Tony Lu (tianyua1)
Julian Tutuncu-Macias (jtutuncu)

15-418: Final Project Proposal

Making Minecraft Fast

URL: rex-ludorum.github.io

SUMMARY: We are going to optimize Craft, a lightweight version of Minecraft written in C with OpenGL, by parallelizing computations in the multiplayer server environment to more efficiently process modifications to the world environment and ensure seamless, performant rendering for all players as the quantity of players scales.

BACKGROUND: Minecraft is a popular sandbox game in which the world is composed of cube-shaped blocks. Users can destroy and build parts of the world by breaking or placing blocks, respectively. While Minecraft is closed-source, Craft is a lightweight open-source version that supports less features. Its renderer already makes some optimizations, like partitioning the world into 32 by 32 block “chunks" and only rendering exposed faces. Our optimization will involve multithreading the processing of updates made to distinct chunks by multiple players interacting in a server environment. We will also port the renderer to Vulkan, a lower level graphics API that offers parallel tasking.

THE CHALLENGE: The state of the world environment is stored in an sqlite3 database as a delta of the initial world; currently, any modification to the environment made by a player connected to a Craft server is transmitted from the client to the server, where the world database is updated with the modification before re-rendering the entire chunk which a player inhabits. Rendering the scene itself presents opportunities for parallelism, since we will have to determine a scheme to split up the pixels on the screen. We must also find opportunities to exploit the spatial locality of the multiple players’ locations at any point in time. For example, on the server side, a single background thread is responsible for receiving world updates, and the computation of any additional updates is halted before the current modification has completed rendering. In the case where many players are inhabiting and modifying the world environment at the same time, this single thread presents a bottleneck for performant rendering, as there is much more computation than communication. We must find as many ways to parallelize the computation as efficiently as possible.

RESOURCES: We will use the existing implementation of Craft and add our code to it (https://github.com/fogleman/Craft). We will use machines with different levels of graphics computing power to collect statistics about our renderer.

GOALS AND DELIVERABLES: We plan to at least successfully implement a multithreaded client execution scheme for parallelized database writes. If time allows, we will adapt our scheme to be able to handle the case where players are in close proximity and are rendered by each other, and make our implementation performant on as constrained hardware resources as possible. Another additional goal is porting to Vulkan to achieve speedup over OpenGL. If our work progresses more slowly, we will aim to match the base implementation's frame rate as closely as possible. For our demo, we will have graphs featuring our speedup, and computers for people to play on to compare implementations.

PLATFORM CHOICE: We will use different machines to benchmark our implementation. Some will have dedicated GPUs and some will only have integrated graphics, like our laptops. Since the base Minecraft implementation is lightweight and is performant on our Macbook Pros with only integrated graphics, we will focus on systems with less resources to benchmark our implementation.

SCHEDULE:

November 4 - 10:
- Familiarize ourselves with current implementation of Craft, develop performance metrics & tests
- Study specifications of computers we are benchmarking on, especially integrated graphics

November 11 - 17:
- Familiarize ourselves with Vulkan
- Begin porting to Vulkan
- Draft multithreading schemes

November 18 - 24:
- Test multithreaded database writes

November 25 - December 1:
- Finalize prediction schemes

December 1 - 8:
- Finish poster and report

UPDATED CHECKPOINT SCHEDULE:

November 4 - 10:
- Familiarize ourselves with current implementation of Craft, develop performance metrics & tests (DONE)
- Study specifications of computers we are benchmarking on, especially integrated graphics (IN PROGRESS)

November 11 - 17:
- Familiarize ourselves with Vulkan (RELEGATED TO SIDE GOAL)
- Begin porting to Vulkan (RELEGATED TO SIDE GOAL)
- Draft multithreading schemes (IN PROGRESS)

November 18 - 24:

- Benchmark multithreading schemes
- Refine schemes to achieve > 1 speedup

November 25 - December 1:

- Finalize multithreading schemes
- Finish code

December 1 - 8:

- Collect final statistics
- Finish poster and report