Assignment 2: Introduction and Setup of OpenGL

**CS Spring 2018**

*Due Date:*

Follow the instructions carefully. If you encounter any problems in the setup, please do not hesitate to reach out to TA.

**Requirements**

This section lists the requirements for compiling OpenGL 4 code on different system platforms. In the next sections, we will go over the details for Windows, macOS, and Linux.

• C++ Compiler with C++11 Support

• OpenGL 4.1 Support

• CMake 2.8+

• Git

**macOS** You will need to manually compile this library:

• Open Asset Import Library

You will also need to extract the SDL2 framework file (instructions below).

**Windows** On Windows, we have provided pre-built binaries.

**Linux** You will need to manually compile/download the following libraries from your package repository:

• SDL 2

• Open Asset Import Library

• FreeImage

* GLEW

**Windows Setup**

**C++ Compiler** Download the latest version of Visual Studio (Community Version) to make sure you have Microsoft’s latest compiler. Visual Studio does not come by default with the C++ tools needed so you will have to open Visual Studio and create a Visual C++ project before running CMake. Any version of Visual Studio after (and including) Visual Studio 2013 should also work should you already have it.

**OpenGL 4.1 Support**  Your system should have the OpenGL headers already. Make sure your graphics card drivers are updated from Nvidia/AMD/Intel.

**CMake** Download the latest CMake binaries from the CMake website.

**Linux Setup**

Note that the instructions here are tailored to Ubuntu. If you have a different distribution, the steps should be similar as the only major difference should be the package names.

**C++ Compiler** On Linux, you’ll want to be using G++ 4.6 or later. If possible, you’ll want to be using G++ 4.8.1+. On Ubuntu you can run:

sudo apt-get install g++-4.8

If you are on the latest version of Ubuntu (Ubuntu 16), you can instead get G++ 5:

sudo apt-get install g++-5

You will also be needing to run the ‘make’ command so generally it is recommended that you install the ’build-essential’ package (on Ubuntu).

sudo apt-get install build-essential

The G++ 4.8 package is available on Ubuntu 14.04 LTS or later. If you have another distribution, check your distribution package manager to see what version of G++ they have.

**OpenGL 4.1 Support** Make sure you have the latest drivers for your GPU. After you installed the drivers (be it open source or closed source), you can double check to make sure you have OpenGL 4.1 by installing the “mesa-utils” package on Ubuntu (may be named “mesa-demos” on other distributions). Then in the terminal run the command below to make sure you get a version that’s greater than or equal to 4.1!

glxinfo | grep "OpenGL version"

**CMake** You can download the latest CMake using your package manager. On Ubuntu, you can run:

sudo apt-get install cmake

**SDL 2** SDL 2 should be available in your distribution’s package repository. On Ubuntu:

sudo apt-get install libsdl2-2.0-0

sudo apt-get install libsdl2-dev

If it is not in the repository, you will have to compile manually. See a CA if you need assistance.

**Open Asset Import Library** We have provided the Open Asset Import library source code with the assignment framework under “external/assimp.” Create a build folder “external/assimp/build/unix”. In our terminal, go to the build folder and run:

cmake -G "Unix Makefiles" ../../

make -jN

Note that you should run the above commands in the “external/assimp/build/unix” folder. Where N is the number of cores you want to use for compilation. Afterwards, in the ‘external/assimp/build/unix/code’ folder, there should be shared object library files for assimp. Copy them using the following command:

cp external/assimp/build/unix/code/libassimp.so\* external/assimp/distrib/unix/

Note this command assumes you are in the root folder for your project. If you are in the “external/assim-p/build/unix” folder, run:

cp code/libassimp.so\* ../../distrib/unix/

**FreeImage** FreeImage should be available in your distribution’s package repository. On Ubuntu:

sudo apt-get install libfreeimage3

sudo apt-get install libfreeimage-dev

If it is not in the repository, you will have to compile manually.

**GLEW** GLEW should be available in your distribution’s package repository. On Ubuntu:

sudo apt-get install libglew1.10

sudo apt-get install libglew-dev

Other versions of GLEW should be okay. People using Windows/OSX will be using GLEW 1.13. If it is not in the repository, you will have to compile manually.

**Compilation**

Compiling is a two step process regardless of whether you are using the CMake GUI or the CMake Command-Line Interface (CLI). First you will use CMake to generate the proper build files (Visual Studio, XCode, or Makefiles) and use Visual Studio/Xcode/Makefiles to actually build the application (aptly named cs148opengl4). This section assumes that you have the assignment code already downloaded somewhere. Before you start, make sure you create a “build” folder inside the project directory (i.e. the build folder should be in the same folder as the “common” folder as well as the “assignment” folders). The location of the “build” folder technically does not matter; however, unless you know what you are otherwise doing, follow the instructions as is. If you will be using the command line, we recommend also creating a “build/Release” and a “build/Debug” folder.

**CMake Configuration and Generation**

**Special Notes for Visual Studio Solutions and XCode Projects** If you are using CMake to generate for Visual Studio or XCode, you can run CMake once (the very first time) and then use Visual Studio or Xcode to manage any new files you might add (instead of using CMake to do that for you). However, should you ever run CMake again, you will overwrite your Visual Studio solution/XCode project and lose any changes you made.

**GUI (Windows, Mac OSX)** Open CMake and point ‘Where is the source code’ to the root directory of your project located wherever you downloaded the assignment framework (i.e the parent directory of the ‘build’ folder) and point ‘Where to build the binaries’ to the ‘build’ folder. If you do not have a “build” folder, create one in your project’s root directory. At this point, you can click the Configure button. You should be greeted by a screen to choose the generator for the project. On Mac OSX you will want to choose “Xcode” (or Unix Makefiles, only do this if you know what you are doing!) and on Windows you will want to choose whatever version of Visual Studio you have. Note: On Windows, you must also choose the Win64 version of Visual Studio instead of the Win32 version. Now press the Generate button—your Visual Studio solution or XCode project should exist within your previously selected build folder.

**CLI (Linux, Mac OS X)** Usually, you’ll only be using the CLI on Mac or Linux so we will be ignoring Windows here. If you want to generate the build files for XCode, first go into the “build” directory and then run:

cmake ../ -G "Xcode"

Afterwards, the file “cs148-opengl4.xcodeproj” will exist and you will be able to open it up in XCode.

If you want to generate Makefiles, go into “build/Release” or “build/Debug” folder. Then you will want to run:

cmake ../../ -G "Unix Makefiles" -DCMAKE\_BUILD\_TYPE=CONFIG

In this case, CONFIG is either Release or Debug depending on which folder you are in.

**Grading**

This assignment will be graded on the following requirements

• OpenGL compiles on your computer and test program runs successfully. You are expected to see two red, blue, and green triangles.

according to the following rubric.

• – Meets the requirement.

• – OpenGL does not compile and test program does not run.