NYU Tandon School of Engineering
CS6533/CS4533
Zebin Xu
zebinxu@nyu.edu

Compiling OpenGL Programs on macOS or Linux using CMake

This tutorial explains how to compile OpenGL programs on macOS using CMake – a cross-platform tool for managing the build process of software using a compiler-independent method. On macOS, OpenGL and GLUT are preinstalled; GLEW is not needed as we will use the core profile of OpenGL 3.2 later when we use shaders; Xcode is not required unless you prefer programming in an IDE. At the end we also discuss how to compile on Linux.

Contents/Steps:

1. Install Homebrew

Homebrew is a package manager for macOS. If you have installed Homebrew before, skip this step.

To install Homebrew, simply paste the command from https://brew.sh into your terminal and run. Once you have installed Homebrew, type "**brew**" in your terminal to check if it's installed.

We will use Homebrew to install CMake.

2. Install CMake

I strongly suggest installing CMake via Homebrew as it will also pick up any related missing packages during installation (such as installing a needed command line tool for Xcode even if you don't have Xcode). If you have installed CMake, just skip this step.

To install CMake, simply type "**brew install cmake**" in the terminal. Once you have installed CMake, type "**cmake**" in your terminal to check if it's installed.

3. Build and run the OpenGL program

To compile `example.cpp`, we need an additional file in the same directory: `CMakeLists.txt`, which will be used to generate build files.
CMakeLists.txt:

```
cmake_minimum_required(VERSION 2.8)

# Set a project name.
project(HelloOpenGL)

# Use the C++11 standard.
set(CMAKE_CXX_FLAGS "-std=c++11")

# Suppress warnings of the deprecation of glut functions on macOS.
if(APPLE)
   add_definitions(-Wno-deprecated-declarations)
endif()

# Find the packages we need.
find_package(OpenGL REQUIRED)
find_package(GLUT REQUIRED)

# Linux
# If not on macOS, we need glew.
if(UNIX AND NOT APPLE)
   find_package(GLEW REQUIRED)
endif()

# OPENGL_INCLUDE_DIR, GLUT_INCLUDE_DIR, OPENGL_LIBRARIES, and
GLUT_LIBRARIES are CMake built-in variables defined when the packages
are found.
set(INCLUDE_DIRS ${OPENGL_INCLUDE_DIR} ${GLUT_INCLUDE_DIR})
set(LIBRARIES ${OPENGL_LIBRARIES} ${GLUT_LIBRARIES})

# If not on macOS, add glew include directory and library path to
lists.
if(UNIX AND NOT APPLE)
   list(APPEND INCLUDE_DIRS ${GLEW_INCLUDE_DIRS})
   list(APPEND LIBRARIES ${GLEW_LIBRARIES})
endif()

# Add the list of include paths to be used to search for include
files.
include_directories(${INCLUDE_DIRS})

# Search all the .cpp files in the directory where CMakeLists lies
and set them to ${SOURCE_FILES}.
# Search all the .h files in the directory where CMakeLists lies and
set them to ${INCLUDE_FILES}.
```

```
file(GLOB SOURCE_FILES ${CMAKE_CURRENT_SOURCE_DIR}/*.cpp)
file(GLOB INCLUDE_FILES ${CMAKE_CURRENT_SOURCE_DIR}/*.h)

# Add the executable Example to be built from the source files.
add_executable(Example ${SOURCE_FILES} ${INCLUDE_FILES})

# Link the executable to the libraries.
target_link_libraries(Example ${LIBRARIES})
```

3.1 Build via the Command Line

Now we have two files (CMakeLists.txt and example.cpp) in the same directory. Run the commands to generate the Makefile (Note: $ stands for the terminal prompt):

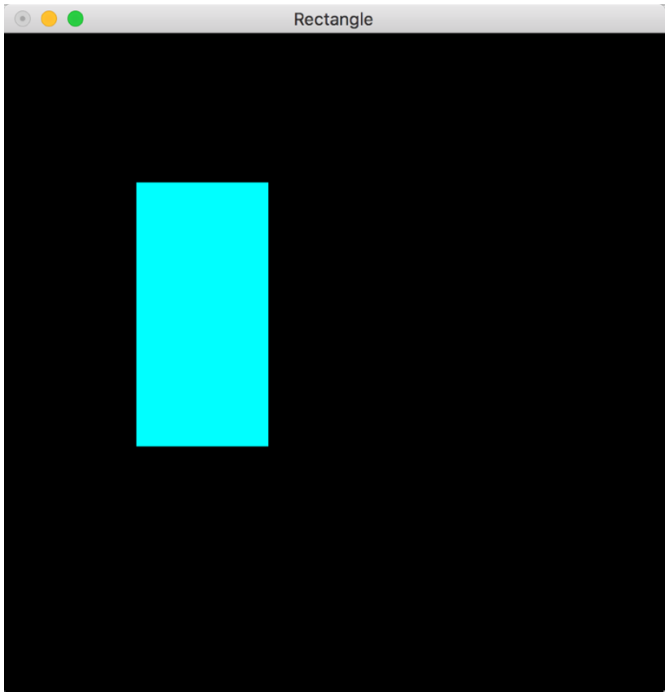
```
$mkdir build
$cd build
$cmake ..
$make
```

The 1st line will create a directory called **build** that will contain all the build artifacts so that you don't mix these files during compilation with the original source files.
The 3rd line will generate Unix Makefile based on the CMakeLists.txt located in the parent directory (.. specifies the location of CMakeLists.txt).
The 4th line will run the Makefile that will compile and link the program.

To **run the program** in the command line,

```
$./Example
```

Now you should be able to see the OpenGL window.



If you update your code without adding or deleting files, just run **`make`** to recompile and link. Otherwise (adding or deleting some files) you may need to make some changes to the CMakeLists.txt file, remove the build directory, and follow the above steps to regenerate the Makefile.

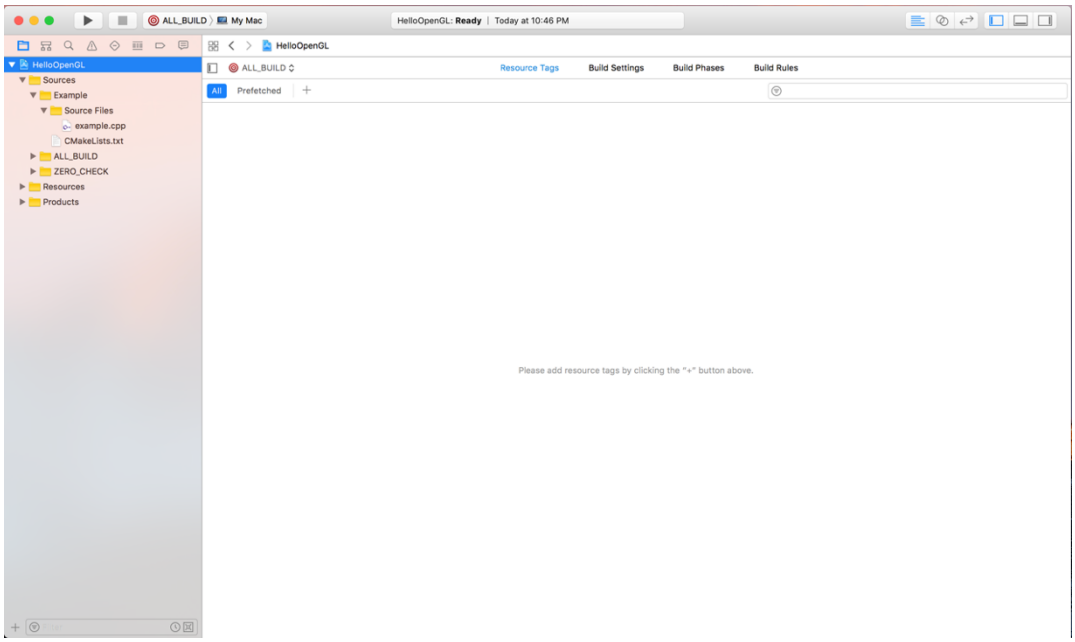
3.2 Build via the Xcode IDE

If you have installed Xcode and want to use Xcode as the development environment, you will need to specify a parameter when running cmake:

```
$mkdir build
$cd build
$cmake .. -G Xcode
```

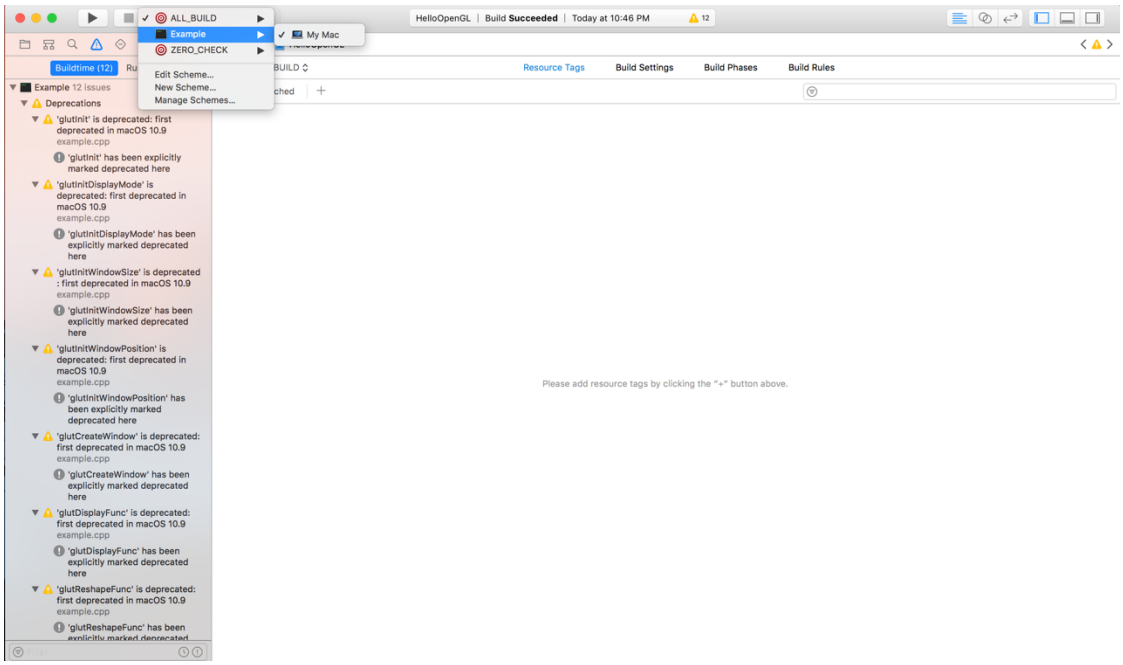
Now the Xcode project file called HelloOpenGL.xcodeproj is generated in the build directory. To open it, execute `open HelloOpenGL.xcodeproj` in the command line or double click it in the GUI.

Click the button  to build the project.

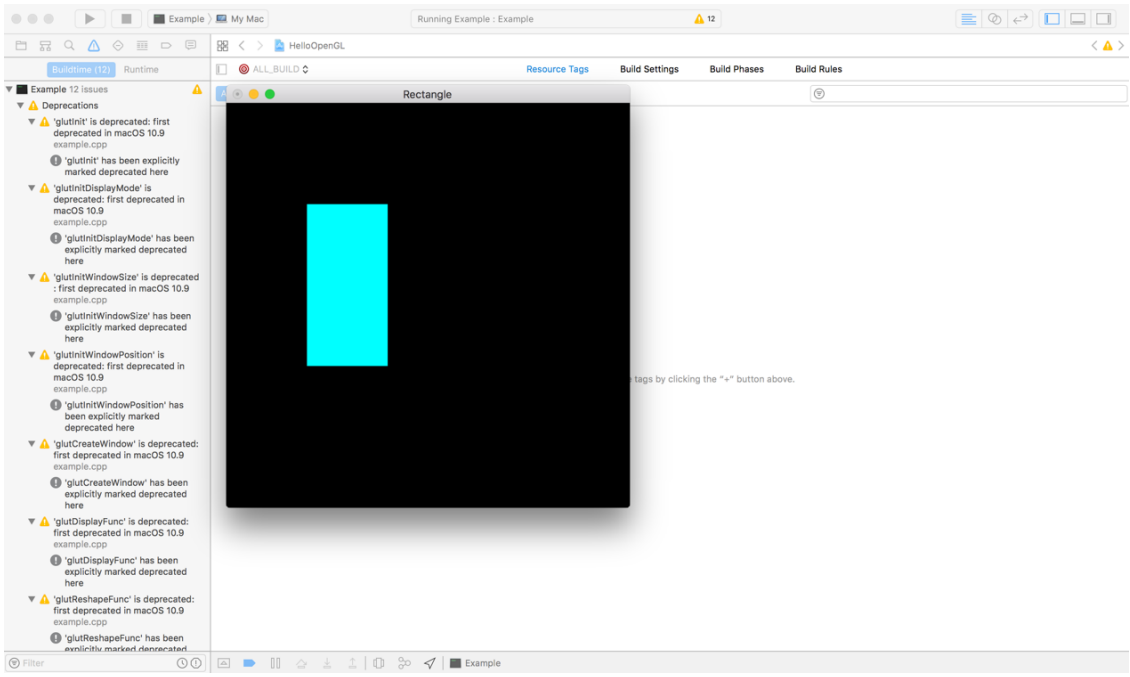


Switch the scheme to the executable named Example, and click button  to run it.

NYU Tandon School of Engineering
CS6533/CS4533
Zebin Xu
zebinxu@nyu.edu



Now you should be able to run and debug your OpenGL program on Xcode.



4. Compilation on Linux

For Linux user, you can use the same given CMakeLists.txt to generate Makefile, compile and run.

For Ubuntu, you need to install OpenGL, GLUT, and GLEW:

```
$sudo apt-get install freeglut3-dev
$sudo apt-get install libglew-dev
$sudo apt-get install libxmu-dev libxi-dev
```

You will need to install libxmu and libxi (the 3rd line) if you have the following error when running cmake:

```
CMake Error: The following variables are used in this project, but
they are set to NOTFOUND.
Please set them or make sure they are set and tested correctly in the
CMake files:
GLUT_Xi_LIBRARY (ADVANCED)
…
GLUT_Xmu_LIBRARY (ADVANCED)
…
```

5. Notes

Reading shader files

Shader files are literally plain-text files containing the shader source code. Hence, if you specify a **relative** path for the shader files in the C/C++ code, say the current directory (e.g., `program = InitShader("./vshader42.glsl", "./fshader42.glsl");`), then in order to read the shader files you should run your program from the **same path** as the files, i.e., the **execution path** is the root of the specified relative path.

For command line user:
Say you are in the root of the following directory hierarchy:

```
.
├── Angel-yjc.h
├── CMakeLists.txt
├── CheckError.h
├── InitShader.cpp
├── build
│   ├── CMakeCache.txt
│   ├── CMakeFiles
│   ├── Makefile
│   ├── RotateCubeNew
│   └── cmake_install.cmake
├── fshader42.glsl
├── mat-yjc-new.h
├── rotate-cube-new.cpp
├── vec.h
└── vshader42.glsl
```

The executable **`RotateCubeNew`** is in the build directory and shader files are in the root directory. Stay in the root directory and run:

```
$./build/RotateCubeNew
```

Alternatively, you can modify the relative path in the code:

```
program = InitShader("./../vshader42.glsl", "./../fshader42.glsl");
```

Then you can run directly in the build directory:

```
$./RotateCubeNew
```

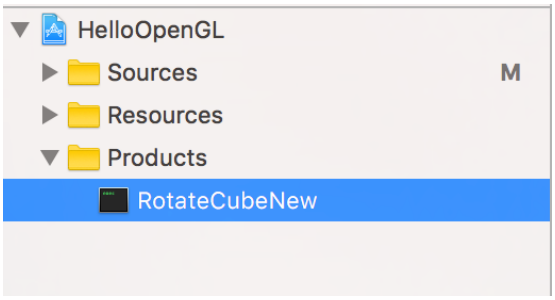
For Xcode user:

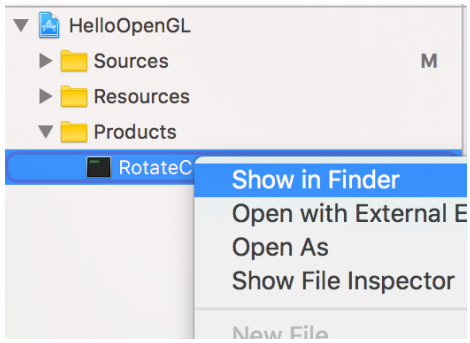
Say you are in the following directory hierarchy:

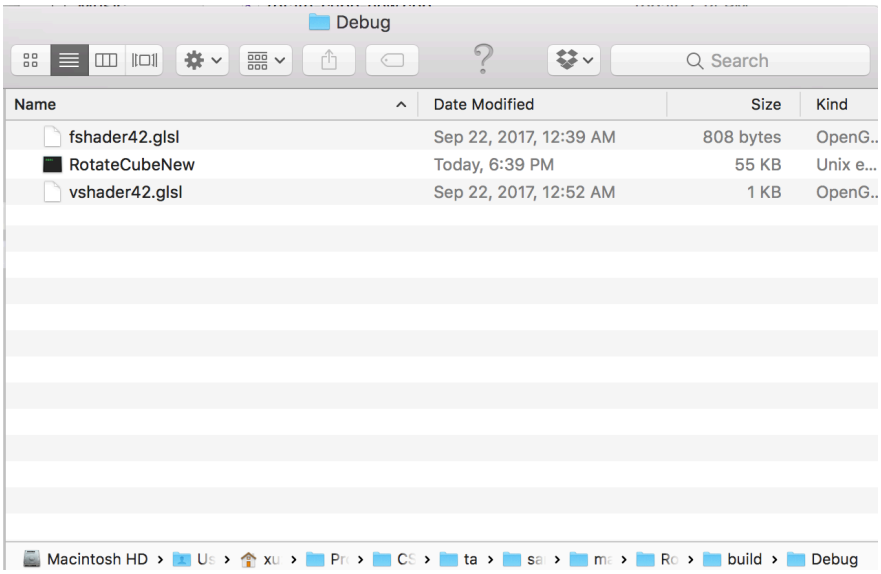
```
.
├── Angel-yjc.h
├── CMakeLists.txt
├── CheckError.h
├── InitShader.cpp
├── build
│   ├── CMakeCache.txt
│   ├── CMakeFiles
│   │   ├── 3.7.2
│   │   ├── CMakeOutput.log
│   │   ├── CMakeTmp
│   │   ├── TargetDirectories.txt
│   │   ├── cmake.check_cache
│   │   ├── feature_tests.bin
│   │   ├── feature_tests.c
│   │   └── feature_tests.cxx
│   ├── CMakeScripts
│   │   ├── ALL_BUILD_cmakeRulesBuildPhase.makeDebug
│   │   ├── ALL_BUILD_cmakeRulesBuildPhase.makeMinSizeRel
│   │   ├── ALL_BUILD_cmakeRulesBuildPhase.makeRelWithDebInfo
│   │   ├── ALL_BUILD_cmakeRulesBuildPhase.makeRelease
│   │   ├── ReRunCMake.make
│   │   ├── ZERO_CHECK_cmakeRulesBuildPhase.makeDebug
│   │   ├── ZERO_CHECK_cmakeRulesBuildPhase.makeMinSizeRel
│   │   ├── ZERO_CHECK_cmakeRulesBuildPhase.makeRelWithDebInfo
│   │   └── ZERO_CHECK_cmakeRulesBuildPhase.makeRelease
│   ├── Debug
│   │   └── RotateCubeNew
│   ├── HelloOpenGL.build
│   │   └── Debug
│   ├── HelloOpenGL.xcodeproj
│   │   ├── project.pbxproj
│   │   ├── project.xcworkspace
│   │   └── xcuserdata
│   └── cmake_install.cmake
├── fshader42.glsl
├── mat-yjc-new.h
├── rotate-cube-new.cpp
├── vec.h
└── vshader42.glsl
```

The default execution path is `./build/Debug/`, which is generated after running build.

You can also find it by right-clicking the executable `RotateCubeNew` in Xcode and select "Show in Finder" to browse to the execution path. You need to **copy and paste the shader files to this path** in order to run. Note that if you modify the shader files located in the root directory, you should update the corresponding shader files in the execution path (which I admit is inconvenient).







Alternatively, to avoid copying and pasting, you can modify the relative path:

```
program = InitShader("./../../vshader42.glsl", "./../../fshader42.glsl");
```

They refer to the shader files located in the root of the hierarchy so that your program running in its execution path can correctly read them.

More Information

CMake tutorial: https://cmake.org/cmake-tutorial/
OpenGL 3.2 core profile spec:
https://www.khronos.org/registry/OpenGL/specs/gl/glspec32.core.pdf