HW0 tutorial

Starter Code for Programming Projects

The purpose of this tutorial is to introduce OpenGL programming and set up your computer so that you can begin developing OpenGL applications. It should take you no more than 20 minutes to complete the tutorial.

In order to display an image, you must create a drawing canvas, which is essentially a window. With OpenGL, you create a window and then create an OpenGL drawing context associated with that window, which allows you to use your video hardware to draw 3D objects directly into that window. Opening that window and creating the OpenGL context is cumbersome and varies from system to system. The purpose of GLFW is to simplify and unify this initial step.

The PCs in the computer lab already have GLFW and Visual Studio installed. It is also possible to run your projects on the lab computers in the AP&M basement, though you may need to go through the given setup instructions before the projects can work.

For your personal computer we will show you how to download and install GLFW and configure Visual Studio/XCode to use it.

The starter code is available on the course website: https://cseweb.ucsd.edu/classes/wi20/cse167-a/

For Gradescope, please submit a zip file of the source code including the shader files.

UPDATE YOUR SYSTEM

We're using modern OpenGL (at least 3.3+, but some systems with OpenGL 3.1 somehow have forward compatibility to 3.3) which came out in 2010, meaning most recent systems should support it by default. However, just in case, make sure your graphics drivers are updated to the latest version.

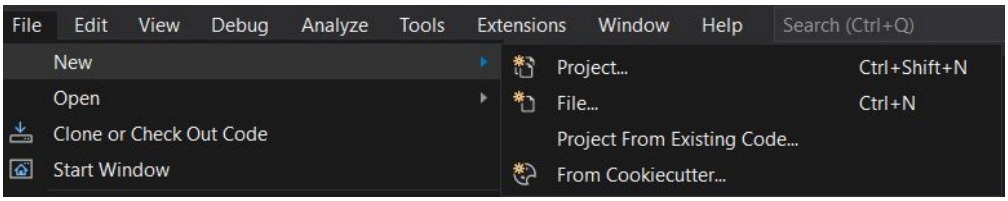
- Windows: Update your graphics driver to the latest version. If your graphics card is too old, replace your graphics card or use a different computer.
- OS X: Update your OS X version to at least 10.9. OpenGL is tied to your operating system version.

On Windows (Skip to the Mac section if you have a mac)

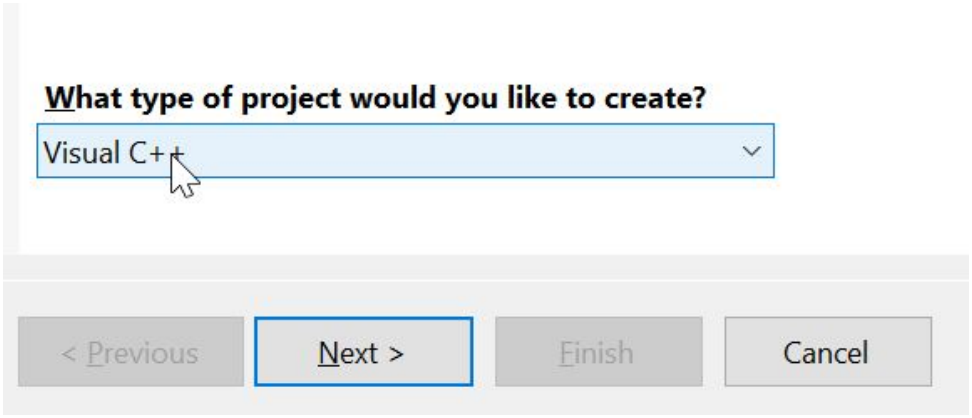
The very first thing you'll be doing is downloading and installing the Microsoft Visual Studio 2019 Community Edition (https://visualstudio.microsoft.com/downloads/) for free.

Once that is done and you have unzipped your starter code, we're ready to begin configuring our Microsoft Visual Studio solution to work with OpenGL.

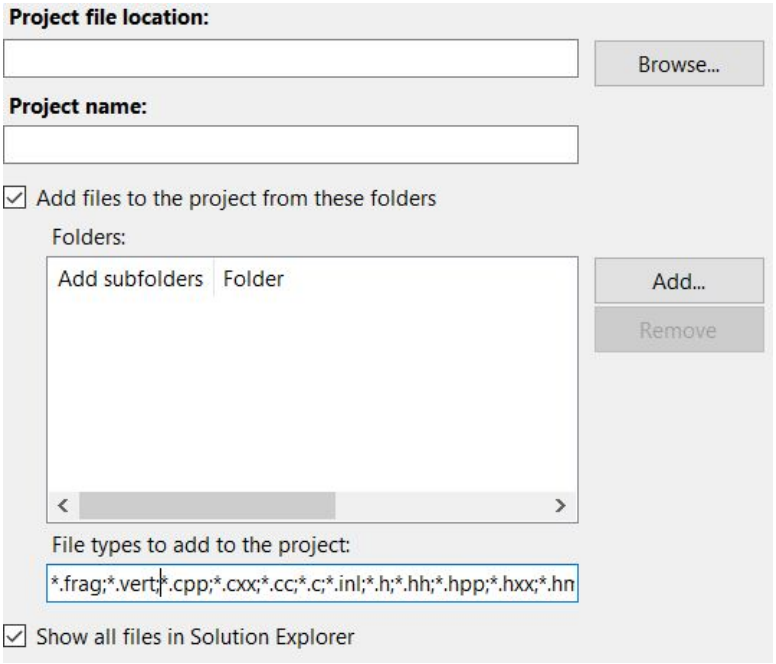
1. Open Microsoft Visual Studio and select “Continue without code”. Go to "File->New->Project From Existing Code...".



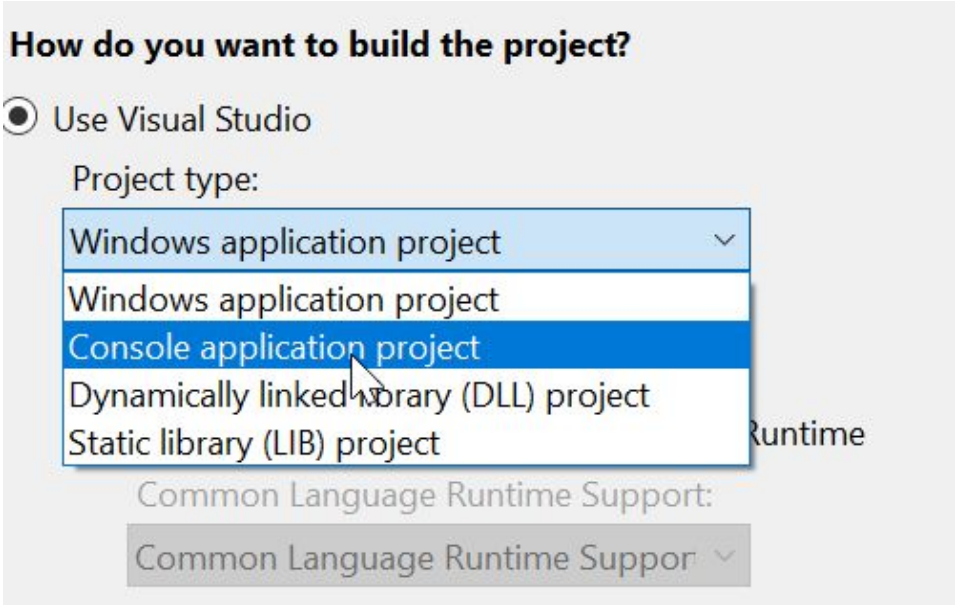
2. Create a **Visual C++** project.



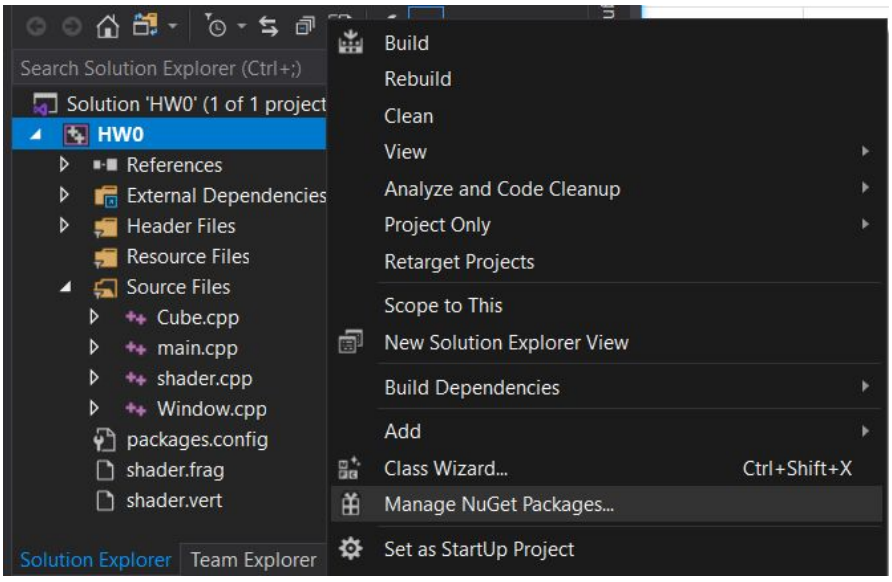
3. On the next page, enter the path to the starter code as the project file location and enter a name for the project (whatever you like). Prepend ***.frag;*** and ***.vert;*** to “File types to add to project” to also see the shader files in the solution explorer.



4. On the next page, create a console application project and click **Finish**.

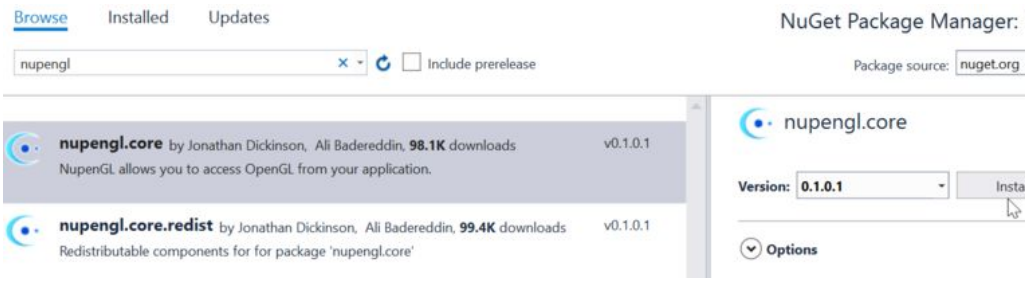


5. Open the **NuGet Package Manager** by right-clicking your project on the left and select **Manage NuGet Packages...**

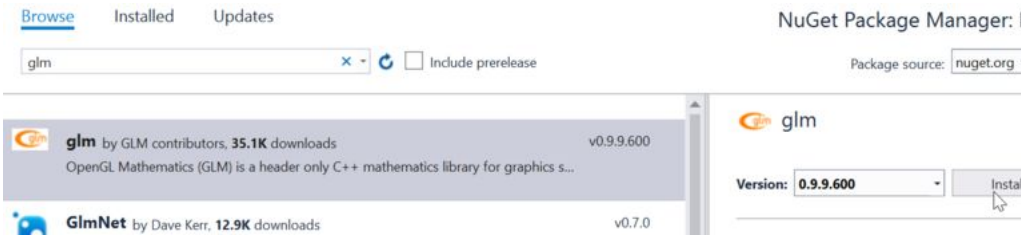


6. Make sure you're in the Browse tab. Search for nupengl and install the **nupengl.core** package (NOT nupengl.core.redist). Alternatively, if you feel that this is too overkill (since nupengl contains 5 or 6 different commonly used libraries in OpenGL applications), download the glew

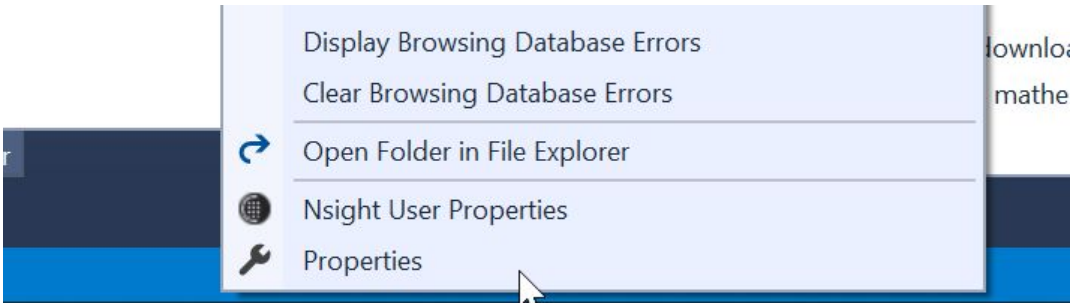
and glfw packages (NOT the .redist packages).



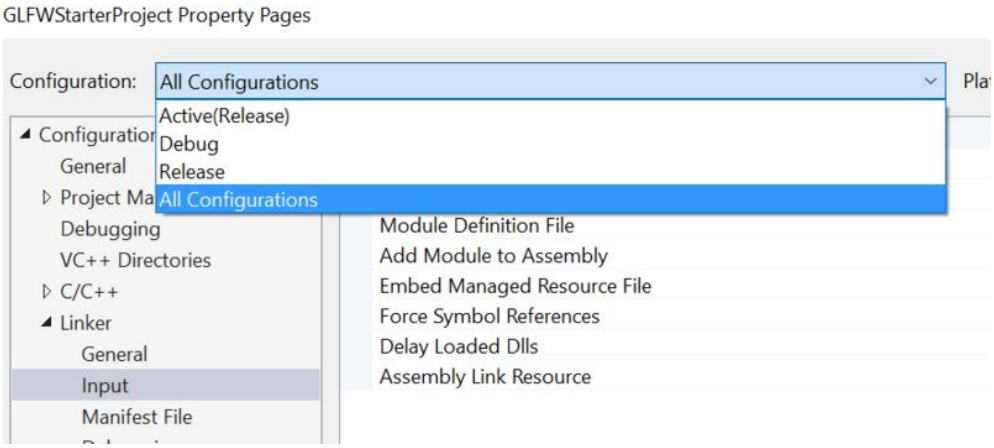
7. Similarly, install the **glm** package.



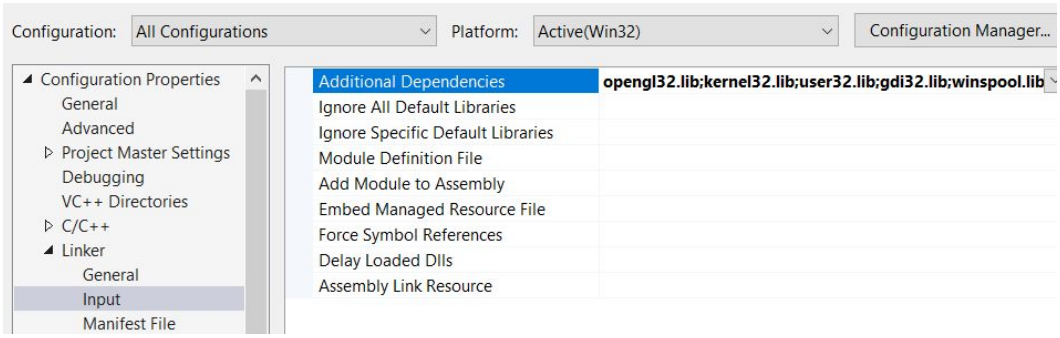
8. Now that we have all our packages, we just need to Link against the OpenGL libraries.
9. We can do that by going to our project's(note: NOT solution) Properties. Right-click your project in the **Solution Explorer** and select **Properties**



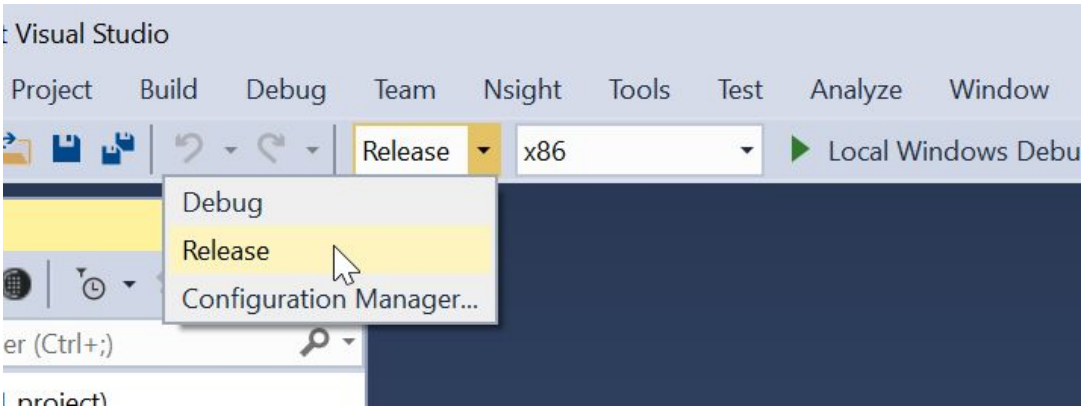
10. Once in project properties, go to Configuration dropdown to change it to All Configurations. This will ensure we edit both the debug/release configuration at the same time, and don't have to repeat the step.



11. Navigate to Linker->Input to prepend **opengl32.lib** to Additional Dependencies.



12. Change the active configuration to **release** for faster execution! If you want to do breakpoint debugging, remember that you'll have to change this back to **debug** to make sure you can see the variables.



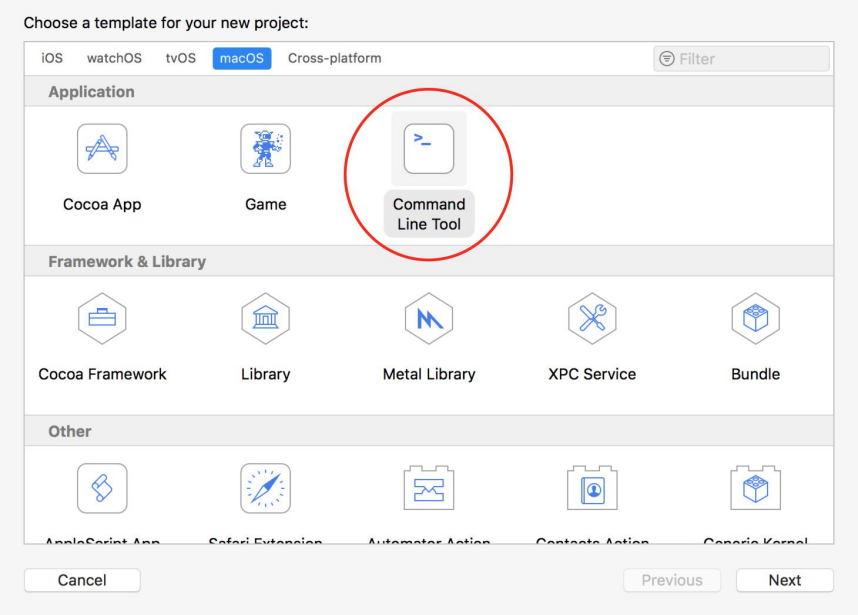
13. Once this is done, build and run the program(CTRL+F5) to see a spinning cube!

On OS X

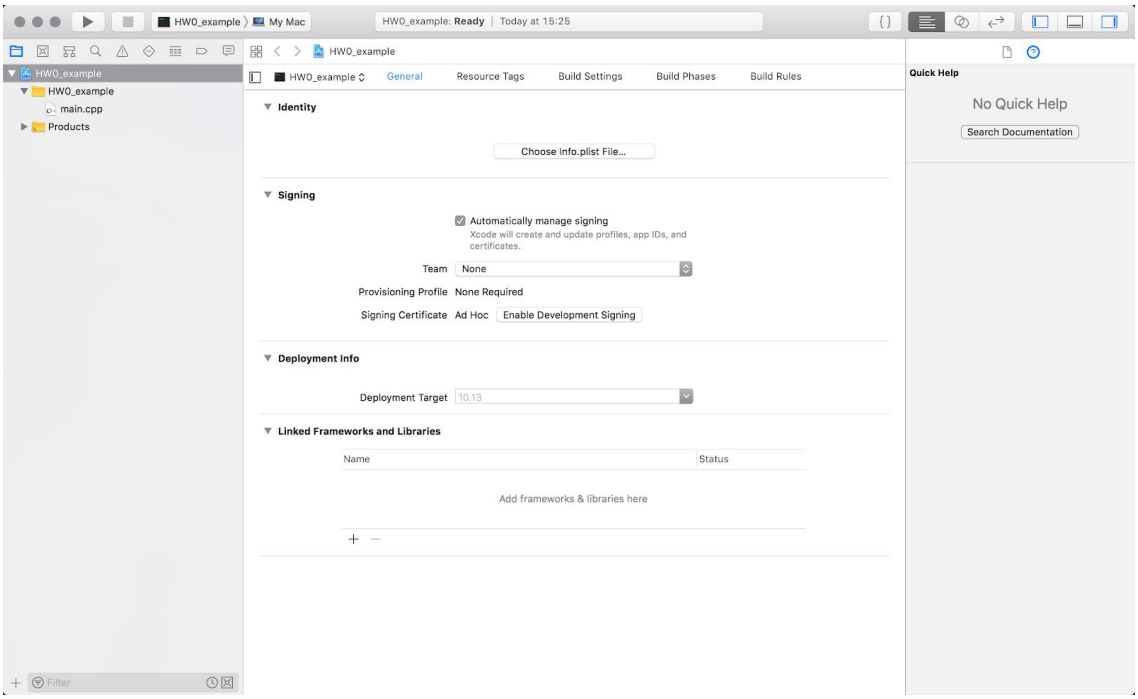
First, let's make sure we have all of the starter code downloaded (linked above).

Once you have unzipped the project, open XCode and create a new project.

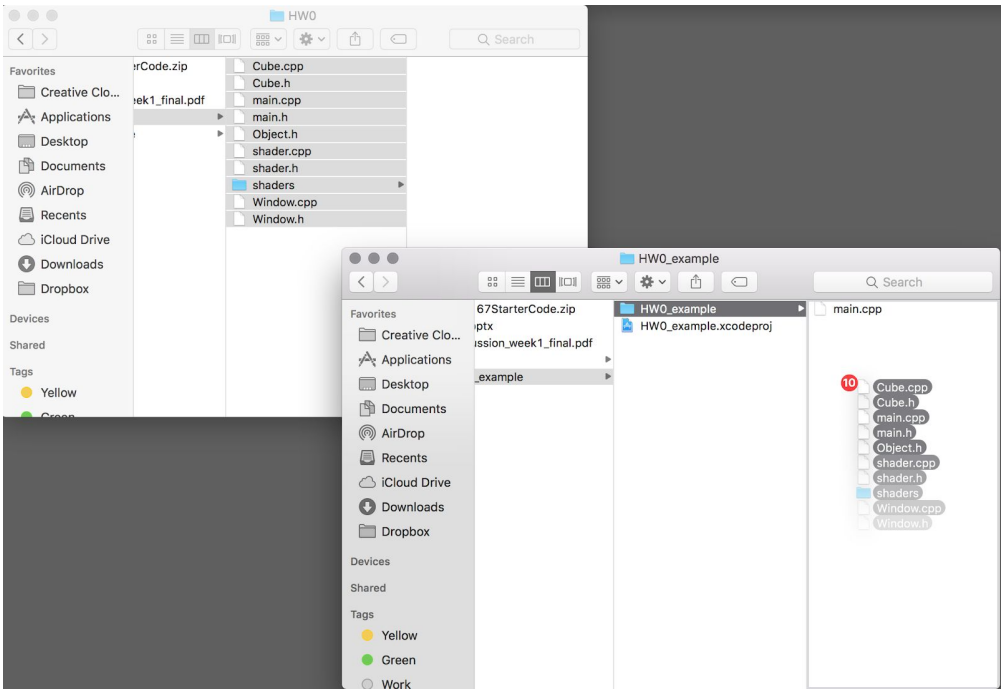
1. Select Command Line Tool as the template for the new project.



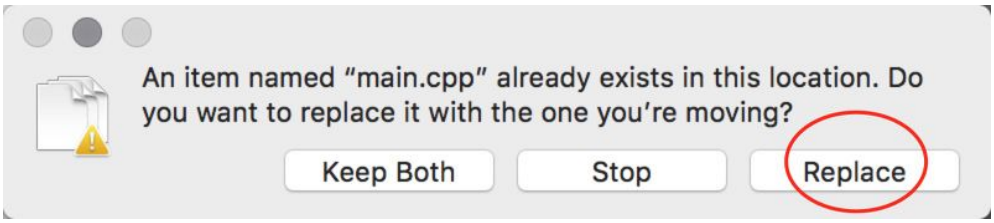
2. Name your project any name of your choosing.
3. Upon completion, you should be presented with something like this:



4. Let's open the directory enclosing our newly created XCode project alongside the unzipped starter code directory. Copy and paste the starter code to the project directory you just created.



5. If presented with the option, replace the already existing files (main.cpp in this case)



All is good for now. We will come back to this later.

Install GLFW and GLM

1. (If you have homebrew already, skip this step) Begin by installing homebrew following this link, or type the following in your command line:

```
$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

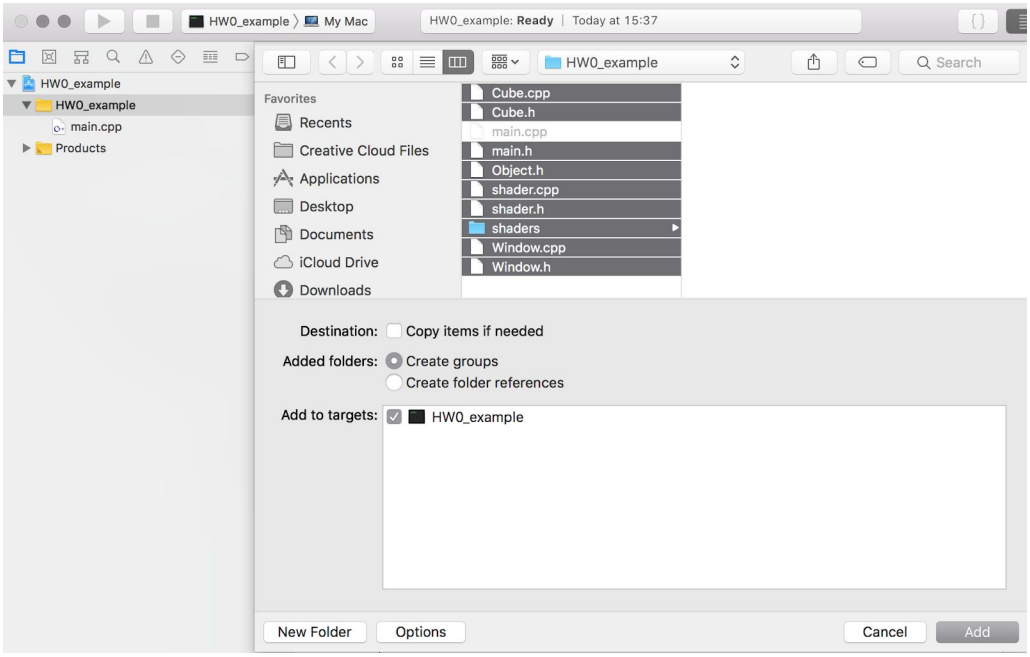
1. Install glfw and glm with homebrew by typing the following in your command line:

```
$ brew install glfw glm
```

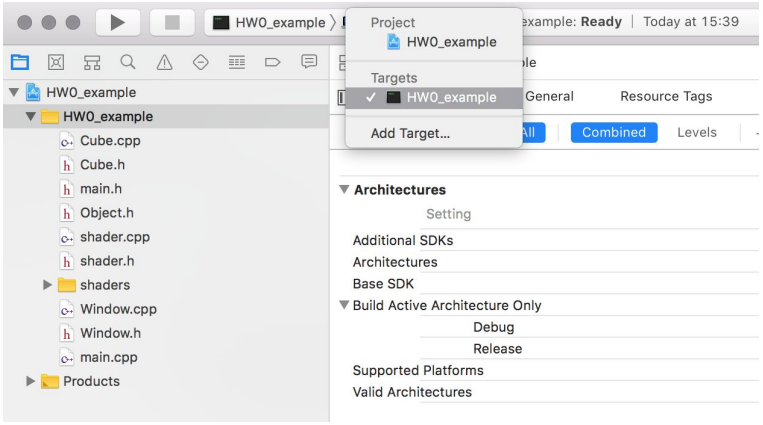
XCode Setup

The last step is to set up XCode to find where we put all of our code, libraries and headers

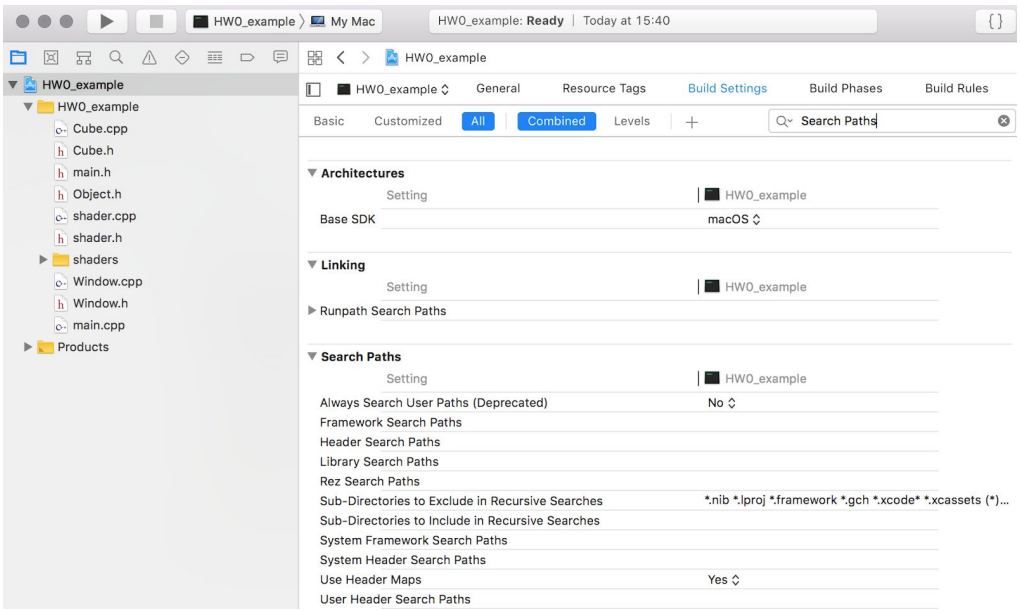
1. Open up the XCode project
2. In the far left panel of the Xcode window, you will see a folder matching the project name.
    1. Right click this folder and click "Add files to <project name>".
    2. Choose the files as shown below and press "Add"



3. Above this folder, you will see the top level which shares the same name as your project.
    - Click this and choose the binary with the same name under TARGETS, then in the "Build Settings" option at the top.



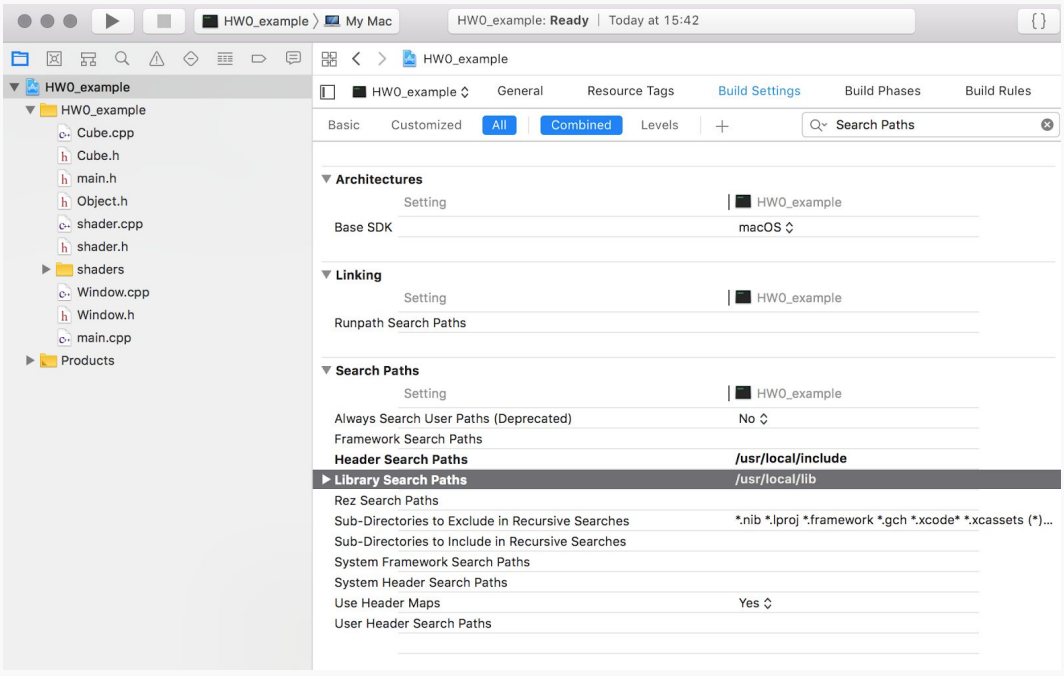
- Type "Search Paths" in the search bar.



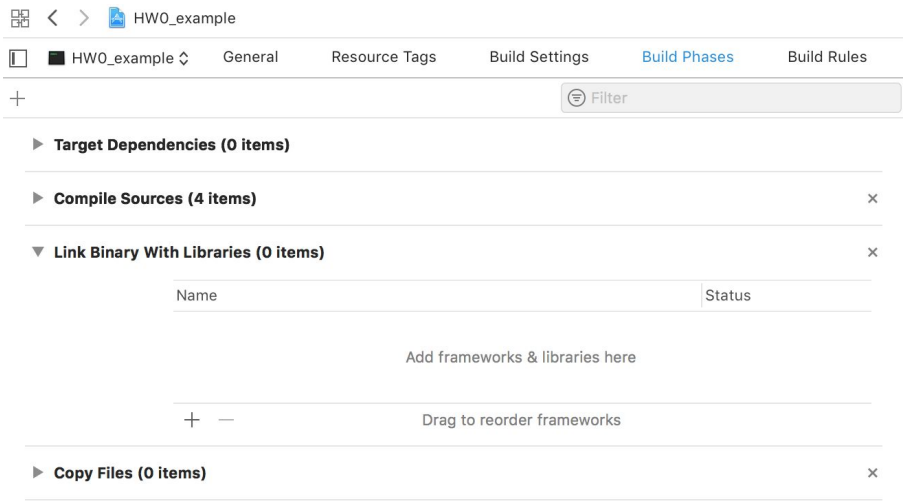
- Add the following paths as shown below so XCode knows where to find all of the glfw and glm headers in our code.

```
/usr/local/include
```

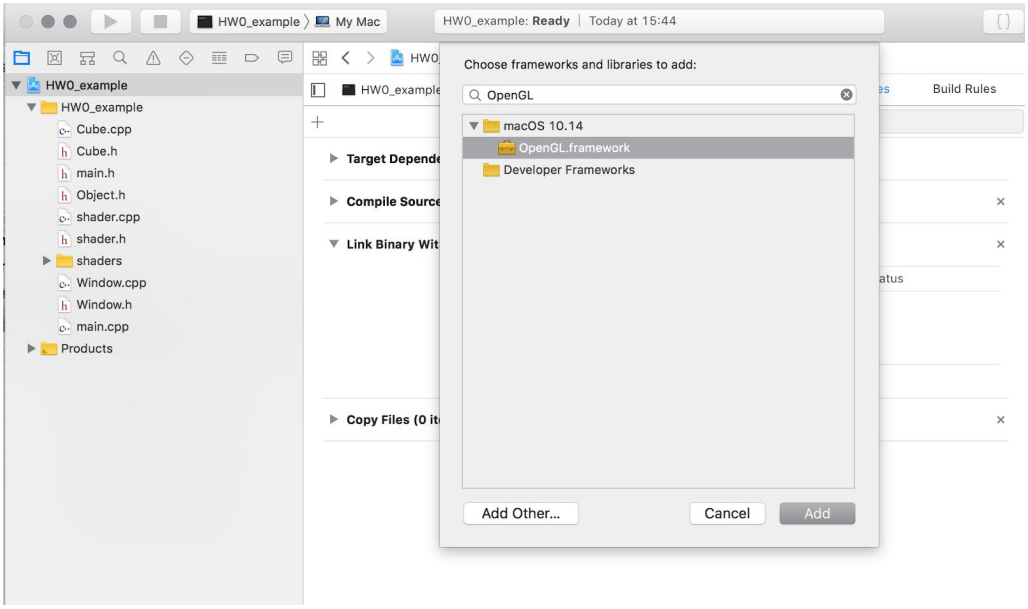
```
/usr/local/lib
```



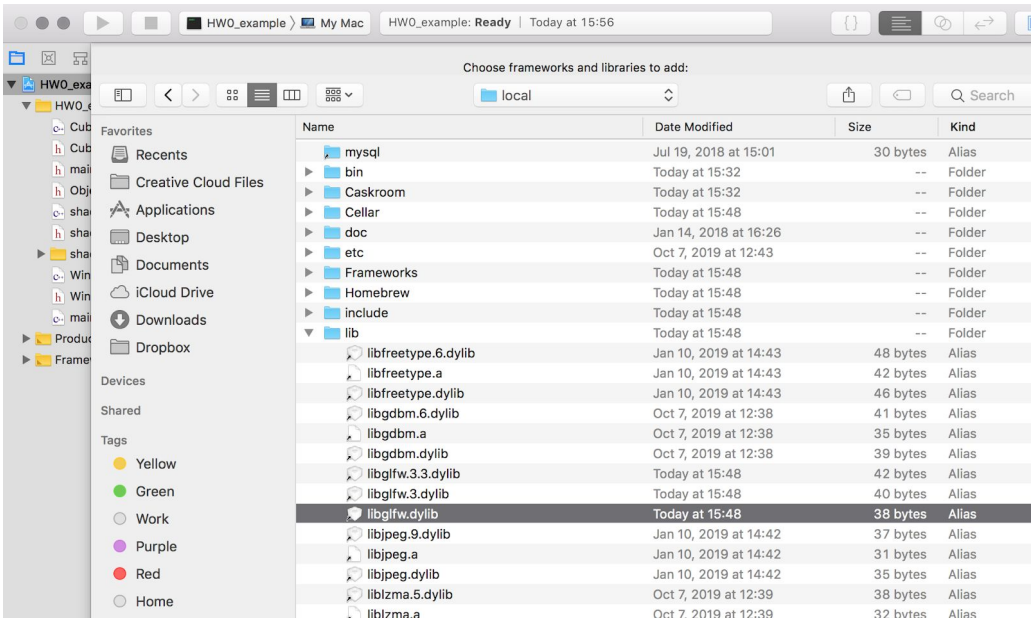
4. At the top of this window, click "Build Phases"
    - Open up the "Link Binary with Libraries" pull down



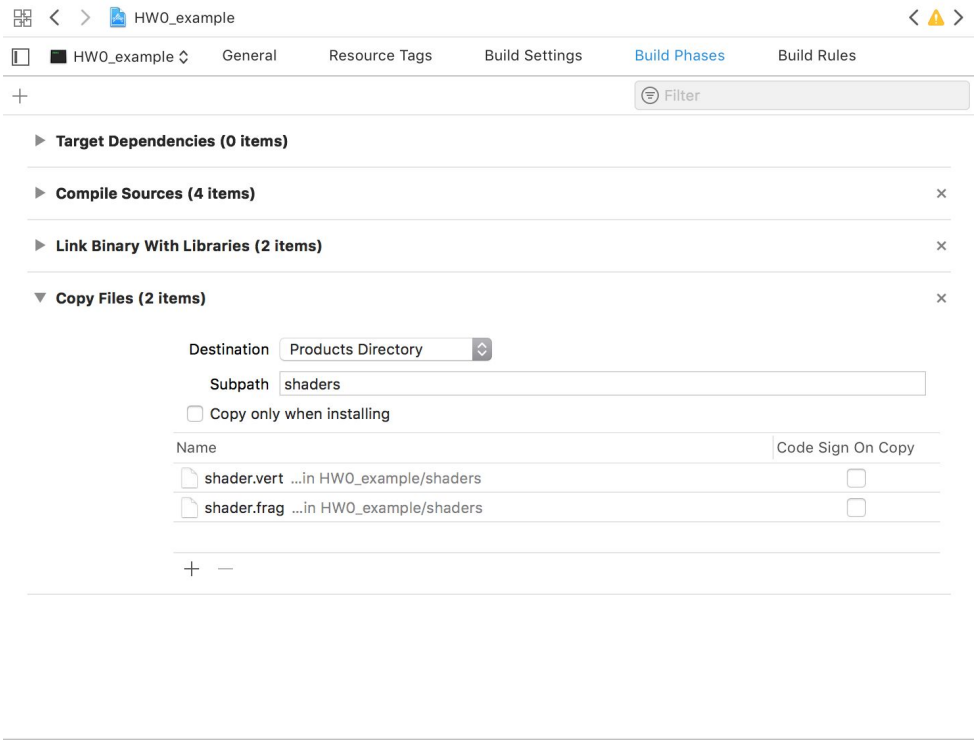
    - Press the little "+" sign and type "OpenGL"
    - Choose "OpenGL.framework" and press "Add"



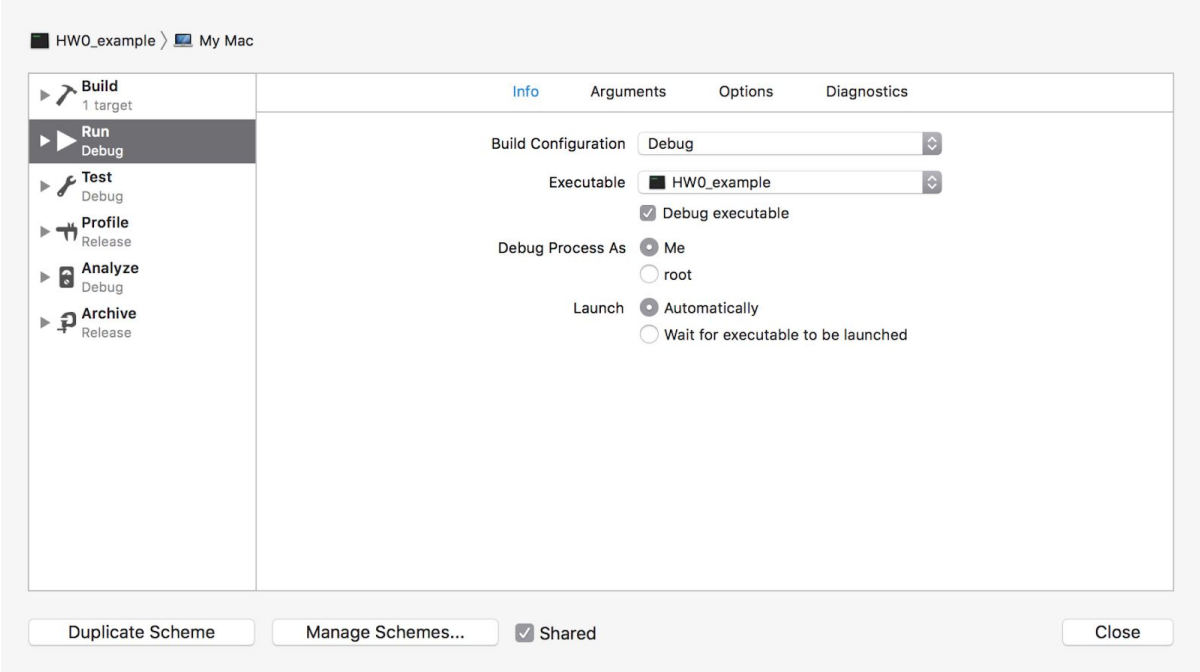
    - Press the "+" sign again and choose "Add Other..."
    - Navigate to /usr/local/lib, choose "libglfw.dylib" and press "Open", use shift+cmd+g to type in any directory you want to go.



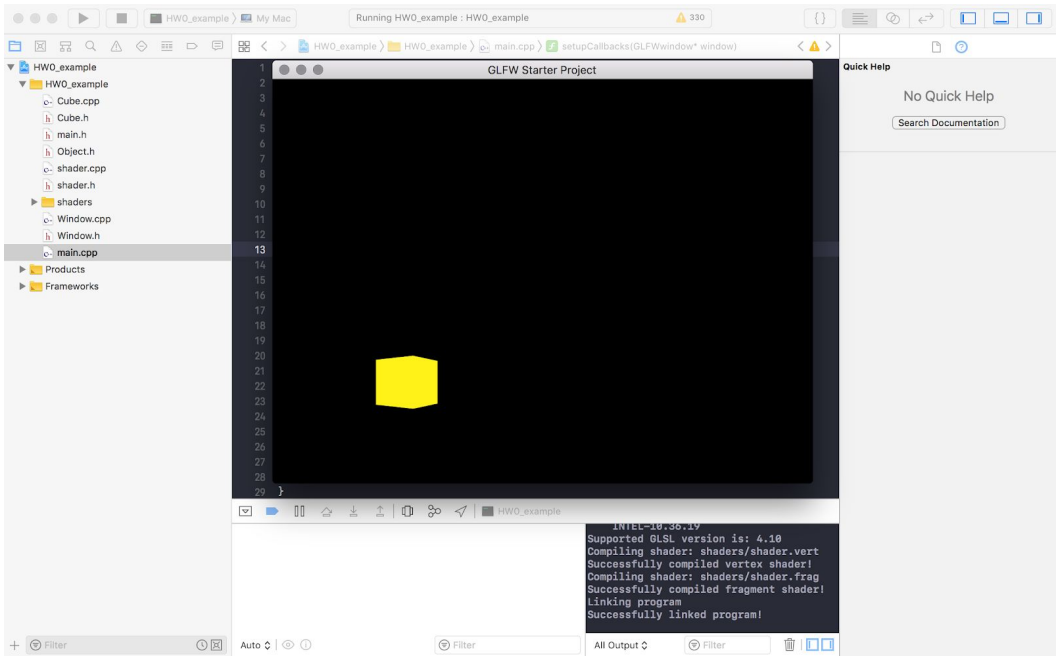
- Open up the "Copy Files (0 items)" pull down
- Select "Products Directory" as the Destination
- Change the subpath to "shaders" and uncheck the "Copy only when installing" box.
- Click on the little "+" sign and choose two shader files



5. Optional: We like to see a good frame rate, so we prefer you run in Release (at least during the grading demo).
    - Open Product > Scheme > Edit Scheme, and choose "Release"



6. Run that baby (press the play button at the top left or press Command + r). Note that the default color of the cube may change depending on what quarter it is (we like being different)



On Linux

The staff don't officially support Linux, but the ingredients you'll need are GLFW3.x.x, GLM 0.9.x.x, and GLEW 1.10+.x. You might be able to find these libraries in your favorite package managers, but make sure the versions are sufficient/compatible.