PROJECT REPORT

LAB SECTION L

Yousif Alaa Lubbad

SUBMISSION DATE: 12/11/24

Overview

The objective of the project was to create a block-based game similar to Minecraft. The game is rendered through text in the terminal. The project includes a number of basic features that are found in Minecraft, it has player movement, placing and breaking blocks, random world generation, and saving/loading worlds.

Related Work

This game is based on Minecraft which has a version that is playable in the browser similar to what is produced from this project. In doing research for this project it was necessary to find a suitable way to render objects in the world. In the end transformation matrices were chosen over other options like ray tracing for their efficiency in drawing the world to the screen, this allowed the game to run consistently upwards of 100 frames per second. The ncurses library was incredibly useful for completing this project as it enables much more control over where things can be drawn to the screen.

Technical Description

(Numbers in parentheses are references to lines in the code)

Projection

The most important part of the project is the matrix conversions to draw things to the screen. (273) First the equation moves all the points so that the head of the player is at the center of the world. Then it rotates the world around the center so that the position that the camera is facing is now looking down the z-axis. After that the world is scaled to the screen size so that the numbers are easy to reference when drawing points on the screen. Then the points are divided by distance to the camera creating the effect of depth where things that are farther away are smaller on the screen.

Drawing Polygons

After the points are moved to the locations that they should appear on the screen, the polygons are drawn based on the points. The polygons are created with three integers as locations in the point array and then one integer to give a color for the polygon. Before they are drawn the polygons are checked to see if they are facing the player or not based on whether the points of the triangle are clockwise or counterclockwise to choose whether to draw each polygon. Then the polygons are ordered based on distance to the screen. When drawing to the screen the program goes through the list of polygons to draw and finds the smallest box that contains the triangle. Then it iterates through the points in the box coloring them if they are inside the triangle.

Storing Block Data

The block data is stored in a 3D array that is arranged as array[x][y][z]. This arrangement makes it easy to edit and store block data based on world coordinates. The blocks are stored as integers that are used to tell which block they are. The polygons are generated based on whether the block in a

specific direction of the tested block is air or not. If it is then the polygon list will be updated with new polygons to cover that face. (612)

Collision Detection

Collision detection was one of the hardest parts of the project to get working properly. It starts by setting the offsets of 12 points on the edge of the player relative to the world. (790) These points are set on the eight corners of the player collision box and halfway up the player to detect collisions that would pass through the top and bottom points. The first direction to be checked is the y-axis which is handled by checking either the top four or bottom four for collision with blocks depending on whether the player is moving up or down. (798) Then it deals with x followed by z in the same way. (833) (859) After that is done it then applies a downward force to the player if they are not on the ground so that they fall.

Placing/Breaking

The placing and breaking of blocks is the most unoptimized part of the program. (224-230) The program removes all the polygons from the list and regenerates the entire world every time a block is changed. This is completely overkill but was done for the sake of time as it was able to reuse a previous function and had a negligible impact on framerate. (612) The program still does some things that were more optimized for the task such as how it detects the block the player is currently pointed at. It does this by moving incrementally forward in the direction the player is looking until the coordinates are inside of a block, then it uses the coordinates of that point and the previous point to give the locations of the block to break or the space to place a new block.

Saving/Loading

Saving and loading blocks was likely the simplest part of the project as it was uncompressed. When saving, the program would generate a text file that would have the block type of every block in the game stored as a character. Those characters could then be read through the program when loading by using fgetch() and converting the character back into an integer.

Results

The project has many features totaling over 1000 lines of code that include:

- A 3D text based graphics engine that runs in the terminal.
- Collision detection between the player and the world
- Placing and breaking of blocks
- Functional movement system
- Menu system
- Save/Load system for worlds
- Steady 100+ fps gameplay

References

https://tldp.org/LDP/lpg/lpg.html Linux Programmer's Guide

http://www.codinglabs.net/article_world_view_projection_matrix.aspx 3D projection matrices

https://www.desmos.com/calculator For testing functions for graphics

https://stackoverflow.com/questions/53708076/what-is-the-proper-way-to-use-clock-gettime

Many other stack overflow pages that I no longer can find.