

Hans Dulimarta

- [RSS](#)

Search
Navigate... 

- [Blog](#)
- [Archives](#)
- [C/C++](#)
- [Java](#)
- [Android](#)

Using GLEW, GLFW, and GLM

What Are These Libraries?

- [GLEW](#) is an OpenGL library for handling OpenGL extensions. Users interested in diving into its source code can use git to clone [its repository](#).
- [GLUT](#) has been the defacto standard library for many OpenGL programmers, but it has not been actively maintained for decades. The last update to [GLUT 3.x specification](#) was dated in 1996. The most logical alternative to GLUT is [FreeGLUT](#), however installation on Windows machines requires manual compile steps. I'm trying to look for a simpler alternative for my students.

After some research, I found [GLFW](#). Windows users can download either the [32-bit](#) or [64-bit](#) version of the library.

- [GLM](#) is a header-only math library designed for OpenGL programmers. It provides functions and classes for dealing with vectors, matrices, quaternions, and related operations.

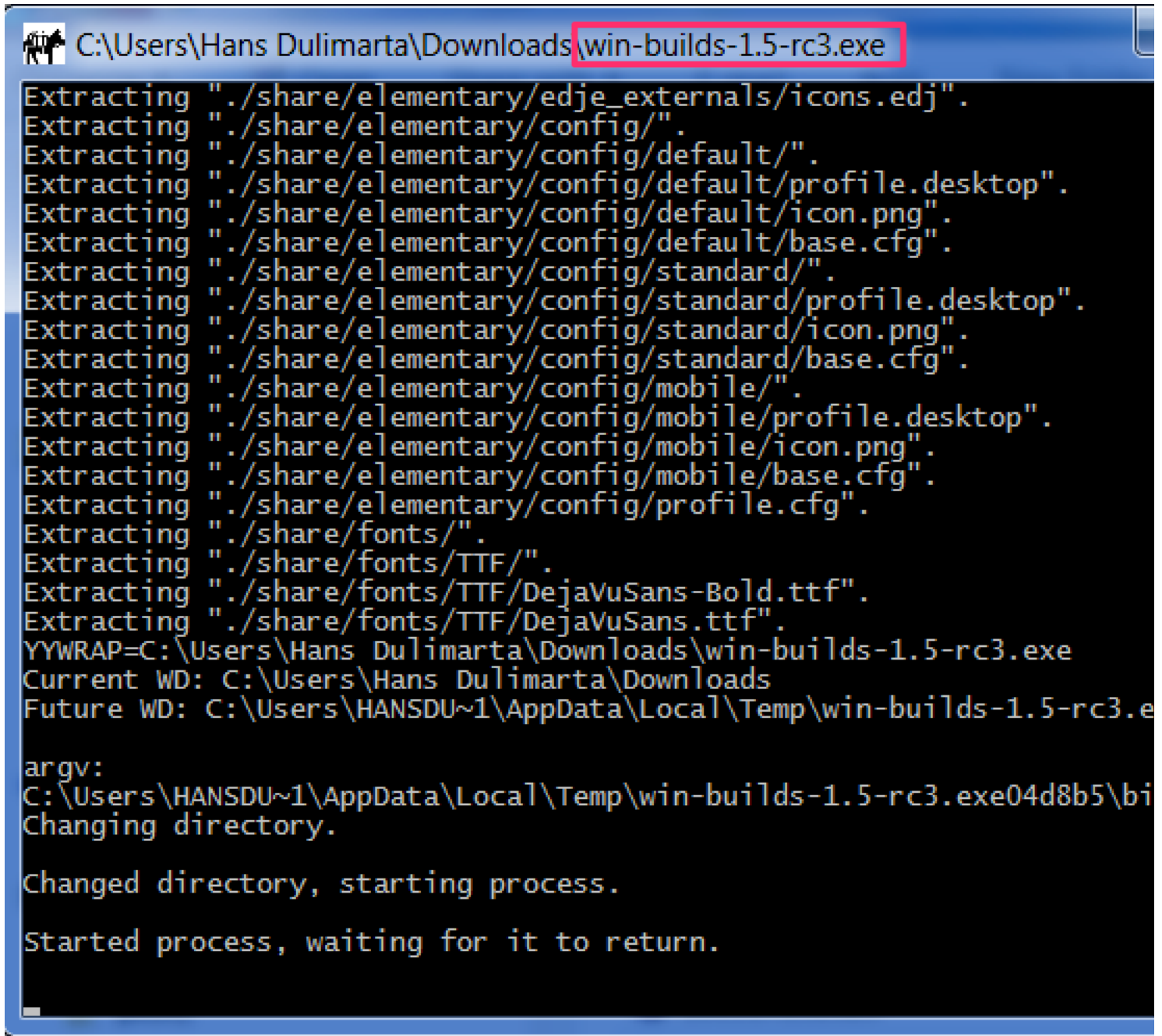
This document shows how to use these libraries, mainly on Windows machines. At the end, I also show instructions for using these libraries with CLion. Using these libraries on OSX or Linux is relatively easier; specific instructions for [Linux](#) and [OSX](#) are provided at the end of this document.

The main steps are described below.

Step1: Install MinGW Using Win-Builds

If you have a previous version of MinGW, it is strongly recommended that you uninstall it first.

Windows users must install MinGW (32-bit or 64-bit) for CLion to compile C/C++ programs. To setup MinGW, first download [win-build](#) package manager. Linux/OSX users most likely already have g++ installed.



```

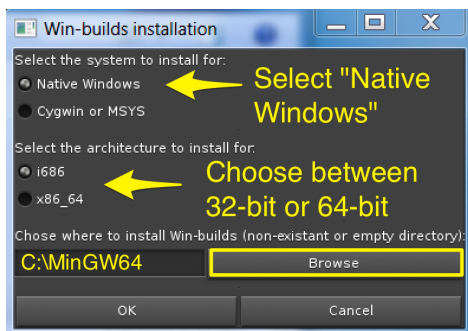
C:\Users\Hans Dulimarta\Downloads\win-builds-1.5-rc3.exe
Extracting "./share/elementary/edje_external/icons.edj".
Extracting "./share/elementary/config/".
Extracting "./share/elementary/config/default/".
Extracting "./share/elementary/config/default/profile.desktop".
Extracting "./share/elementary/config/default/icon.png".
Extracting "./share/elementary/config/default/base.cfg".
Extracting "./share/elementary/config/standard/".
Extracting "./share/elementary/config/standard/profile.desktop".
Extracting "./share/elementary/config/standard/icon.png".
Extracting "./share/elementary/config/standard/base.cfg".
Extracting "./share/elementary/config/mobile/".
Extracting "./share/elementary/config/mobile/profile.desktop".
Extracting "./share/elementary/config/mobile/icon.png".
Extracting "./share/elementary/config/mobile/base.cfg".
Extracting "./share/elementary/config/profile.cfg".
Extracting "./share/fonts/".
Extracting "./share/fonts/TTF/".
Extracting "./share/fonts/TTF/DejaVuSans-Bold.ttf".
Extracting "./share/fonts/TTF/DejaVuSans.ttf".
YYWRAP=C:\Users\Hans Dulimarta\Downloads\win-builds-1.5-rc3.exe
Current WD: C:\Users\Hans Dulimarta\Downloads
Future WD: C:\Users\HANSDU~1\AppData\Local\Temp\win-builds-1.5-rc3.e
argv:
C:\Users\HANSDU~1\AppData\Local\Temp\win-builds-1.5-rc3.exe04d8b5\bi
Changing directory.

Changed directory, starting process.

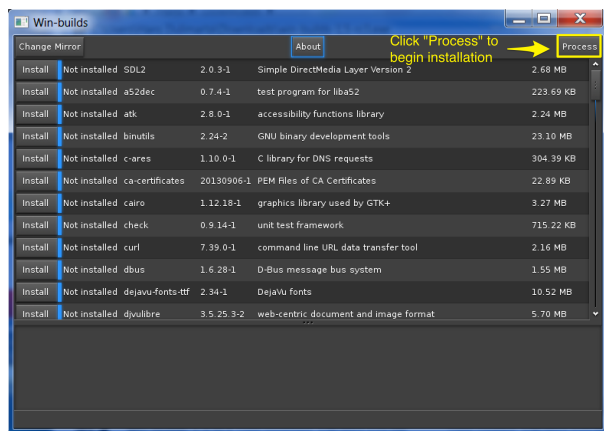
Started process, waiting for it to return.

```

On the next screen, select the installation type (Native Windows), select the architecture (32- or 64-bit) to match your Windows. Click the **Browse** button to select the top directory of your MinGW installation (the following screenshot assumes a 64-bit architecture). Click "OK" to continue with installation.



On the next dialog window, click the "Process" button on the upper right corner of the window.



Edit the PATH environment variable to include C:\MinGW64\bin (or whatever name you chose during installation).

Step 2: Verify gcc and g++ are installed

From the command prompt type

```
gcc
g++
```

If the response is something other than “Command not found”, your compilers are installed correctly.

Step 3: Install GLFW

Windows users can download either [32-bit](#) or [64-bit](#) binary. Unzip the file **directly under C:\ drive**, (i.e. **do not** extract the files to C:\MyCourseWork, for instance), then rename the top installation directory to C:\GLFW. When the installation is complete, you will find the following subdirectories (among others):

- C:/GLFW/include/
- C:/GLFW/lib-mingw

OSX/Linux users should first use cmake to compile GLFW from source. **After downloading GLFW source code**, do the following:

```
1 cd <top-dir-of-your-glfw-installation>
2 mkdir build
3 cd build
4 cmake ..
5 make
6 make install
```

If everything went smoothly, GLFW header files will be installed at /usr/local/include and its static library will be installed at /usr/local/lib.

Test your GLFW installation by typing the following program:

```
1 #undef GLFW_DLL
2 #include <GLFW/glfw3.h>
3 #include <cstdlib>
4 #include <iostream>
5 using namespace std;
6
7 int main() {
8     if (!glfwInit()) {
9         cerr << "Can't initialize GLFW" << endl;
10        exit (EXIT_FAILURE);
11    }
12
13    GLFWwindow *win;
14    win = glfwCreateWindow (450, 300, "Hello", NULL, NULL);
15    glfwMakeContextCurrent(win);
16    while (!glfwWindowShouldClose(win)) {
17        glfwWaitEvents();
18    }
19    glfwDestroyWindow(win);
20    glfwTerminate();
21    return 0;
22 }
```

The `#undef GLFW_DLL` directive will compile the non-DLL version of the GLFW functions. Assuming the above code is saved as `main.cpp`, compile it with the following command line:

```
g++ -std=c++11 -IC:/GLFW/include -LC:/GLFW/lib-mingw main.cpp -o yay -lglfw3 -lopengl32
```

If the code is compiled using the DLL version of GLFW functions, the compiler options are:

```
g++ -std=c++11 -IC:/GLFW/include -LC:/GLFW/lib-mingw main.cpp -o yay C:/GLFW/lib-mingw/glfw3dll.a -lglfw3 -lopengl32 -lglu32 -lgdi32
```

The additional flags for the above compile are:

- -IC:/GLFW/include the include directory for GLFW header files
- -LC:/GLFW/lib-mingw and -LC:/GLFW/lib-mingw\glfw3dll.a the link directory and library archive for GLFW (DLL) functions

When everything goes well, you will find a new file `yay.exe`, but running the program will show an error message (missing `glfw3.dll`). Copy the missing file from `c:/GLFW/lib-mingw` to the current directory. After copying the `.dll`, you can now run `yay.exe`

yay

Step 4: Install GLEW Windows Binaries

Download and unzip the Windows binary of [GLEW](#). *Follow the same step as installing GLFW above (extract to C:\ drive and rename)*. The online installation instructions direct you to copy certain files to certain location of your windows, but **I skipped those instructions**.

After the installation you will find the following directories:

- C:\GLEW\include
- C:\GLEW\bin\Release\Win32
- C:\GLEW\bin\Release\x64
- C:\GLEW\lib\Release\Win32
- C:\GLEW\lib\Release\x64

Under the `lib\Release\...\` you'll find two library files `glew32.lib` (for non-static linking) and `glew32s.lib` (for static linking). We will use the *static* library

```
1 #define GLEW_STATIC
2 #include <GLEW/glew.h>
3 #undef GLFW_DLL
4 #include <GLFW/glfw3.h>
5 #include <cstdlib>
6 #include <iostream>
7 using namespace std;
8
9 int main() {
10     if (!glfwInit()) {
11         cerr << "Can't initialize GLFW" << endl;
12         exit (EXIT_FAILURE);
13     }
14
15     GLFWwindow *win;
16     win = glfwCreateWindow (450, 300, "Hello", NULL, NULL);
17     glfwMakeContextCurrent(win);
18     glewInit();
19     while (!glfwWindowShouldClose(win)) {
20         glfwWaitEvents();
21     }
22     glfwDestroyWindow(win);
23     glfwTerminate();
24     return 0;
25 }
```

Compile the above code using the following:

```
g++ -std=c++11 -IC:/GLFW/include -IC:/GLEW/include -LC:/GLFW/lib-mingw main.cpp -LC:/GLEW/lib/Release/x64 -o yay C:/GLFW/lib-mingw/glfw3dll.a
```

The two additional flags here are:

- -IC:/GLEW/include (the include directory for the header file `glew.h`)
- -LC:/GLEW/lib/Release/x64 (for the 64-bit version of the static library)

Step 5: Install GLM

Download and unzip [GLM](#) or clone it from [GitHub](#). This is a header only C++ library, no further installations steps required. Test your GLM installation by typing the following program:

```
1 #include <glm/vec3.hpp>
2 #include <glm/gtx/io.hpp>
3 #include <iostream>
4
5 using namespace std;
6 using glm::vec3;
7
8 int main () {
9     vec3 i_am_zero;
10
11     cout << i_am_zero << endl;
12     return 0;
13 }
```

Assuming GLM is installed at `c:/GLM` the above code is `glmtest.cpp`, compile the program as follows:

```
g++ -std=c++11 -IC:/GLM glmtest.cpp -o runme
runme
```

Expect to see output of a zero vector.

After the three libraries (GLEW, GLFW, and GLM) are installed, you can now compile your OpenGL program from the command line:

```

1 g++ -std=c++11 \
2 -IC:/GLFW/include -IC:/GLEW/include -IC:/GLM \
3 -LC:/GLFW/lib-mingw -LC:/GLEW/lib/Release/x64 \
4 main.cpp __place_more_cpp_file_here__
5 -o yay C:/GLFW/lib-mingw/glfw3dll.a \
6 -lglfw32s -lglfw3 -lopengl32 -lglu32 -lgdi32

```

If you see a warning “corrupt .drectve at end of def file”, ignore it.

The above command is split into several lines for clarity. If you prefer to use an IDE or Linux “make” tool proceed to the next step.

Step 6: Prepare CMakeLists.txt [for CLion]

To speedup development time, I recommend you use [cmake](#), a multi-platform development tool for setting up project for various IDEs. The following list of IDEs is just a small fraction of what cmake supports:

- Visual Studio
- Codeblock
- Eclipse
- Xcode
- Unix make
- KDevelop

Use `cmake --help` to see the complete list and use `cmake -G` to generate files for a specific project. For instance to generate an Xcode project:

```

1 mkdir build
2 cd build
3 cmake -G"Xcode" __path_to_CMakeList.txt__

```

Because cmake generates lots of files, it is recommended that we create a separate directory (build in the above example) when running it.

To use cmake, we must create/edit a text file CMakeList.txt. If you plan to use CLion (the next step), it is not necessary to download cmake yourself because CLion comes bundled with cmake.

In the directory where you save your .cpp and .h files, create/edit CMakeLists.txt and add the following:

- `include_directories("C:/GLFW/include" "C:/GLEW/include" "C:/GLM")`
- `link_directories("C:/GLFW/lib-mingw" "C:/GLEW/lib/Release/x64")`
- `target_link_libraries(..... glfw3 glew32s glu32 opengl32)`

See the following example:

```

1 cmake_minimum_required(VERSION 2.8.4)
2 project(sample)
3 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")
4
5 if (NOT WIN32)
6 set(CMAKE_MODULE_PATH /usr/local/lib/cmake /usr/local/lib/x86_64-linux-gnu/cmake)
7 set(CMAKE_PREFIX_PATH /usr/local/lib/cmake/glfw)
8 endif (NOT WIN32)
9
10 find_package (PkgConfig REQUIRED)
11 find_package (OpenGL REQUIRED)
12 if (WIN32)
13 include_directories("C:/GLFW/include" "C:/glm" "C:/glew/include")
14 link_directories("C:/glew/lib/Release/x64" "C:/GLFW/lib-mingw" )
15 else (WIN32)
16 find_package (glfw3 REQUIRED)
17 find_package (GLM REQUIRED)
18 find_package (GLEW REQUIRED STATIC)
19 endif (WIN32)
20 if (APPLE)
21 include_directories(/usr/local/include)
22 find_library(COCOA_LIBRARY Cocoa REQUIRED)
23 find_library(IOKIT_LIBRARY IOKit REQUIRED)
24 find_library(COREVID_LIBRARY CoreVideo REQUIRED)
25 endif (APPLE)
26
27 set(SOURCE_FILES main.cpp __add_other_cpp_files_here__)
28
29 ## "sample" is the name of the executable
30 add_executable(sample ${SOURCE_FILES})
31 target_link_libraries (sample
32     ${GLFW3_LIBRARY}
33     ${OPENGL_LIBRARIES}
34     ${GLEW_LIBRARY}
35     ${COCOA_LIBRARY} ${COREVID_LIBRARY} ${IOKIT_LIBRARY})
36 if (WIN32)
37 target_link_libraries (sample
38     ${OPENGL_LIBRARIES} glfw3 glew32s glu32 opengl32)
39 endif (WIN32)
40
41 if (UNIX)
42 target_link_libraries (sample
43     ${OPENGL_LIBRARIES}
44     ${GLFW3_LIBRARY}
45     ${GLEW_LIBRARY}

```

```

46         Xxf86vm pthread Xrandr Xinerama Xi Xcursor)
47 endif (UNIX)

```

Place the above CMakeLists.txt in the same folder as your .cpp files and that folder can now be *imported* as a CLion project. To test whether the above file is configured correctly, try to compile the code from the command line:

```

1 cd __to_the_directory_where_you_save_CMakeLists.txt__
2 mkdir mybuild # create a child directory
3 cd mybuild    # change into that directory
4 cmake ..      # CMakeLists.txt is in the parent dir
5 make          # compile
6 sample       # run sample.exe

```

To run the executable on Linux, you have to type: ./sample.

For subsequent compiles, you just need to type make.

```

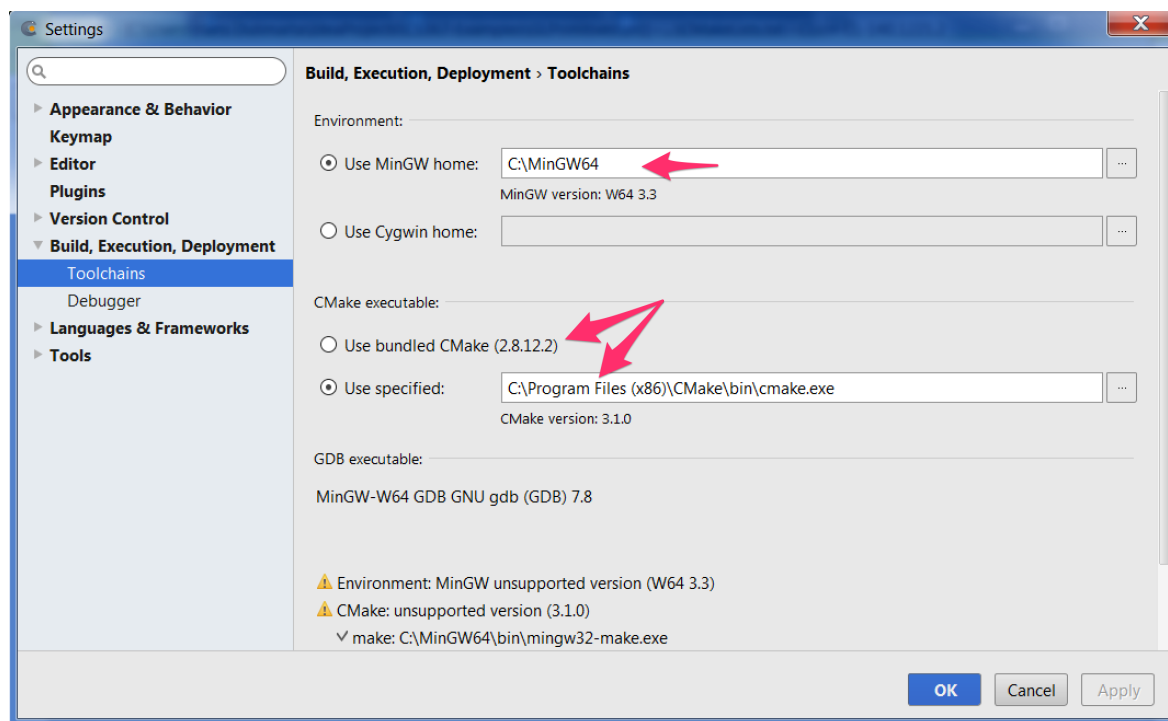
1 edit-your-code-again
2 make # compile again
3 sample

```

Step 7: CLion Setup

Once the three libraries (GLEW, GLFW, and GLM) are installed, you should be able to compile your program using CLion.

Download and install [CLion](#). During the setup, select the top directory of your MinGW installation and select your CMake executable.



Visit [Jetbrains E-store](#) to apply for academic license.

Once you have a working copy of CLion, **import** your project into CLion (be sure the CMakeLists.txt is configured to match your personal settings).

Linux Users

Install mesa-common-dev, git-core, and the libraries

```

1 sudo apt-get install cmake git-core
2 sudo apt-get install mesa-common-dev xorg-dev libglew-dev libglu1-mesa-dev
3
4 cd __to_installation_dir_of_these_packages
5 git clone https://github.com/glfw/glfw
6 git clone https://github.com/g-truc/glm

```

On Linux, GLFW depends on xorg-dev and libglu1-mesa-dev packages.

Build GLFW

```

1 cd __to_your GLFW_local_clone__
2 mkdir build
3 cd build
4 cmake ..
5 make
6 sudo make install # install GLFW headers and library

```

After a successful compile and install you'll find the following directories:

- /usr/local/include/GLFW: the location of GLFW header files
- /usr/local/lib/cmake/glfw: GLFW3 cmake config files
- /usr/local/lib/libglfw.so.*: GLFW3 shared library

Build GLM

```

1 cd __to_your_GLM_local_clone__
2 mkdir build
3 cd build
4 cmake ..
5 sudo make install # install GLM headers

```

The installation will create:

- /usr/local/include/glm: directory of GLM headers
- /usr/local/lib/x86_64-linux-gnu/cmake/FindGLM.cmake: CMake config file

The following CMakeLists.txt can be used to compile your code:

```

1 cmake_minimum_required(VERSION 2.8.4)
2 project(sample)
3 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")
4
5 set(CMAKE_MODULE_PATH /usr/local/lib/cmake /usr/local/lib/x86_64-linux-gnu/cmake)
6 set(CMAKE_PREFIX_PATH /usr/local/lib/cmake/glfw)
7
8 find_package (PkgConfig REQUIRED)
9 find_package (OpenGL REQUIRED)
10 find_package (glfw3 REQUIRED)
11 find_package (GLM REQUIRED)
12 find_package (GLEW REQUIRED STATIC)
13
14 set(SOURCE_FILES main.cpp __add_other_cpp_files_here__)
15 add_executable(sample ${SOURCE_FILES})
16 target_link_libraries (sample ${OPENGL_LIBRARIES} ${GLFW3_LIBRARY}
17                        ${GLEW_LIBRARY} Xxf86vm pthread Xrandr Xinerama Xi Xcursor)

```

In addition to OpenGL related libraries, Linux programs also depend on X11-related libraries: Xxf86vm, Xrandr, Xinerama, Xi, and Xcursor.

OSX Users

Software development tools and compilers should be already installed on your OSX when you have Xcode installed. In addition, OSX users are recommended to use [Homebrew](#) for package management. After Homebrew is installed you can then install cmake, pkg-config, and glew:

```

1 brew install cmake
2 brew install pkg-config
3 brew install glew

```

Clone and install GLFW and GLM using the instructions for Linux users above.

To compile using XCode:

```

1 cd __to_the_directory_where_you_save_CMakeLists.txt__
2 mkdir build # create a new directory for cmake generated files
3 cd build
4 cmake -GXcode .. # generate the project file, read the CMakeList.txt in the parent directory
5 xcodebuild # compile from the command line

```

When the compile succeeds, the binary executable will go to the Debug directory of the current build.

The last command will generate a new Xcode project file. Open that using Xcode 5 or Xcode 6. CMake generates three targets: ALL_BUILD, ZERO_CHECK, and the-executable-name. Switch to the last target to run the program from Xcode

[GLEW](#), [GLFW](#), [GLM](#), [MinGW](#), [OpenGL](#)

About Hans Dulimarta

Born in Indonesia. Received a B.Sc. in Computer Science from [Institut Teknologi Bandung](#) M.Sc. and Ph.D. from [Michigan State University](#) Faculty at the School of Computing and Information Systems since 2002.

