Goals and main task(s): Set up the programming environment for a program that can simulate a real-time FTC game. It should run on various platforms (Windows, Mac OSX, Linux, and Android) and have a graphics engine and physics engine.

Tasks: Download development tools

Reflections: I first set up Android Studio (our preferred programming environment for developing applications) and the Native Development Kit (NDK). The NDK allows us to program in native code (C/C++) instead of Java for Android apps. I decided to use native code because it runs faster than Java, and this efficiency is essential for graphics applications. After setting up the NDK, I installed Make and CMake. Using Make and CMake allows us to create scripts for each platform we plan to support that will build the same native code. This way, we can write native code that will run on any machine. After installing Make and CMake, I installed Apache Ant, which is a program that allows us to deploy and execute native code on an Android device.

Tasks: Import external libraries/dependencies

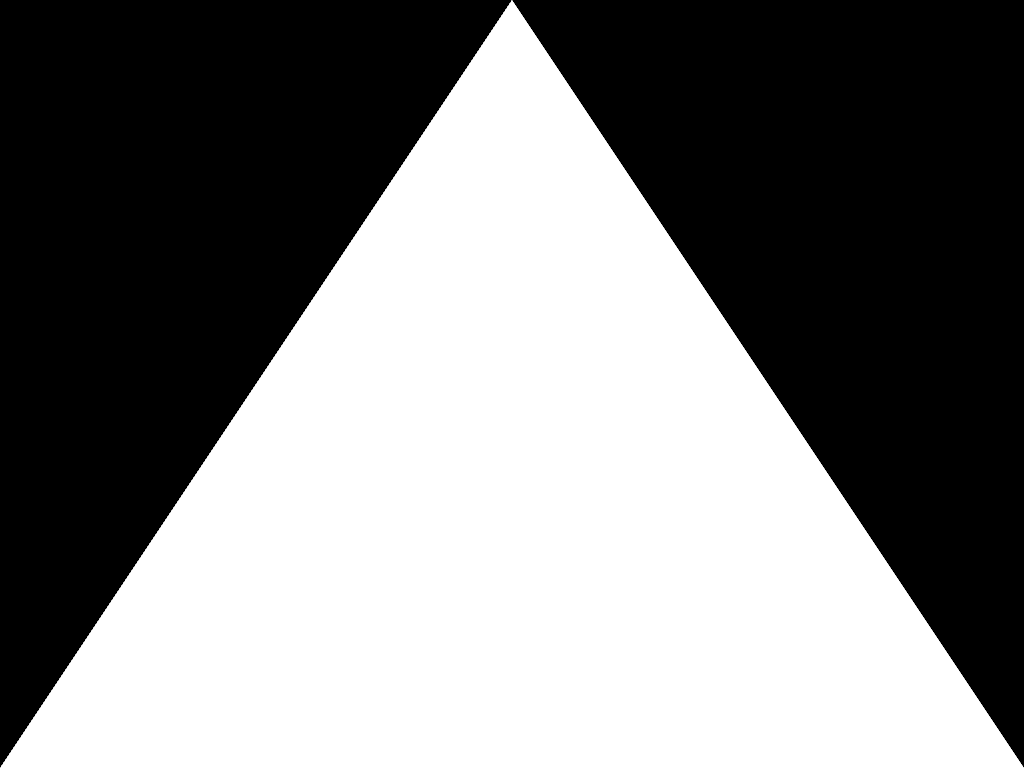
Reflections: I researched the libraries and dependencies I would need in advance because I wanted to set them up before developing the graphics engine. This task was very difficult because I had to compile binaries for every platform using different compilers. I first needed a graphics application programming interface (API). A graphics API allows you to communicate with a graphics card directly to draw graphics on the screen. Examples of graphics APIs include: Vulkan, OpenGL, Metal, and DirectX. I eventually settled on using a combination of the libraries GLEW and SDL. SDL handles windowing, input, and other game-essentials. GLEW is used for accessing OpenGL functions. I began by trying to compile both these libraries, but I ran into problems right off the bat. Compiling using the C++11 standard yielded several “code errors”, so I downgraded the C++ version to C++98. This fixed most of the problems, but I still had issues with macros in the code. I had to get rid of Android Neon support and remove all macros that had anything to do with it. After doing that, I was finally able to compile the libraries. I also imported Assimp, which is a library that loads graphics models and extracts important information from them. Lastly, I imported the common math library GLM, the user-interface helper library imgui, and several other miscellaneous libraries.

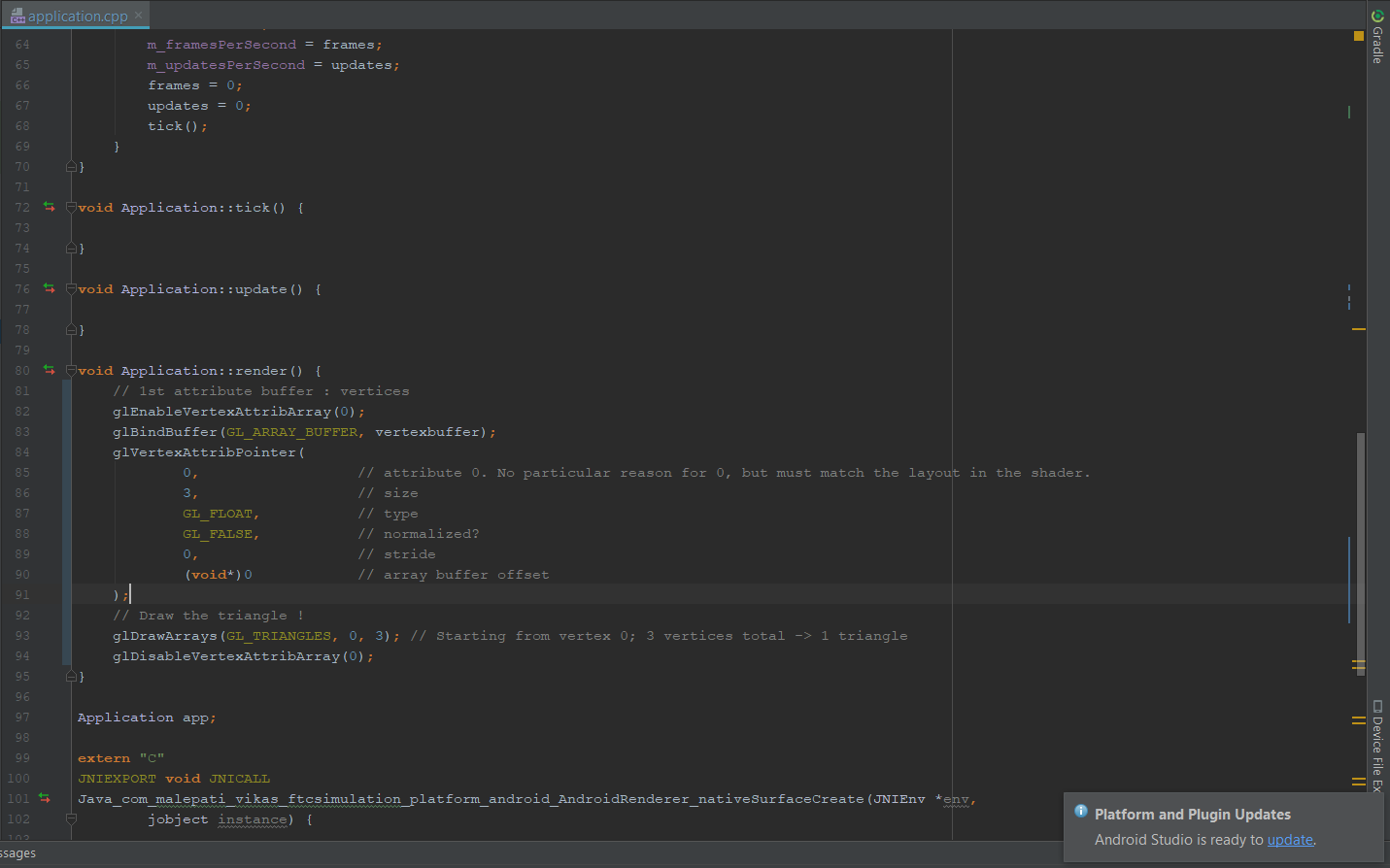
Tasks: Write test OpenGL code to draw a simple triangle

Reflections: After setting up most of my build environment, I wrote very basic OpenGL code to render a simple triangle (shown in the images below).

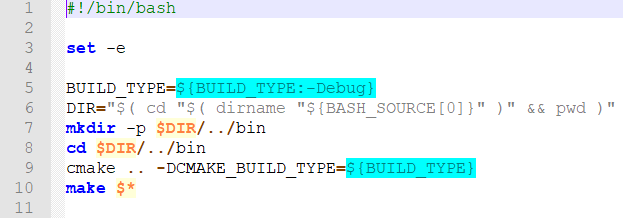
Tasks: Create scripts to build the code on each platform

Reflections: After writing all my code and setting up the environment, I then wrote scripts to run the code. I then tried building the code on several platforms, and surprisingly, the build only failed on Android. The issue with Android was that I was using a newer version of the software development kit (SDK version 26), which did not come with the Apache Ant tools I required. I had to downgrade the version of my Android SDK to version 23. This fixed the problems I had, and I was able to run the test code on all platforms.



Above: The final triangle drawn in the program

Below: The code that draws the triangle

Above: The script for building on Mac OSX and Linux

Below: The script for building on Android