Frank Collebrusco
Arsh Guntakal
CS 384G Sp25

CS 384G Final Project
Optimized Face Instancing C++ Minecraft Engine

Outline

This project aimed to build a fast, optimized Minecraft renderer and engine in C++. Minecraft is a notoriously poor performing game, particularly older Java versions. Building a procedural and editable voxel engine has some challenges, for one, how does one represent the mesh? How does one build it? In class we instance cubes. This is great because it uses instancing to be performant while rendering a VAO thousands of times. However, real Minecraft does not do this. As far as my research told me, Minecraft re-generates the mesh of a chunk on the CPU and re-buffers it to the GPU each time the chunk changes. When it generates the chunk, it only emits faces that are visible in the chunk: that is, exposed to air. We wanted to implement this method in C++ in order to try and build a performant renderer. Rebuffering the whole mesh on update is still expensive – and we'll talk about how we avoided this in the engineering section. Building this mesh is fairly simple: just loop over blocks and emit a face if it's exposed. This results in very fast draw times, allowing us to extend the render distance and stay real-time.

Engineering

The engine is built using my graphics library that I've been developing for a couple of years (flgl). It's built on OpenGL + GLFW. I'll be the first to admit that I spent too much time engineering the engine such that I ran out of time to utilize it to build the game. However, the foundation is strong and the renderer is very smooth.

As far as I know, Minecraft does not use instanced rendering. Each chunk mesh is unique, so they can't. We instance faces. There is one face VAO on the GPU, and the only thing re-buffered when a chunk updates is the instance buffer. It turns out that we can represent a face of a chunk with one 32-bit integer.

Chunks are 16 x 256 x 16. There are 6 possible orientations, and 4096 textures. So each face can be fully specified by:

- 4 bits for X
- 4 bits for Z
- 8 bits for Y
- 3 bits for orientation
- 12 bits for texture

This is 31 bits total. Thus, we can rebuffer chunks at a cost of 4 bytes per face! If we were rebuffering the whole mesh, each face would require 4 verts each with 8 floats (pos, uv, norm) plus 6 element indices. This is 152 bytes total. This is a 38x bandwidth improvement!