Unsupported MacOS Installation Instructions

1 Important Preface

The following are instructions for building and running the source template on a device running MacOS. Although we provide these instructions, we do not officially support MacOS with this course. Use either a Linux machine (available in the department lab) or a Windows machine unless absolutely necessary. All submitted assignments must build and run on the department machines, and this may not be the case if you configure it to run on MacOS. Make sure you test on a department machine to be sure it runs before submitting.

Much of the time these instructions are not enough to get your code running on MacOS. Due to the deprecation of OpenGL on MacOS as well as the introduction of Apple Silicon, there will likely be a plethora of other issues that need to be resolved. Depending on your specific device, there may not even be a workaround. Resolving the issues that arise with MacOS will require a high level of knowledge of OpenGL and C++ in general. Do not try to do this if you are not familiar with both.

2 Alternatives and Frequent Issues

1. If you own a Mac device that uses Intel hardware, and not Apple Silicon, using Boot-Camp to install a Windows partition on your device is an option to develop using Windows. We have had students do this in the past and it seems to work, but we do not endorse it.

2. If your device uses Apple Silicon, there are alternatives to emulate Windows on ARM, but they will likely not work well and we do not endorse any of them.

3. One of the frequent issues with building on MacOS is due to the libraries not being found during the compilation process. Sometimes this is because Homebrew installs them in a different location than is assumed in the CMakeLists file. Inside of the CMakeLists file, consider adding the 2 lines

   "include_directories("/opt/homebrew/include")"
   "link_directories("/opt/homebrew/lib")"

   after the "include_directories(/usr/local/include)"
   "link_directories(/usr/local/lib)" lines.

4. Another common issue is that the main() function in main.cpp is not recognized properly. In this case, the issue is because the signature is not accepted by some MacOS compilers. This can often be fixed by replacing the signature of the function to:

   int main(int argc, char* argv[])

5. Finally, another common issue arises due to OpenGL being deprecated for MacOS. This means that features for newer versions of OpenGL are not available, so when they are used they either not compile or give errors when running. An example is that in shader code, the layout() format cannot be used in some MacOS devices. Sometimes this can be replaced with a usage of a uniform variable, but it can be quite inconvenient to not have access to this feature, among many others.

3 Instructions for A1

1. Download and unzip the source template. It should match the structure specified in the Template section of this document. The package can be downloaded from the course website.

2. The template is built using CMake, installed as detailed in the preliminary assignment.

   For MacOS users, install dependencies using Homebrew (`https://brew.sh/`):

   ```
   brew install pkg-config
   brew install glfw3
   brew install sdl2
   brew install sdl2_mixer
   ```

   And for MacPorts (`https://www.macports.org/`):

   ```
   port install pkgconfig
   port install glfw
   port install libsdl2
   port install libsdl2_mixer
   ```

   Create an empty directory as the build directory, which we assume is named `build`, you could place it in `template/build`. Note that running CMake and building the project will copy files and data to this folder. **Do not edit any files in the build folder** since they can be overwritten during the build process. Only edit files in the `src` and `shader` folders (create new assets in `data`).

   You can configure the project using CMake GUI or the command line. For the GUI, enter the assignment template folder (which should contain a CMakeLists.txt file) as Source and the `build` folder as the Build. Then, press configure, and if the configuration if successful, press generate. For the command line, cd inside the `build` and run:

```
cmake [path_of_assignment_template] -DCMAKE_BUILD_TYPE=[Debug|Release]
```

Now you can build the generated project using make or your favorite IDE. CMake can be configured to output project folders for Visual Studio, Xcode and others. You require a compiler that supports C++14.

3. To verify that the installation was successful, compile and start the program in your IDE. It should start an OpenGL window with a salmon and turtles appearing. Make sure that your debugger works, as detailed in the previous assignment.