**Introduction**

Welcome, to what is now your first [OpenGL](http://www.swiftless.com/opengltuts.html) tutorial. Today, we are going to learn how to create a Window using [OpenGL](http://www.swiftless.com/opengltuts.html), GLUT (The OpenGL Utility) and GLEW.

But first, let me give you a brief rundown of GLUT and OpenGL, this is after all, your first time working with them both, and you must have some idea at least about OpenGL if you want to learn it.

**OpenGL**

OpenGL, also known as the Open Graphics Library, is a library designed for cross platform 3D graphics. It is directly, the only competitor to Direct3D in the DirectX library. OpenGL is also hardware accelerated, just like Direct3D, and both of these libraries are the main focus for graphics card performance.

**GLUT**

So what about GLUT? GLUT (The OpenGL Utility), is a tool which allows the creation of windows and handling of input on multiple systems. Making OpenGL cross-platform and extremely simple to set up. Unfortunately, the original GLUT is no longer being developed, but don’t worry, there is a remake of GLUT called FreeGLUT, which even works with the original GLUT dll files and is exactly the same to use.

**GLEW**

And finally GLEW, GLEW is the OpenGL Extension Wrangler, and gives us easy to use calls to OpenGL extensions, without us having to declare the dll entry points, which while are documented and are on the internet, is still a pain, and a waste of time, when you have a library designed for this.

**Installation**

Now that you know a little bit about them both, why don’t we actually get to using them? That is after all, why you are here.

**Downloads**

First off you are going to need to download:

FreeGLUT – [http://freeglut.sourceforge.net/](http://freeglut.sourceforge.net)
GLEW – <http://glew.sourceforge.net/>

Once you have them both, follow their installation instructions, which are fairly simple. In short, find the “lib” and “include” folders for your current Visual Studio installation (for VC, not for VC#), and put the required files in each folder. Then find your System32 or SysWOW64 folder, and put the required dll files in them.

**Visual Studio Project Configuration**

Then you are going to want to open Visual Studio (all these tutorials have been written in the latest Visual Studio 2010, and they will work in everything prior) and start a New Project. From the project selection list, find and click on “Win32 Console Application”. You should then be prompted with a setup wizard, click on “Next” and tick “Empty Project”, and press “Finish”.

You should now have an empty Win32 console application, which we need to link to some library files. Go to your project properties, and under: “Configuration Properties” -> “Linker”, click on “Input” and add two additional dependencies. Add “glew32.lib” and “freeglut.lib” on separate lines.

Visual Studio should now be configured for us to start learning OpenGL.

**Coding**

And finally, let’s get on to some coding.

First, start a new C++ file in your project, you can call it whatever you like, but I always like to call my first file “main.cpp”, because I’m just so original 

Inside your cpp file, you will want to first of all, include the header files for GLUT/FreeGLUT and OpenGL. Which after you have entered them, will look something like:

[view plaincopy to clipboardprint?](http://www.swiftless.com/tutorials/opengl/window.html)

1. #include  // Include the GLEW header file
2. #include  // Include the GLUT header file

#include // Include the GLEW header file

#include // Include the GLUT header file

And then create your standard main method, something like:

[view plaincopy to clipboardprint?](http://www.swiftless.com/tutorials/opengl/window.html)

1. int main (void) {
2. }

int main (void) {

}

If this compiles, then you have your header files setup correctly. Congratulations!

**GLUT**

Now it’s time to actually start using GLUT and get a window appearing. The first thing we will want to do is redefine our main method, this is because all of our GLUT calls will be done in this method, and these parameters are required by GLUT. Change the declaration of your main method to:

1. int main (int argc, char \*\*argv) {
2. }

 The parameters argc and argv allow us to add command line arguments, and are required when we initialize GLUT.

It is possible to pass some values from the command line to your C programs when they are executed. These values are called **command line arguments** and many times they are important for your program specially when you want to control your program from outside instead of hard coding those values inside the code.

The command line arguments are handled using main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program. Following is a simple example which checks if there is any argument supplied from the command line and take action accordingly:

It should be noted that **argv[0]** holds the name of the program itself and **argv[1]** is a pointer to the first command line argument supplied, and \*argv[n] is the last argument. If no arguments are supplied, argc will be one, otherwise and if you pass one argument then **argc** is set at 2.

You pass all the command line arguments separated by a space, but if argument itself has a space then you can pass such arguments by putting them inside double quotes "" or single quotes ''. Let us re-write above example once again where we will print program name and we also pass a command line argument by putting inside double quotes:

So now we initialize GLUT, which is done like so:

1. glutInit(&argc, argv); // Initialize GLUT

After we have initialized GLUT, we need to tell it how we want to setup our window. Here we tell GLUT if we want one or two buffers, if we want an alpha channel, if we want depth and stencil buffers, etc. Let’s start off simple, and go with the most basic, GLUT\_SINGLE which will give us a single buffered window.

1. glutInitDisplayMode (GLUT\_SINGLE); // Set up a basic display buffer (only single buffered for now)

Next, we need to set the size and the position on the screen for our GLUT window:

1. glutInitWindowSize (500, 500); // Set the width and height of the window
2. glutInitWindowPosition (100, 100); // Set the position of the window

And finally, we create our window and give it a title/caption:

1. glutCreateWindow (“You’re first OpenGL Window“); // Set the title for the window

 This is going good, we now have a window of the size and position we want, and this should compile and display if you would like to test it (I know I did!). Testing your code often is a great habit to get into. I have known a lot of people to write a lot of code, compile it, and then not know where to start debugging. Even if all you are doing is checking for compilation errors, it’s good to fix these up early.

Once you run this, you will have a window appear with a console window behind it, and it will then disappear as your application would have run out of code and have exited.

So now we have a couple more methods to set up before GLUT opens a window, and then keeps it open.

The first of these things, is we want to tell GLUT what method will store our code for drawing, so that GLUT can then call on it when needed. As per a lot of OpenGL tutorials, I am going to use a method called “display”, as I have picked up the habit. You can call this anything you like, but it has to be declared like such:

1. void display (void) {
2. }

Once we have a method we can use to do our drawing, we need to actually tell GLUT to use this method. To do so, we make a call to glutDisplayFunc() and pass in our display method, this is done inside of our main method after we create our window.

glutDisplayFunc(display);

Now that GLUT knows what method to use for drawing, all we have to do is tell GLUT to enter its Main loop. This is just a giant loop that continues forever, whilst calling the display, idle, [keyboard](http://www.swiftless.com/tutorials/opengl/keyboard.html), mouse and any other call back methods. So in code, it looks theoretically something like:

  while (running) {

 display();

 }

And while this loop is going, so is your application. So to start it, call:

glutMainLoop();

If you choose to run this, your window will stay open, but your window doesn’t have any OpenGL calls in it yet. So to show that OpenGL is working, lets fill in a basic display method.

To do this, we will first set the colour we want the window background to be. We do this with the call to glClearColor, which takes in 4 float parameters, each one referring to a colour, and in the other (red, green, blue, alpha).

So add this line to your display function, and we should get a nice red background:

1. glClearColor(1.f, 0.f, 0.f, 1.f); // Clear the background of our window to red

Now that we have the background colour set, let’s get into the habit of some cleaning up in OpenGL. Every time this display method is called in our loop, it adds on to what is currently stored in OpenGL’s giant state machine. So we need to clean this out, and reset it. We will reset the colour buffer, so that we get our red background every time, and we will load the identity matrix, so all our drawing starts at the same location as the previous time this method was called:

1. glClear (GL\_COLOR\_BUFFER\_BIT); //Clear the colour buffer (more buffers later on)
2. glLoadIdentity();  // Load the Identity Matrix to reset our drawing locations

And finally to top it all off, after we have drawn everything we want, we need a way to tell OpenGL to draw everything we have currently told it. In a single buffered window, we want to just do a straight flush from OpenGL to the window, and we can do this with:

glFlush(); // Flush the OpenGL buffers to the window

And that is it, if everything has worked along the way, you should now have a window that shows up with an OpenGL context, and displays a red background.

Well done!
If you have any questions, please email me at swiftless@gmail.com

1. #include <GL/glew.h> // Include the GLEW header file
2. #include <GL/glut.h> // Include the GLUT header file
3.
4. void display (void) {
5. glClearColor(1.0f, 0.0f, 0.0f, 1.0f); // Clear the background of our window to red
6. glClear(GL\_COLOR\_BUFFER\_BIT); //Clear the colour buffer (more buffers later on)
7. glLoadIdentity(); // Load the Identity Matrix to reset our drawing locations
8.
9. glFlush(); // Flush the OpenGL buffers to the window
10. }
11.
12. int main (int argc, char \*\*argv) {
13. glutInit(&argc, argv); // Initialize GLUT
14. glutInitDisplayMode (GLUT\_SINGLE); // Set up a basic display buffer (only single buffered for now)
15. glutInitWindowSize (500, 500); // Set the width and height of the window
16. glutInitWindowPosition (100, 100); // Set the position of the window
17. glutCreateWindow ("You’re first OpenGL Window"); // Set the title for the window
18.
19. glutDisplayFunc(display); // Tell GLUT to use the method "display" for rendering
20.
21. glutMainLoop(); // Enter GLUT's main loop
22. }