UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Rendering a Minecraft Scene

Project Documentation

Grad Laurentiu-Calin
Group 30435

Computer Science
January, 2025

Contents

1 Subject Specification

The purpose of this project is to design a scene and render it using OpenGL (with the help of the GLEW, GLFW and GLM extensions). Being an open-choice-themed scene, there were multiple implementation "paths" to pick from. The first one was to search the internet for open-source models, download them, import them into OpenGL and render them onto the screen. This download-and-paste method appeared to be a menace to me because 1. the models I found on the internet were not designed to be used with our project core configuration and 2. it is almost impossible to find open-source models that fit seamlessly (coherently) into a scene. Consequently, I decided to look for texures and map them myself onto objects using Blender. I am a Blender rookie so I looked for the simplest model I can texture: the default cube. This way the idea to design a Minecraft-related scene was born.

For me, simulating the feeling of looking at Minecraft scenery is best described by the game-generated villages, so I decided to stick to this theme. The charm of a Minecraft village is a result of its composing structures: houses, churches, libraries, fountains, light poles made of torches, fences and wool etc. Creating some of these buildings became a core objective of the project.

I did not look forward to implementing a one to one copy of a Minecraft village (I am particularly amazed by Minecraft shaders), meaning that I opted for a non-default, open-source texture pack. This is where all the textures I used came from.

2 Scenario

2.1 Scene & Object Description

All the building/structures that are visible in the scene (except the land/floor/grass) are built from basic 2x2x2 blocks (or scaled versions, i.e. 4x2x2 for a door) created in Blender and exported as a pair (.obj, .mtl). This means each building is composed of between 20 up to 100 entities that are manually-placed level by level.

The scene is inspired from a default village generated in a taiga (spruce-tree-related) biome. The structures are made up by spruce logs and planks, cobblestone and stone bricks. When designing them I added a personal touch to their appearance (replaced cobblestone with planks or stone bricks, changed classic fences into morphed oak logs and so on).

The scene takes place at night in a taiga village.

2.2 Functionalities

1. **camera movement** - any place in the scene can be reached by moving the camera accordingly;

    - **W** - move forward
    - **A** - move left
    - **S** - move backward
    - **D** - move right
    - **mouse/touchpad** - rotate the camera around all axis

- **scroll/pinch** - zoom in/out

2. **automated tour of the scene** - press *1* to start an automated tour of the scene; during this process no input from outside (i.e. mouse, keyboard) is taken into account.

3. **toggle lighting** - press *L* to toggle the light poles in the scene.

3 Implementation Details

3.1 Functions & Algorithms

Functions & Algorithms: Explain the functions and algorithms used in your project.

3.1.1 Transparency

In order to make transparent elements (glass blocks, the windows in the door frame, the leaves), I had to sort them based on what direction I was facing. My implementation places all the objects to be rendered in a STL vector that is processed by the graphics pipeline. A thing to note is that the building blocks are added there without respect to the opacity. Consequently, after creating the scene, but before entering the game loop I had to separate them into transparent (glass, door, leaves) and non-transparent blocks (cobblestone, spruce and oak log, stairs etc). To do this I used a 2-pointer approach: i starts at the beginning and stops whenever it encounters a transparent block, j starts at the end and stops whenever it encounters a non-transparent block. When both of them stop I swap *i with *j. When i and j iterate over the entire array I stop. The final state of i is saved for further references.

When I finish the above-mentioned process I solve 2 problems: I moved all the transparent blocks references in a contiguous memory location and I saved a pointer to the first element there. From now on, if I do not add new objects in the scene I know that all the objects I must sort to account for transparency are placed in the STL vector starting at position i up to the end.

Finally, to account for transparency I must sort the transparent objects with respect to the direction the camera is facing. This means that in each iteration of the game loop I have to call a sort method on the ending part of the scene's object vector.

3.1.2 Possible Solutions

Solutions: Discuss the possible solutions you considered for implementing the project.

3.1.3 Motivation

Motivation: Explain the motivation behind your chosen solution.

3.2 Graphics Model

Graphics Model: Describe the graphical model used in your project.

3.3 Data Structures

Data Structures: List and explain the data structures used in your project.

3.4 Class Hierarchy

Class Hierarchy: Provide a diagram or description of the class hierarchy in your project.

4 Graphical User Interface Presentation

GUI Presentation: Present the graphical user interface of your project.

5 User Manual

User Manual: Provide instructions on how to use your project.

6 Conclusions

Conclusion:

- **Purpose Fulfillment:** Reflect on whether your project achieved its intended goals.
- **Potential Benefits:** Discuss how your solution can help others.
- **Practical Improvements:** Suggest ways in which the project could be improved.

7 Further Developments

Future Work: Outline potential future developments and enhancements.

8 References

References: List all references and sources you used in your project. Use a proper citation style.