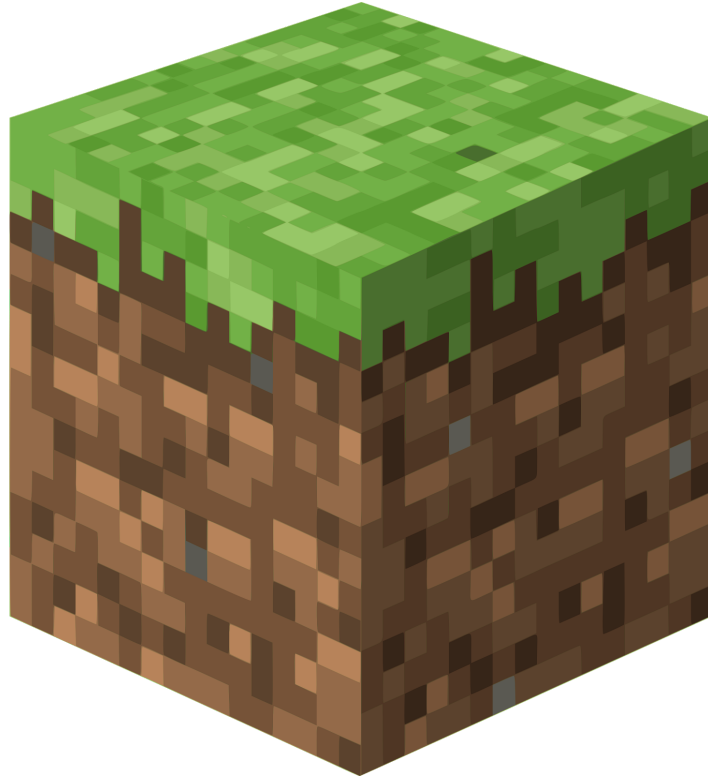


Minecraft in C++/OpenGL

By Jelle Booij & Wessel Mast



Motive

After getting into C++ for the OOP introductory assignment, we really felt like challenging ourselves to go the extra mile and attempt to build a game in C++ without the use of an engine. The core module assignment we then got had us pumped to just try and make it work.

After an initial brainstorming section, we really felt like creating a GTA clone, but in a 2.5D type fashion, like in GTA chinatown. We wanted to try and do this using SFML. Though, we ended up changing our minds about a week later, after discussing the amount of artwork we'd need to make (or copy) to make it work, and it would require one of us to spend a lot of time on just creating some form of map, while we both just really wanted to write some code. We needed something that wouldn't require a lot of artwork, but we still wanted to create this big, amazing world.

It's no surprise that Minecraft quickly came to mind.

Team Roles

Wessel:

- Generation code
- Documentation
- Activity Diagram

Jelle:

- Rendering code
- UI code
- Class Diagram

Code Architecture

Facade pattern:

A big part of our codebase is built upon using the facade pattern. A good example is the Chunk script, which contains a lot of non-trivial logic. Yet it gets used all the time in other scripts, like the the ChunkGenerator or Entity script. It makes it simple for a client to quickly instantiate complex code. It's a facade because it doesn't use polymorphism, it's just a class that handles a lot of non-trivial logic.

Constants:

A lot of the functions and variables in our codebase are constant, meaning if it's a variable, we are not allowed to change that variable (so it's no longer 'variable') and if it's a function, we are not allowed to change any outside variable within it. This makes sure we don't use a function or variable for something it's not supposed to be used for.

Macros:

Every now and then, we use macros in our scripts. An example of this would be 'IS_SELF' in the Entity class. Macros can be very useful when used correctly, to make code a lot more readable. We used it in this case as a more easy to comprehend function.

Templates:

We don't use a ton of these, but there are some templates hidden in the VertexBufferLayout class that make it possible to insert different types into the Push function.

Reusability

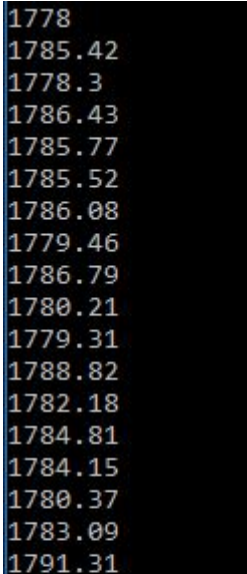
While building the code we made sure that it was reusable for different projects. First of all, the rendering and UI scripts is basically a voxel engine, which could be used for many more voxel projects, but even the generation scripts were created with reusability in mind. An example of this is the perlin noise. It could easily be taken into another project to play around with, without changing a single line of code. The other scripts are fairly connected to each other, but in another voxel project you could easily reuse them, because after changing some simple variables, or adding a different texture pack you can create an entirely different world with them. This makes our code almost 100% reusable if we'd want to create another voxel world using a custom engine.

Performance

Using OpenGL and C++, performance is our middle name! Running the full game (which is about 500mb in size) we get a solid 1780 fps with an NVIDIA GeForce GTX 1050 Ti video card. We made sure to optimise performance by using pointers and double checking any loops used in our project.

In terms of memory, we made sure any raw pointers made got properly deleted and emptied any vectors that still had elements after use.

We truly believe we can't make it much faster without multithreading, which we didn't really get to work in the end, but with this amazing performance that doesn't really matter.



```
1778
1785.42
1778.3
1786.43
1785.77
1785.52
1786.08
1779.46
1786.79
1780.21
1779.31
1788.82
1782.18
1784.81
1784.15
1780.37
1783.09
1791.31
```

Things to Add

These are a few things that would make this project feel even more complete!

- Better water (transparency, bobbing / sinking instead of walking)
- More interesting height mapping (transitions from mountains to fields)
- 3D perlin noise (caves and sky islands)
- Day/night cycle
- Mobs and Monsters
- Multithreading

We are very happy with the outcome though, and all of these are just optional.