NHTV Breda University of Applied Sciences

Personal project

Minecraft: Ray Traced

Marco Jonkers

Supervisor: David Hörchner

Abstract

This project attempts to implement a GPU ray tracer in the video game Minecraft, using CUDA. The ray tracing itself is done in CUDA. At first, I attempt to ray trace the vertex buffers given by Minecraft. Then, I implement my own data structures for increased performance.

April 5, 2017

Contents

Figure 1: In-game screenshot of Minecraft, without heads-up display (HUD) or viewmodel.

1 Minecraft

Minecraft is a 2011 first-person sandbox video game. Originally created by Markus "Notch" Persson, it is currently maintained by Mojang.

Minecraft comes in multiple editions, for various platforms. This paper focuses on the PC version of Minecraft, released for Windows, macOS, and Linux. The PC version is written in Java.

In Minecraft, the game world consists of a three-dimensional grid. The world is procedurally generated, using a mostly comprised of unit blocks, as shown in Figure 1.

2 Minecraft Forge

Minecraft Forge ("Forge") is a community created platform for developing and using modifications ("mods") for Minecraft.[1]

2.1 Setting up a mod development environment

The Mod Development Kit ("MDK") can be downloaded from the Forge website[2]. The MDK distribution includes ForgeGradle[3]. ForgeGradle is a plugin for the Gradle[4] build system. The Minecraft binaries are downloaded, and subsequently decompiled, deobfuscated,

The Mod Coder Pack[5] is a package which is used to decompile, change, and recompile Minecraft Java classes.

The Gradle tasks include:

1. Download Minecraft .jar files.
2. Decompile the Minecraft
3. Generate the Forge Minecraft binary

[1]Modification of Minecraft ("modding") is not officially supported by Mojang. For more information, visit https://account.mojang.com/terms

[2]http://files.minecraftforge.net/

[3]https://github.com/MinecraftForge/ForgeGradle

[4]https://gradle.org/

[5]http://www.modcoderpack.com/website/

2.2 Creating a mod

In general there are three approaches to creating a mod:

1. Build a mod on top of the Forge Minecraft code
2. Edit the Minecraft source directly
3. Change the Minecraft bytecode at runtime

For most mods, the first approach is sufficient. This also enables the developer to freely distribute their mod. Editing the Minecraft source directly means that the mod cannot be redistributed, because the original Minecraft code is copyrighted.

2.3 First attempt

I wanted to see if I could make my mod work using the first approach. Forge adds hooks to the game loop which I could use to intercept the rendering code.

2.4 Second attempt

Here I explain how the project is set up and the technologies involved.

The Minecraft Forge project allows me to do two things: Listen to specific events using pre-installed hooks Edit Minecraft bytecode before it is loaded Using Java Native Interface (JNI), I can call into C++ code from Java. The C++ code contains the CUDA kernel. Data is passed between OpenGL and CUDA using CUDA's graphics interop layer.

I am using the Minecraft Forge API. Forge is built on top of the Mod Coder Pack (MCP). MCP is a tool for decompiling Minecraft. A copy of the game is obtained by using the official installer from Mojang. Forge uses Gradle. During development, Java loads my classpath directly. In release mode, my class files would have to be compressed into an archive first. The path to my DLLs is passed to the virtual machine.

3 Minecraft

3.1 Minecraft Rendering System

There are four geometry groups:

Solid This is solid geometry.

Cutout Used for glass.

Mipped Cutout Identical to Cutout, but mipmapped.

Translucent Used for block which have partial transparency (alpha blending).

Every geometry group has its own vertex buffer.

4 Minecraft Forge

4.1 Coremods

5 Ray Tracing OpenGL Vertex Buffers

I explain something here that is found in Figure 10.

5.1 Viewport and Ray Origin

The viewport is passed from Java to C++ using ByteBuffers.

6 Challenges

The challenges of this project include:

Ray tracing performance Because of gameplay.

Acceleration structure rebuild speed Because of gameplay.

7 Static Geometry

Test for citing [1].
I am also citing [2].
I am also citing [3].

8 Benchmarks

9 Vertex Buffer Preprocessing

10 Future work

Future work is addressed here.

11 References

[1] John Amanatides, Andrew Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10, 1987.

[2] Paulo Ivson, Leonardo Duarte, and Waldemar Celes. Gpu-accelerated uniform grid construction for ray tracing dynamic scenes. *Master's thesis, Departamento de Informatica, Pontificia Universidade Catolica, Rio de Janeiro*, 2009.

[3] Erik Reinhard, Brian Smits, and Charles Hansen. Dynamic acceleration structures for interactive ray tracing. In *Rendering Techniques 2000*, pages 299–306. Springer, 2000.

12 List of Figures

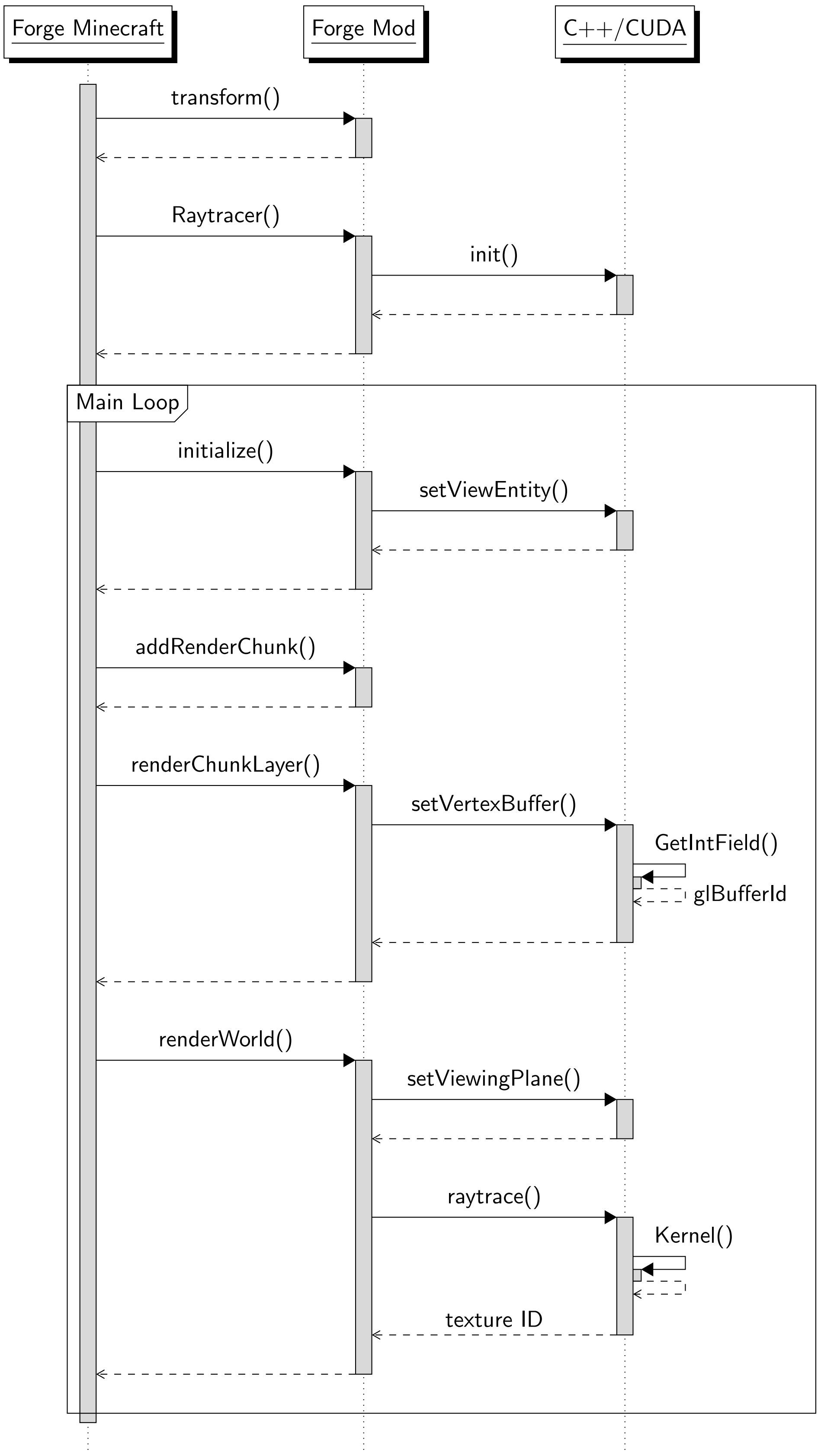


Figure 2: Sequence diagram showing interaction between the modules

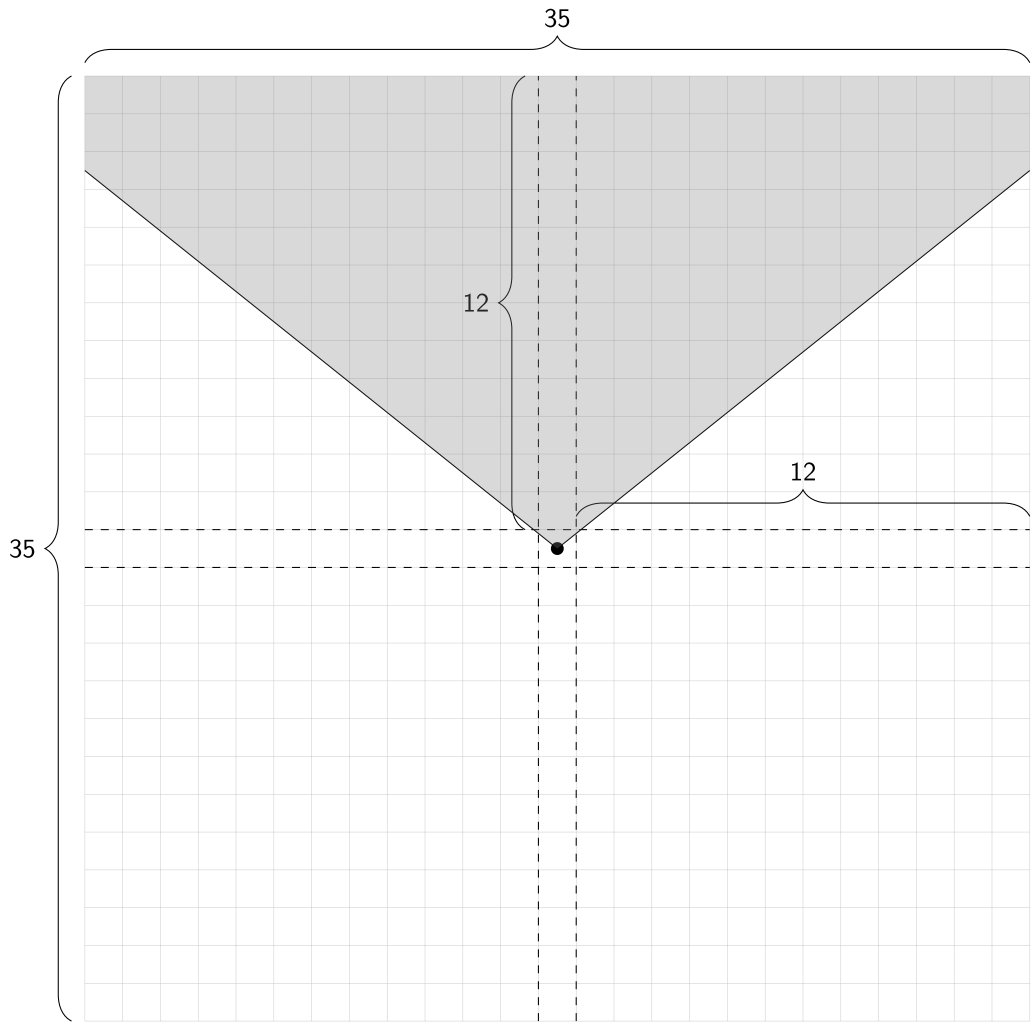


Figure 3: A top-down view of the vertex buffer array, assuming a render distance of 12. The player's view cone is shown facing north. Each cell contains 16 RenderChunks. When the player crosses a horizontal chunk boundary, every buffer in the grid is move once space.

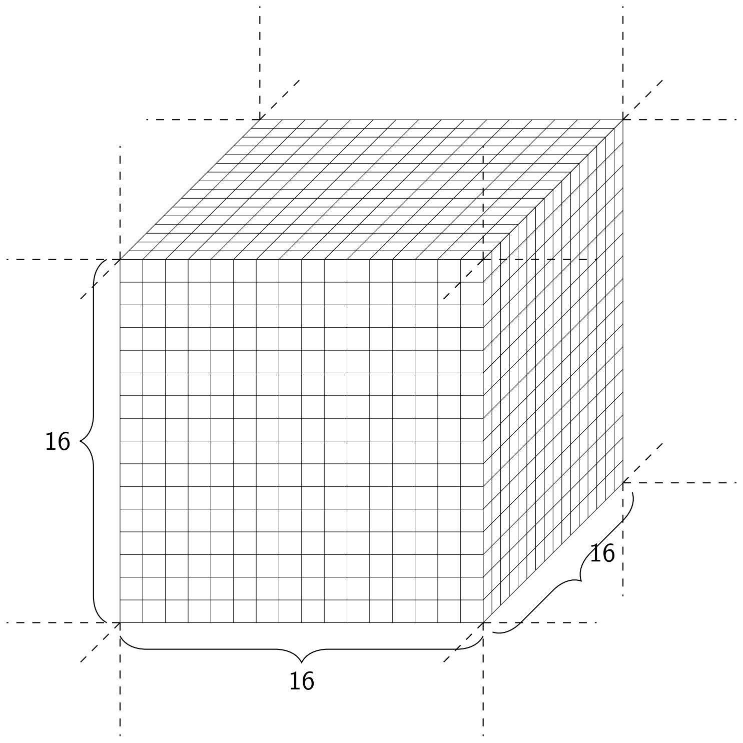
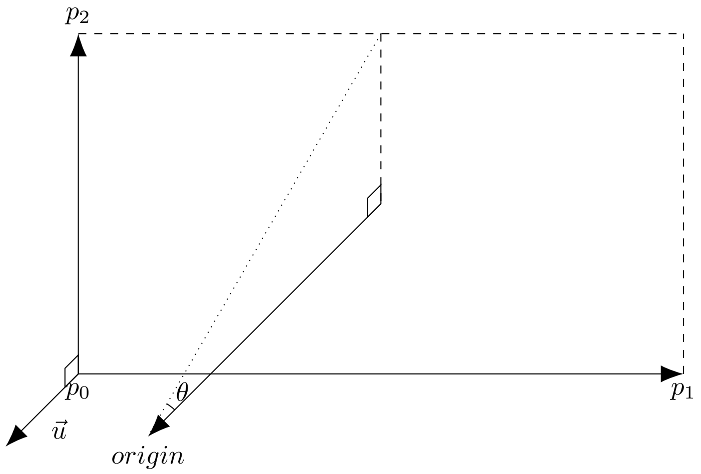


Figure 4: A single, completely filled RenderChunk.



$$\theta = \frac{vertical\ field\ of\ view}{2}$$
$$\vec{u} = \overrightarrow{p_0p_1} \times \overrightarrow{p_0p_2}$$
$$origin = \frac{p_1 + p_2}{2} + \frac{\vec{u}}{||\vec{u}||} \cdot \frac{\frac{||\overrightarrow{p_0p_2}||}{2}}{\tan(\theta)} \tag{1}$$

Figure 5: Viewport and ray origin calculations

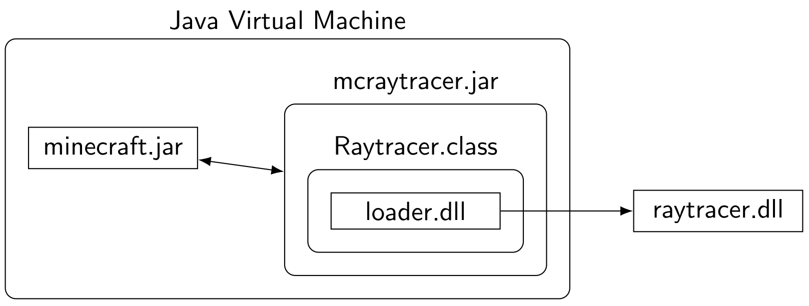


Figure 6: Interaction diagram of the different binaries.