CSE 167 (WI 2025) Homework 0 — Due 1/17/2025

Homework 0 is here to make sure that you can compile and run modern OpenGL, and that you understand how to modify the code to generate desired output image. In case you're mystified by the compilation process, see Lecture note Chapter 2 and Appendix A for more background about compiling OpenGL and compilation in general.

0.1 Overview

All of our programming assignments are C++ programs using a few external libraries listed below. Our own program consists of one or multiple files. In particular, projects that consist multiple C++ source files require the two-stage compiling and linking. The compilation and linking commands have already been set up for you. The only challenge is to make sure the linking using the external libraries are correct. The external libraries are

- OpenGL: Core OpenGL implementations. (On Windows and Linux, the header is GLEW)
- GLUT: A cross-platform library for managing window, keyboard, mouse events.
- GLM: Math library.
- FreeImage: Library for image input/output.

Depending on your operating system, including and linking with these external libraries are different.

0.2 Compile (Operating System Dependent)

0.2.1 Windows

All the external libraries (headers and binaries) have been embedded in the local folder. Their paths are set up using CMake.

Install **Visual Studio** with **C++** support (note that a default installation of Visual Studio only includes C#). Once your Visual Studio is boot up, choose "Open a local folder", then select the folder where `CMakeLists.txt` resides in, which should be the `hw0-windows\` folder if you directly unzipped the starter code. Then, navigate to the "Build" dropdown and choose "Build All". Once both executables are built, click the green start button (or press F5) to run them.

0.2.2 MacOS

Mac has its own implementations of OpenGL and GLUT. These Mac-native libraries are called "frameworks." The OpenGL and GLUT frameworks come with the **macOS SDK** (software development kit) contained in the **command line tools**. Command line tools also consist essential tools such as `g++`, `make`. If you have used your Mac to compile program before, you probably already have the command line tools.

The external libraries GLM and FreeImage can be installed via the **homebrew** package manager.

1. If you have not done so already, install **homebrew**. If homebrew is installed, run "`brew update`" in a terminal. (To check whether homebrew has been installed, open terminal and type "`brew update`" anyway. If we see "`Command not found: brew`" then homebrew has not been installed.) To install homebrew, go to homebrew's website and copy the installation command to your terminal.
2. Install **command line tools**. This one is easy, during the homebrew installation of the previous step, we will see a massage "The Xcode command line tools will be installed." Press return to continue. Alternatively, type "`xcode-select --install`" in a termi-

nal. A pop-up dialog will show up; click "install." This will install the command line tools without installing the full-blown Xcode App. Another way is to simply install the full Xcode App from App Store.
At this point you have OpenGL and GLUT frameworks in

```
/Library/Developer/CommandLineTools/SDKs/ \
    MacOSX.sdk/System/Library/Frameworks/
```

3. Install GLM and FreeImage. In the terminal, type

```
brew install glm freeimage
```

4. Finally we can compile our program. Via terminal `cd` to the directory containing the files, type "`make`" which should generate the executables "`HelloGL`" and "`HelloSquare`" if the compilation and linking are successful. Run the executables by `./HelloGL`, and `./HelloSquare`. (Don't double click on the executable file using finder in MacOS; that will lead to errors, since the paths would not be set up properly.)

Possible trouble shoot for the Apple M1 chip

Some of the Mac users run on a CPU of different architecture, namely the `arm64` architecture on M1 chips, instead of the traditional `x86_64` on an Intel processor (You can type "`arch`" on a terminal to see your architecture). In theory, when we use homebrew to install the libraries, it downloads the binaries of the correct architecture. However, some of us found that brew downloads the `x86_64` binaries for the FreeImage binaries. This causes error in the linking process as it tries to link our `arm64`-built binaries with an `x86_64` binary. In that case, a simple solution is to force all our builds as `x86_64`. To do so, open the `makefile` with a text editor and modify the first line from

```
CC = g++
```

to

```
CC = g++ -arch x86_64
```

It will compile and link everything under the `x86_64` version. You might still be able to run the executable from the terminal:

```
./HelloSquare
```

If terminal is complaining that this executable is of bad CPU type, then run the executable by

```
arch -x86_64 ./HelloSquare
```

0.2.3 Linux

All the external libraries (headers and binaries) have been embedded in the local folder. Use the command line (*e.g.* via terminal) to `cd` to the directory containing the files, type `make` which should generate the executables `HelloGL` and `HelloSquare` if the compilation and linking are successful. Run the executables by `./HelloGL`, and `./HelloSquare`.

0.3 HelloGL and HelloSquare

`HelloGL` is a minimalistic OpenGL program. It creates an empty window with some background color. `HelloSquare` is a more interesting OpenGL program. `HelloSquare` draws a square, shades the square with some color interpolated from the vertices, and shades a white circle in the middle (Figure 1).

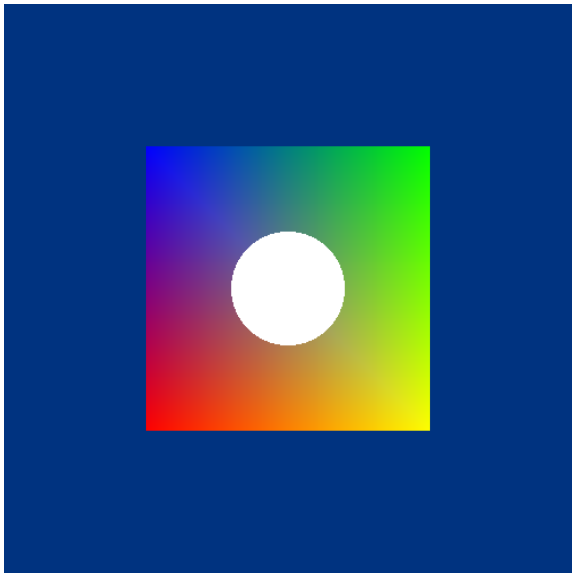


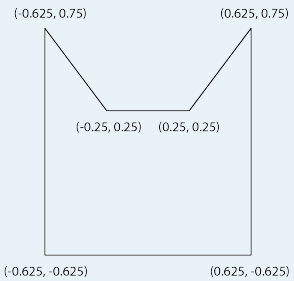
Figure 1 `HelloSquare`

For `HelloSquare`, you can press O to export a screenshot of the viewport.

(Note that the viewport in the window may be off-centered for some display hardwares, such as some MacBooks with Retina Display. In that case, resizing the window can pop the viewport back to fit the window. But if you want to export a screenshot, please do not rescale the window (since the screenshot would be capturing the wrong portion of the viewport). If you are exporting a screenshot, just ignore the fact that the viewport is smaller than the window.)

Programming 0.1 — 2 pts. Upload the `.png` file exported from `HelloSquare`. ■

Programming 0.2 — 3 pts. Duplicate the HW0 folder to keep a backup, because we want to modify the source code. Modify the source code `HelloSquare.cpp`, so that the square is replaced by the following shape



You may also modify the color or even the fragment shader to fill in some color and pattern in the shape. Upload the `.png` file exported from the modified `HelloSquare` and upload the modified `HelloSquare.cpp`. ■