) {

	// Pointers to our window and surface
	SDL_Surface* winSurface = NULL;
	SDL_Window* window = NULL;

	// Initialize SDL. SDL_Init will return -1 if it fails.
	if ( SDL_Init( SDL_INIT_EVERYTHING ) < 0 ) {
		cout << "Error initializing
```

Throughout this section, I will be referencing many functions provided by SDL. The point of these lessons is to teach you how to use the functionality of SDL, not the minutia of parameters, etc. Hence, each reference to an SDL object will include a link to its SDL documentation page. If you have questions about a function's parameters, return type, side effects, or the members of a structure, etc, simply read the documentation. It will be a very valuable resource.

Documentation tips:

- Check the documentation if you're wondering about anything.
- The example code can be very useful.
- Always read the remarks—they may reveal unexpected features and describe when memory should be saved/freed.
- The "related functions" section at the bottom of each page will show you what else to use. We won't be covering every function in every lesson.

Setup

Guides:

Setup for Dev C++
Setup for Visual Studio

```
SDL: " << SDL_GetError() << endl;
		system("pause");
		// End the program
		return 1;
	}

	// Create our window
	window = SDL_CreateWindow(
"Example",
SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED,
1280, 720, SDL_WINDOW_SHOWN
);

	// Make sure creating the
window succeeded
	if ( !window ) {
		cout << "Error creating
window: " << SDL_GetError()  <<
endl;
		system("pause");
		// End the program
		return 1;
	}

	// Get the surface from the
window
	winSurface =
SDL_GetWindowSurface( window );

	// Make sure getting the surface
succeeded
	if ( !winSurface ) {
		cout << "Error getting
surface: " << SDL_GetError() <<
endl;
		system("pause");
		// End the program
		return 1;
	}

	// Fill the window with a white
rectangle
	SDL_FillRect( winSurface,
NULL, SDL_MapRGB( winSurface-
>format, 255, 255, 255 ) );

	// Update the window display
	SDL_UpdateWindowSurface(
window );
```

Other Setup Guides

When using SDL functions or objects, you must (of course) include their header files. The file SDL.h will automatically include just about everything else—most often this is all you will need. However, many functions are prototyped in specific files. Simply check the documentation if you're not sure what you need to include.

```
        // Wait
        system("pause");

        // Destroy the window. This will
also destroy the surface
        SDL_DestroyWindow( window );

        // Quit SDL
        SDL_Quit();

        // End the program
        return 0;
}
```

Also, if you set up your compiler correctly, you should be able to use pointy brackets (e.g. #include <SDL.h>). This tells the compiler to look in its specified include directories.

Initializing SDL

Before doing anything else, you must initialize SDL as a whole. As you might expect, SDL_Init() does this. If you would like to initialize all parts of SDL, pass SDL_INIT_EVERYTHING.

SDL also allows you to initialize particular subsets (or subsystems) of the library individually. To specify, either specify flags to SDL_Init(), or use SDL_InitSubSystem(). If you plan to initialize the subsystems individually, simply pass 0 to SDL_Init().

```
SDL_Init( SDL_INIT_EVERYTHING );
```

Creating a Window

You must create the window that your program will use for multimedia input and output. When creating a Windows application from scratch, you must define a "WinMain," call the operating system to get handles, create windows, etc, etc. SDL provides a much simpler, platform-independent windowing API.

For managing a window, SDL conveniently provides the structure SDL_Window and functions such as SDL_CreateWindow().

Slight tangent: you might notice that there is no documentation link to

Slight tangent: you might notice that there is no documentation link to SDL_Window. This is because the structure is opaque; your program cannot see what is actually contained in a "SDL_Window." You will simply manage a pointer to a SDL_Window.

SDL_CreateWindow() does what you'd expect: it takes parameters specifying the name, size, position, and options for the window, and returns a pointer to the new SDL_Window structure. Look at the SDL documentation for details.

```
SDL_Window* win = SDL_CreateWindow( "my window", 100, 100,
640, 480, SDL_WINDOW_SHOWN );
```

Most SDL functions will return a specific value on failure. For functions that return pointers, this value is NULL. Hence, you can easily check if the operation succeeded.

After any error, the function SDL_GetError() allows you to retrieve a string describing the error.

```
if ( !win ) {
	cout << "Failed to create a window! Error: " << SDL_GetError() <<
endl;
}
```

Surfaces

Once you have created a window, you need a way to draw to it. SDL abstracts any area you can draw to—including loaded images—as a "surface." (This is for software rendering—in the future, we'll get into GPU rendering, which does not use surfaces.)

The structure SDL_Surface and functions such as SDL_LoadBMP() and SDL_GetWindowSurface() provide the software rendering (also known as blitting) API.

As you'd expect, use SDL_GetWindowSurface() to get your window's surface. After drawing onto this surface, the results can be seen in the window by calling SDL_UpdateWindowSurface().

```
SDL_Surface* winSurface = SDL_GetWindowSurface( win );

// do drawing

SDL_UpdateWindowSurface( win );
```

Drawing a Rectangle

To test if your window surface is working correctly, you can fill it with a color. A very easy way to do this is with SDL_FillRect(). To fill the entire window, simply pass NULL instead of a SDL_Rect pointer. Additionally, SDL_FillRect() takes a specifically formated number representing a color. To get a color in this format, you can call SDL_MapRGB with the surface format and desired RGB values.

```
SDL_FillRect( winSurface, NULL, SDL_MapRGB( winSurface->format,
255, 90, 120 ));
```

Shutting Down

Once your program has completed its operation, it must destroy the window and free related resources. As you might expect, SDL_DestroyWindow() does just that. The function will close your window, freeing related memory (including the window surface).

```
SDL_DestroyWindow( win );
win = NULL;
winSurface = NULL;
```

Finally, to shut down SDL as a whole, call SDL_Quit(). It's pretty self-explanatory.

```
SDL_Quit();
```
