Games programming Lab01: Getting started with C++ and the Games Template

Lab 1: Objectives

1. Practice setting up a project in Visual Studio (linking, compiling and debugging).
2. Set up helper libraries (OpenGL, if you did not do our 3D Graphics & Animation course).
3. Understand a template Game project and exploring adding your algorithms to it.
4. Write C++ routines to handle vector/matrix operations.

Warning note here

If you are using Google to search for examples, please be careful you only search for the newer C++ and OpenGL code. If you see functions like malloc, calloc, *glRotate*, *glBegin*, *glEnd* then you are most likely reading old code that is not used anymore. (* So, that solution from StackOverflow that you just found might not be the best solution as it can be using really old code.)

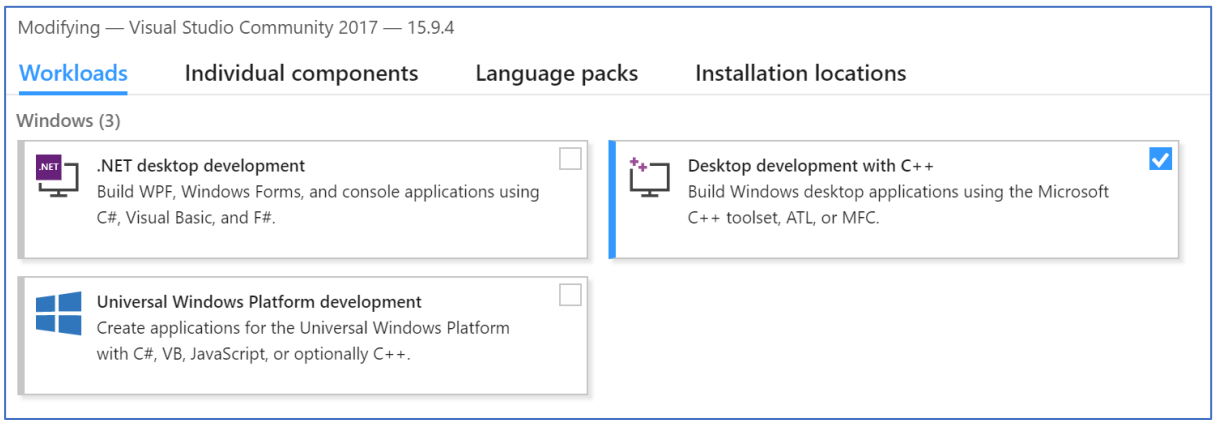
Visual Studio and a simple text game

In this course we are going to be using Visual Studio 2015/2017, C++ and OpenGL libraries.

We also need to use some open-source libraries to help us load our context (GLFW), maths (GLM), and extensions (GLEW).

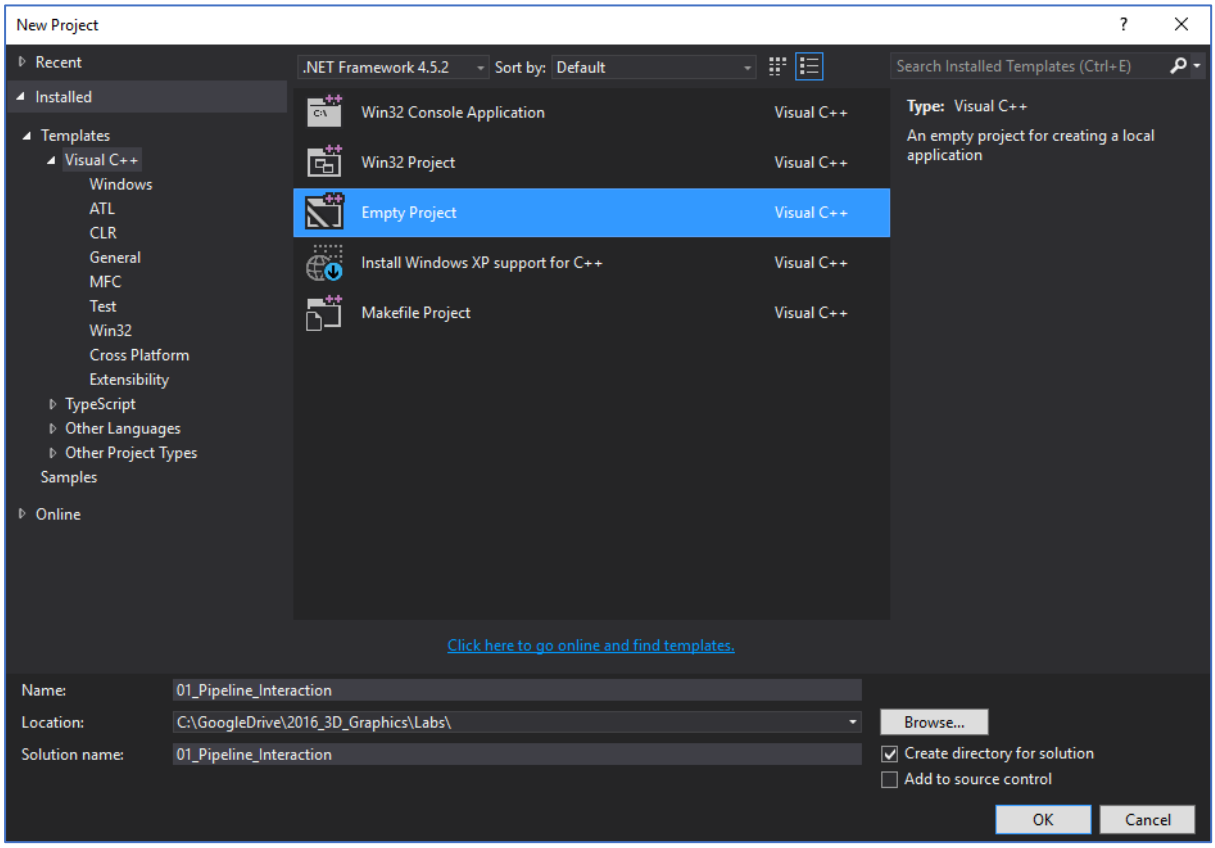
Visual Studio 2015 and 2017 is already installed in the MACS lab machines.

If you want to download it for your own laptop just go to https://www.visualstudio.com/ and select the community edition (free). Do not forget to select to install the C++ components and Windows SDKs. (Visual Studio install >> More >> Modify >> Desktop development with C++).



Extra task for beginners: Try to create a small game with no graphics, just text. Do not forget to allocate memory and use pointers if needed. Instruction below should get you started.

To create a new project click on:
File >> New >> Project and then
Templates >> Visual C++ >> Empty Project

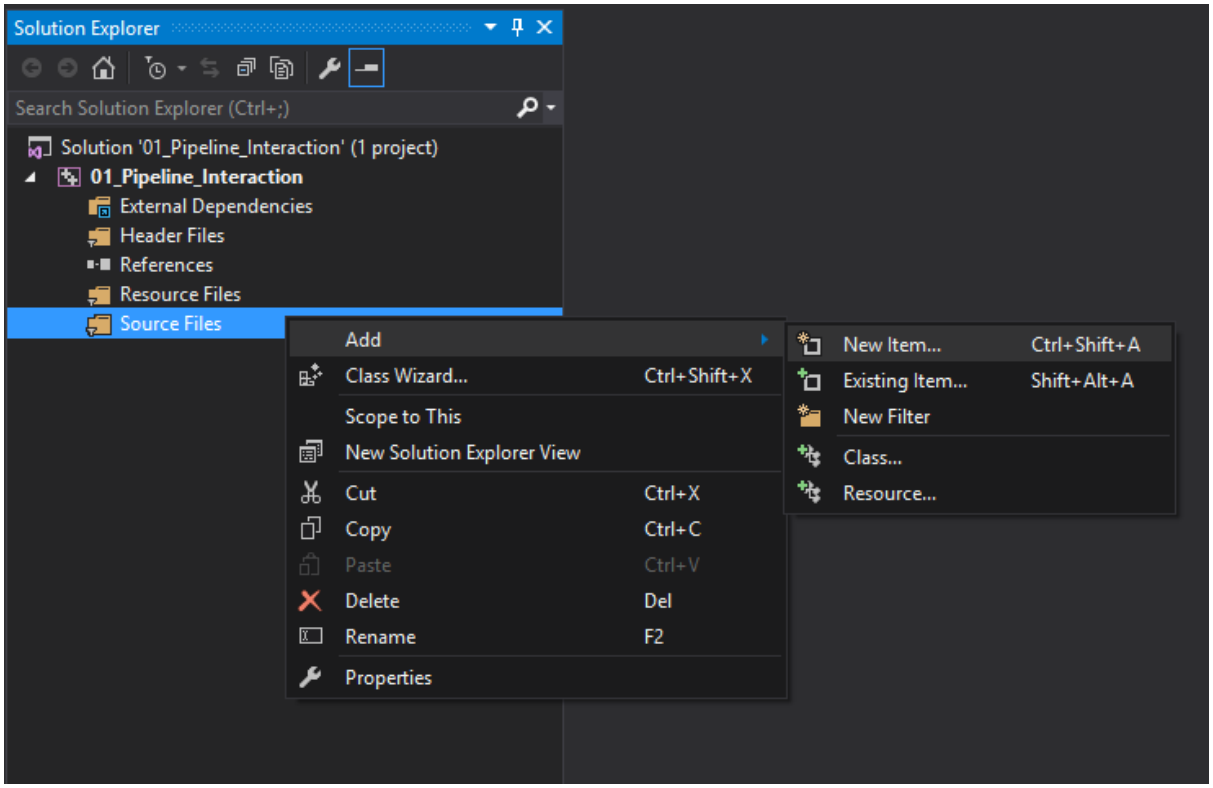


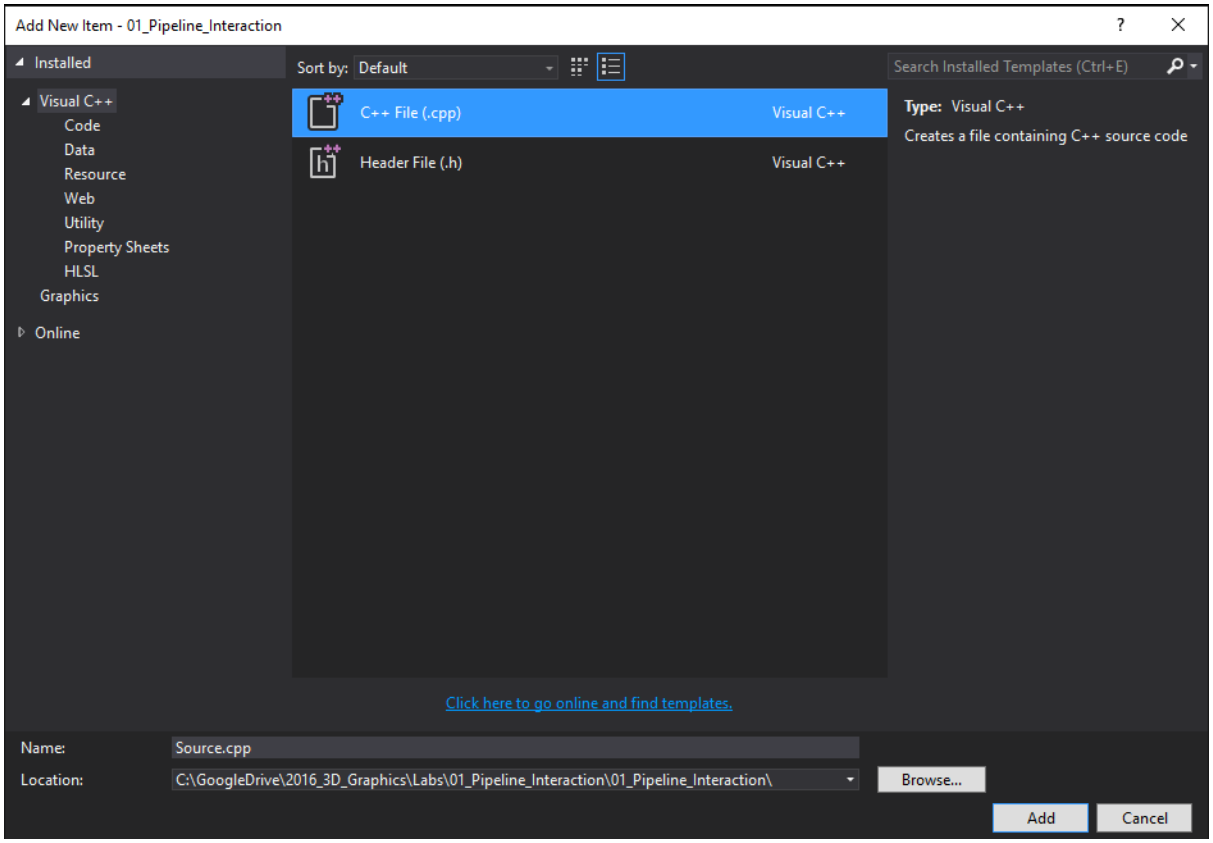
Visual Studio just created an empty project. To start let's by creating a simple C++ source file.

Note: there was a catch-up lecture on C++ and it should be available in Vision ... for now, if you missed that lecture, just create a simple source file in the *Source Files* directory with this code (see below):

```
#include <iostream>
using namespace std;
int main()
{
	cout << "I'm alive!\n";
	cout << "\nPress any key to continue...\n";
	cin.ignore(); cin.get(); // delay closing console to read debugging errors.
	return 0;
}
```

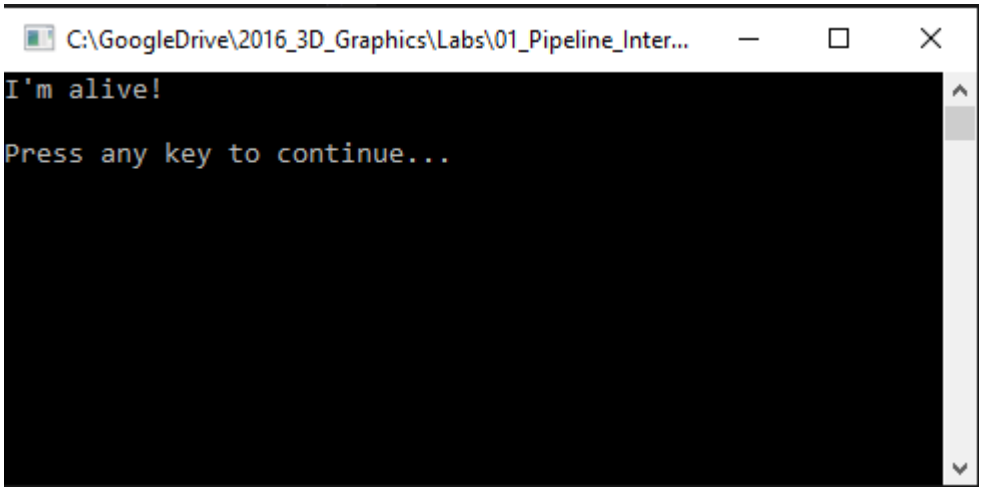
Right click on *Source Files* folder to add C++ file...



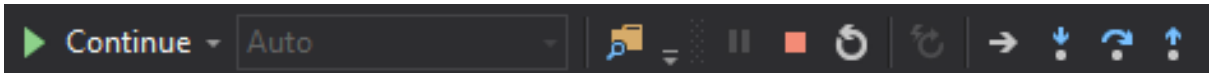


Compile and run your code in debug mode (▶) :

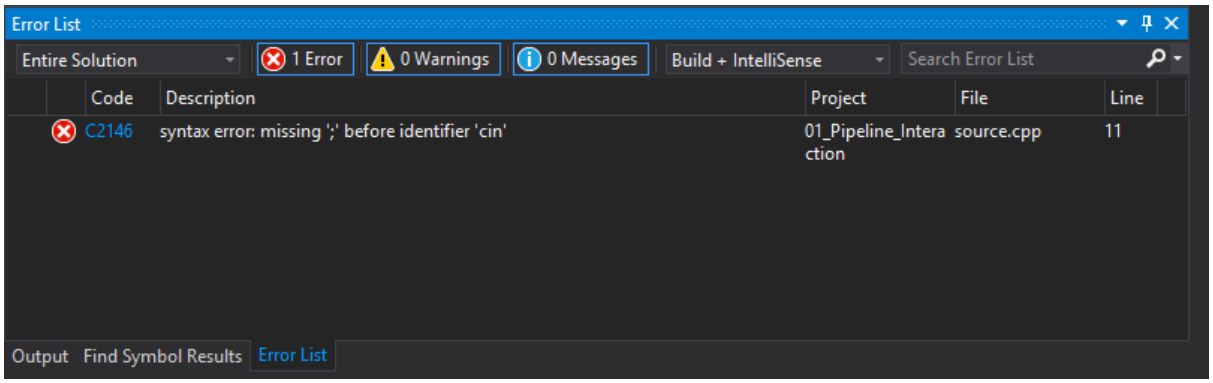
If everything was fine, you should see this window:



Debug using the usual methods and strategies:



If something went wrong, then you can check the Error List. If you can't see this window then press CTRL+\ and e.

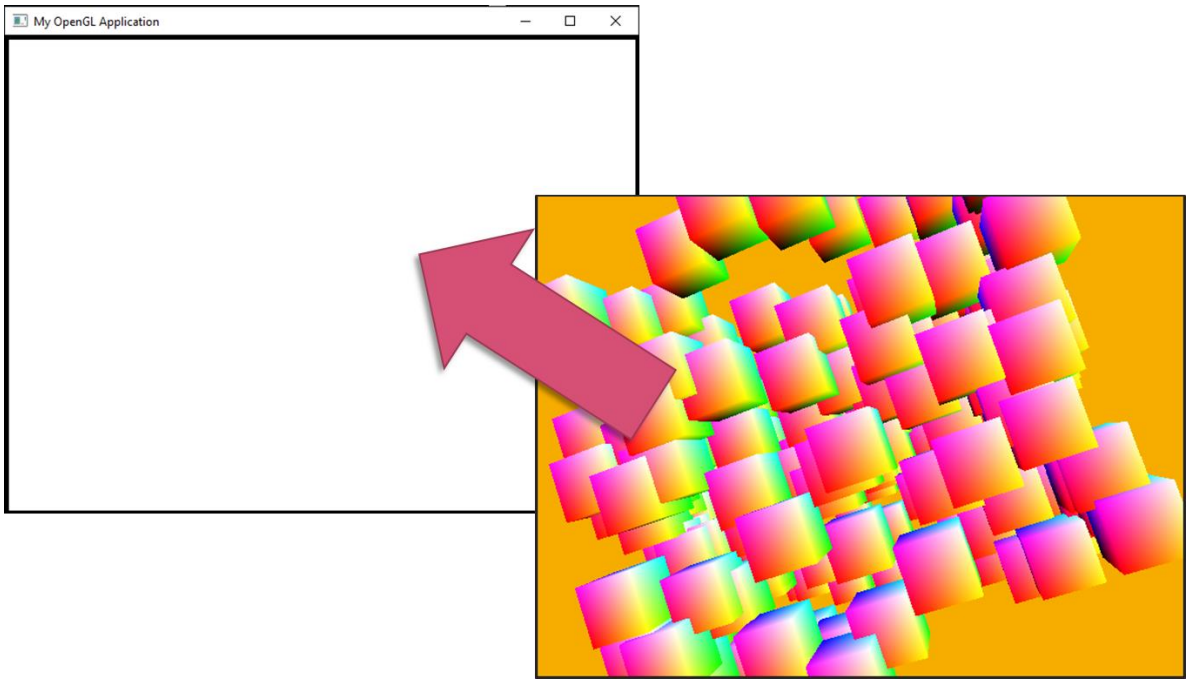


Helpful notes:

- Visual Studio C++ Intellisense is access with CTRL+Space
- Reset Visual Studio Layout using ALT+w and r

A game with graphics (Window Context)

We are going to load a window with an OpenGL context (aka somewhere to render) to it.



To start, download the template from GitHub link below and unzip the files in your machine.
https://github.com/StfnoPad/GPGameTemplate

We are going to use GLFW for our programs. This is an Open Source, multi-platform library for creating windows, contexts and handling inputs and events. This library was added to the GitHub Template. You can also download the files here http://www.glfw.org/download.html (**32-bit Windows Binaries**). I will show you the basic setup here but please read the documentation if you want to know more about the library or if you are stuck.

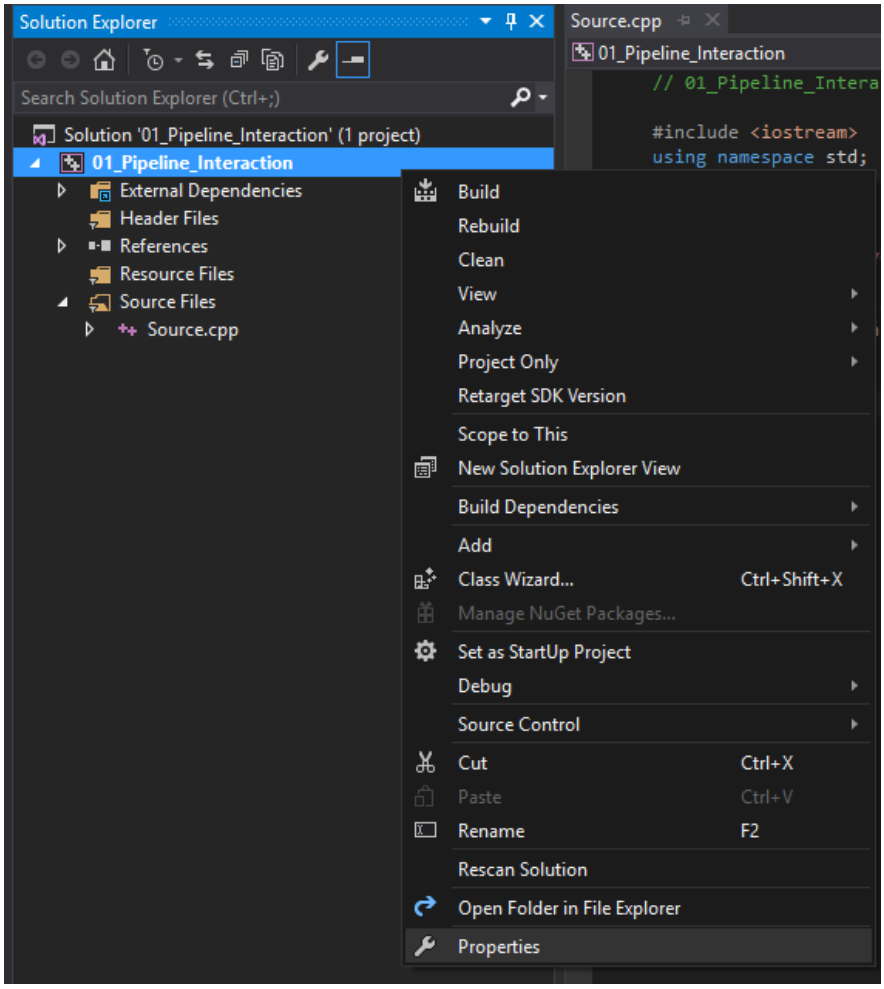
In addition, we are going to be using the OpenGL Extension Wrangler Library (GLEW) to provide us with the newest functions for our rendering context. We need GLEW on Windows as it only provides functionality for OpenGL 1.x. This library was also added to the GitHub file. You can download the library also from http://glew.sourceforge.net (32-bit and 64-bit Windows Binaries).

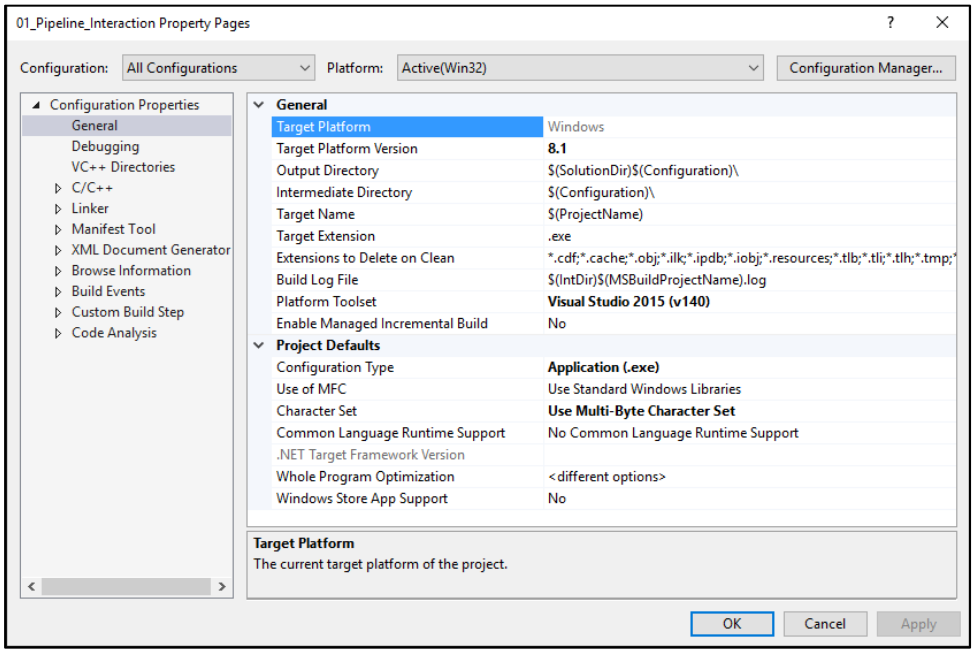
Finally, to do some of the graphics rotations, translations and projections, we use the OpenGL Mathematics Library from G-Truc (https://glm.g-truc.net). This library is also included in the GitHub files and only requires to be header linked.

Correctly linking GLFW, GLEW and GLM

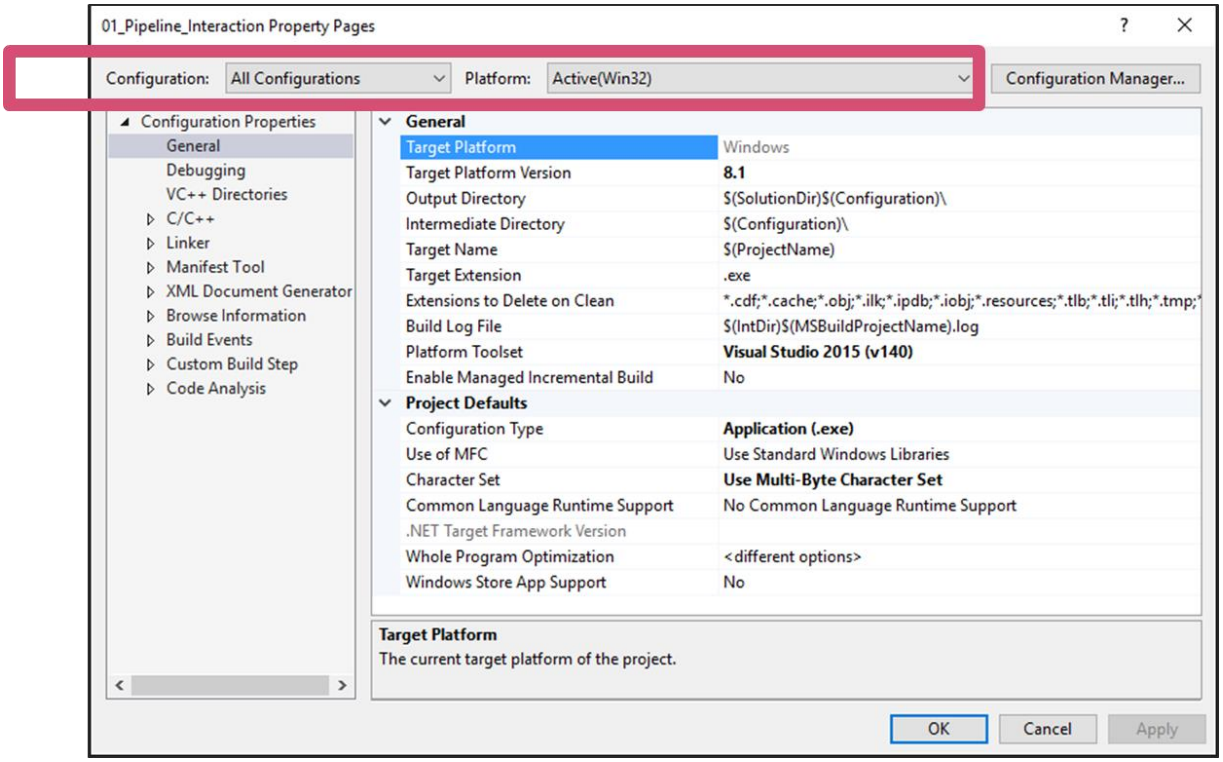
If you compile the project from GitHub in Visual Studio, *these should already be linked and added to your project properties*. Instructions below are there if you created your own project or want to practice a bit more about linking libraries. To link these your our project:

1) Right click on the Project name (see below) and open properties.





2) Make sure you select All Configurations and Active(Win32).



3) Update the links to the libraries (Use the appropriate path depending on where you saved your files).

VC++ Directories >> Include Directories:

- $(SolutionDir)../Libraries\glfw-3.2.1.bin.WIN32\include
- $(SolutionDir)../Libraries\glew-2.1.0\include
- $(SolutionDir)../Libraries\glm-0.9.9.1\glm
- $(SolutionDir)../Libraries/stb_image

VC++ Directories >> Reference Directories:

- $(SolutionDir)../Libraries/stb_image\glfw-3.2.1.bin.WIN32\lib-vc2015
- $(SolutionDir)../Libraries/stb_image\glew-2.1.0\bin\Release\Win32

VC++ Directories >> Library Directories:

- $(SolutionDir)../Libraries/stb_image\glfw-3.2.1.bin.WIN32\lib-vc2015
- $(SolutionDir)../Libraries/stb_image\glew-2.1.0\lib\Release\Win32

C/C++ >> Additional Include Directories:

- $(SolutionDir)../Libraries/stb_image\glfw-3.2.1.bin.WIN32\include

- $(SolutionDir)../Libraries/stb_image\glew-2.1.0\include

Linker >> Additional Library Directories:

- D:\GPGameTemplate-master\glfw-3.2.1.bin.WIN32\lib-vc2015
- D:\GPGameTemplate-master\glew-2.1.0\lib\Release\Win32

Linker >> Input >> Additional Dependencies:

opengl32.lib
glew32.lib
glew32s.lib
glfw3.lib
glfw3dll.lib

A simple template game program

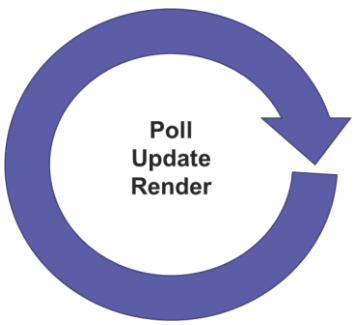
The purpose of the skeleton template program is to bring up a window and run the main rendering loop.

Note that it:

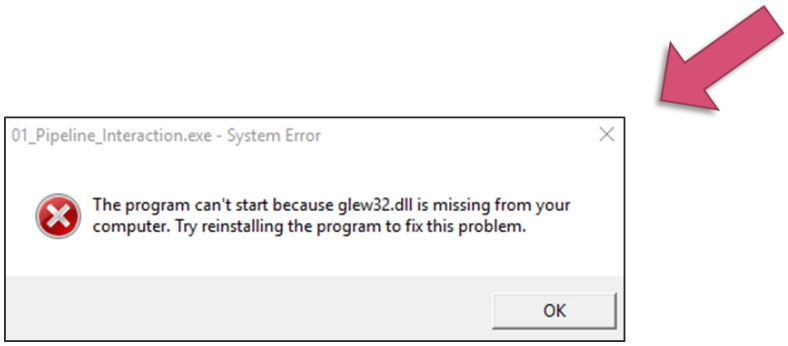
- includes GLFW and GLEW
- starts GLFW, checks for errors and inserts an error handler
- uses a hints function to specify OpenGL context and debug
- creates a window and attaches the context
- produces a rendering loop
- Uses a skeleton render function to define a viewport and clear the screen

The rendering loop

Most interactive systems and operating systems work using events loops (https://en.wikipedia.org/wiki/Event_loop), basically it is a loop that handles all the events for a window. These events include window (resizing, exiting), mouse and keyboard events. Our template program includes a loop to poll events, update, and render our graphics -- until the user decides to exit the application.



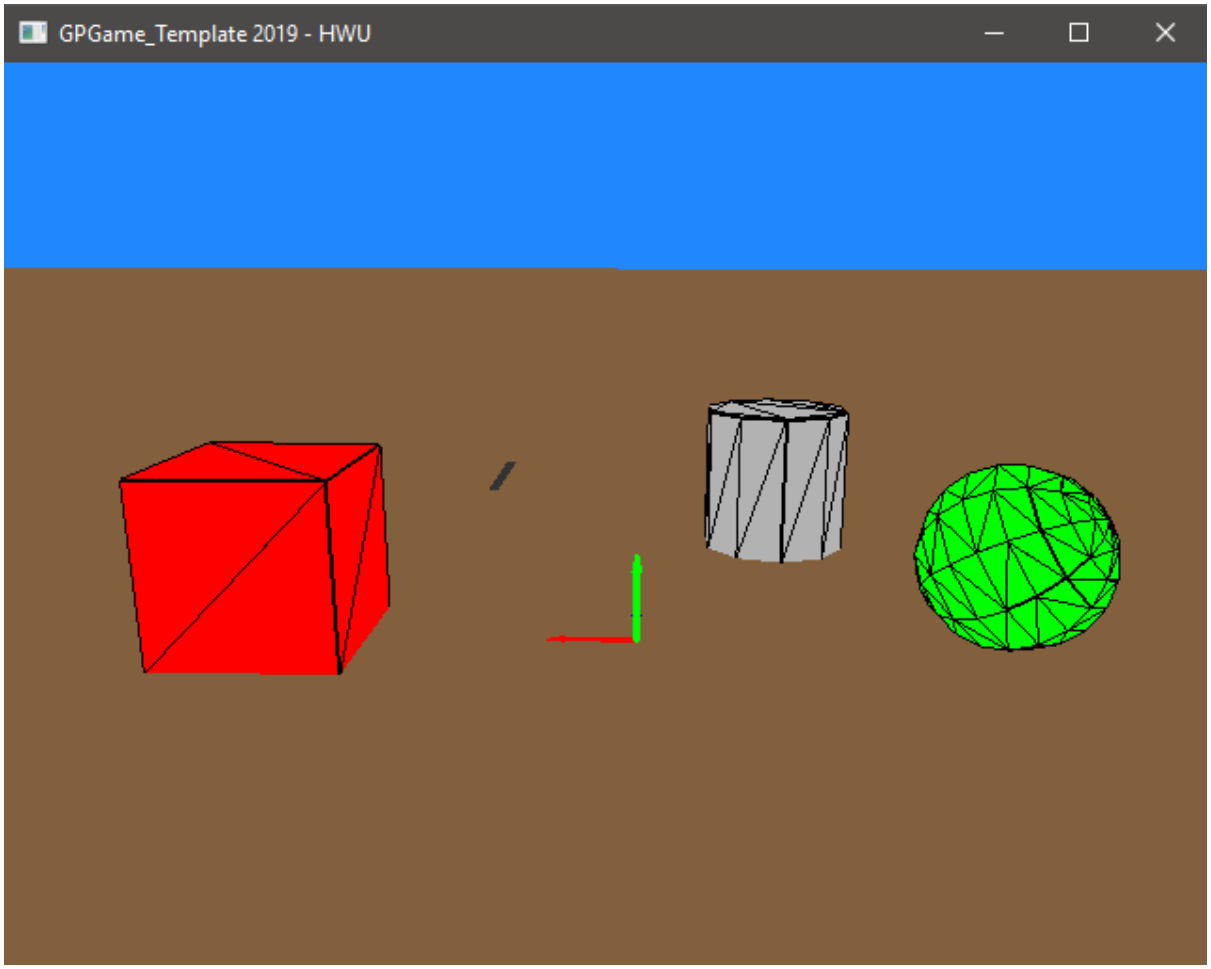
Try running the template program. If it crashed, see below:



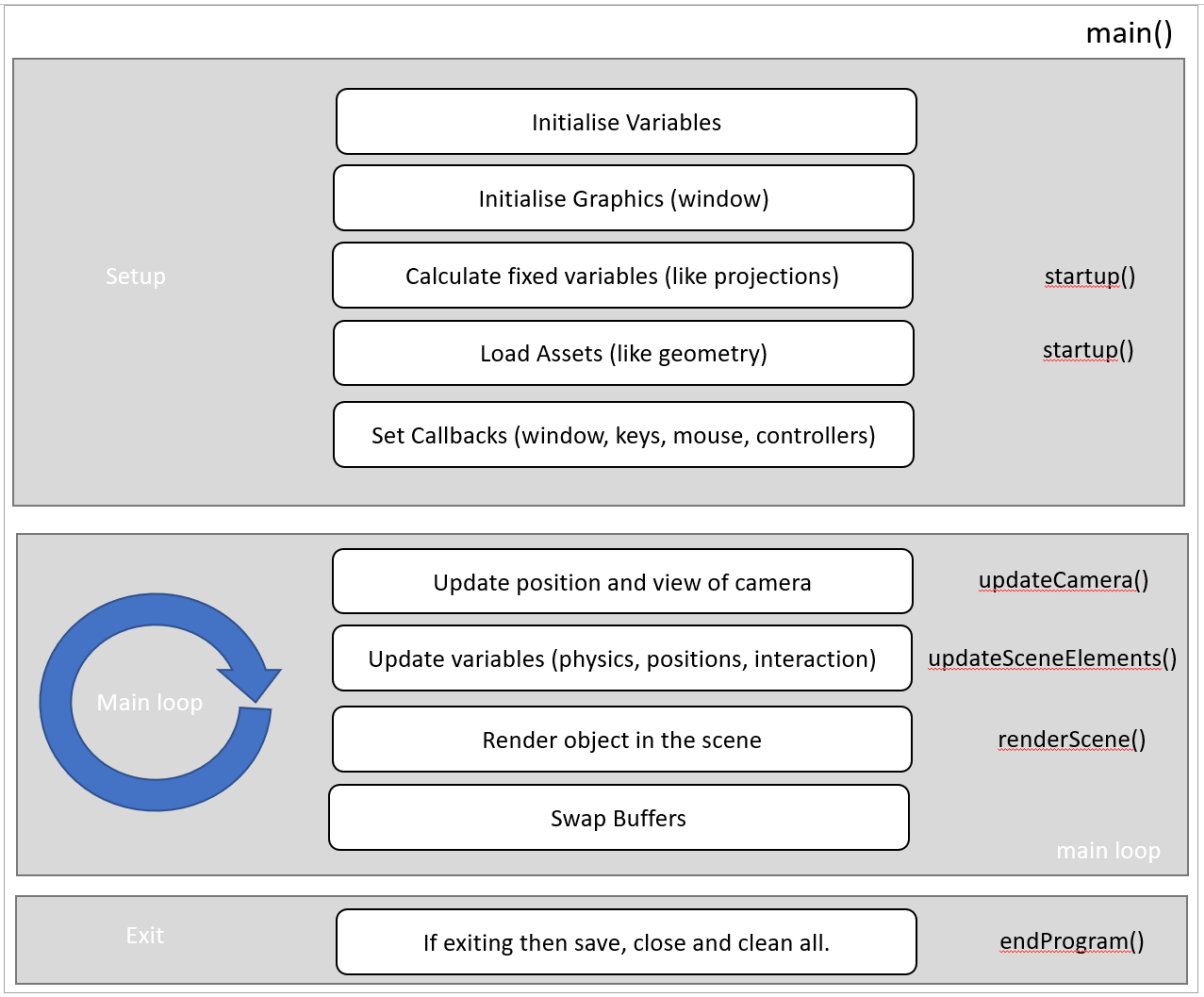
GLEW uses dynamic loaded DLLs instead of static loaded DLLs (ones that we can define in the project properties) - as so we need to add them these DLLs to your program.

The easiest way to do it is just to copy glew32.dll (glew-2.0.0\bin\Release\Win32) to your project folder (ProjectName/ProjectName). If you see your source.cpp file then you are in the right folder.

Everything should be running perfectly now.



The template programme code follows the essential structure of most games (setup, a loop, and exit):



Now try experimenting and understanding the different functions and how they affect the template program.

Advance materials: Game engines also follow a standard order of execution, here is the one for Unity https://docs.unity3d.com/Manual/ExecutionOrder.html

A few more exercises for beginners

These tasks are designed to help you in your coursework.

Task 1: try to add code to animate and rotate the cube. Can you do it when you press a key or when you move your mouse.

Hint: the rotation and displacement is calculated in these lines:

```
// Calculate Cube position
glm::mat4 mv_matrix_cube =
       glm::translate(glm::vec3(2.0f, 0.5f, 0.0f)) *
       glm::mat4(1.0f);
myCube.mv_matrix =  myGraphics.viewMatrix * mv_matrix_cube;
myCube.proj_matrix = myGraphics.proj_matrix;
```

Task 2: add more elements to your scene and try to create a more complex object.

Task 3: using lines, try to create an enclosure or bounding box for all the objects in your scene.