CMP505 – Advanced Procedural Methods Assessment: Ant farm Project

A screenshot of a video game

Description automatically generated

Figure 1: Starting view of demo

Introduction

For this assessment, a playable demo resembling Minecraft was made. While this was not the original intention of the project, it naturally came about due to the block placing nature of the original plan.

As seen in Figure 1, the two main focuses of the project can be seen: The fly jars on the right and the sand block in the middle. The sand block is made up of 1000 cubes in a 10 x 10 x 10 mini cube arrangement. Using controls listed below, the player can generate an ant, and change the view of the sand block to see what it’s doing. Seen on the right are five fly jars. These are transparent blocks, with five ant objects (pretending they’re flies) being generated at runtime. When an ant is generated, multiple randomly generated checks are done to determine the ants’ unique traits:

* dig speed: Range 5 to 9, divided by 10 if in sand block
* erratic nature: coinflip
* lifespan: Range 30 to 180, forever if in glass jar
* digging direction: Range -2 to 1, for each vector co-ordinate
* Cube face spawn: Range 1 to 6: Top, Bottom, Left, Right, Front, Back
  + An additional two random generations calculates the ant’s co-ordinates on that faces plane.

Most of these are self-explanatory, except for the erratic nature. Normally, an ant will continue its digging direction until it collides with one of its block’s faces. Erratic ants however will generate a new digging direction every second, five seconds if it’s a glass jar fly.

Controls

Due to time constraints and an injured right (dominant) hand, there are no mouse controls.

* WASD / Arrow keys: Standard forward/backward/side-to-side movement
* Q / E: Camera y-axis rotation, left / right
* U / Spacebar: Move up
* J / Left Shift: Move down
* P: Pauses ant/fly movement and lifespan countdown
* R: Resets demo to initial state
* 1 / Nm 1: Toggles the display of sand block to glass jar
* 2 / Nm 2: Toggles the display of sand
* 3 / Nm 3: Generates ant in main sand block
* 4 / Nm 4: Toggles the display of sand / eaten sand mini cubes
* 5 / Nm 5: Cycles through available blocks the player can place
* Enter: Places block in location of the hovering half-sized block in front of the player
* Backspace: Deletes block (if not original part of scene) at hovering block
* N: Triggers monochrome post-processing effect
* M: Triggers bloom post-processing effect
* Esc: Quits Demo

Initially, move up and down were jump and crouch respectively, but due to last minute bugs with collision detection they were changed to simple elevation changes. This also meant the removal of the gravity implementation.



Figure 2: Sand block with a few ants, eaten mini cubes

Birds flying over a building

Description automatically generated with medium confidence

Figure 3: Demonstrating custom block placement and glass jar functionality

One feature for this project is the ability to increase the number of flies in a fly jar. By attempting to create a fly jar block inside of another, five flies will be added to the original block without creating a secondary block. There is no limit to the number of flies added per block. One bug (or feature if you like) is that deleting the glass block doesn’t delete the ants inside it. Resetting the level with the r key does, however.

To expand on block placement and the user’s choices, they can place wooden plank, stone, wooden log, and glass jar blocks currently.

At the start of this project, the plan was to allow the player to place multiple sand blocks, or have each block have a denser mini cube ratio (20 to 100). Due to the inefficient code used to check each block for every ant collision (And also the vast number of models being rendered every frame) two or more sand blocks would begin to affect the framerate of the demo.

At the very start of the project, a 100 x 100 x 100 block cube was created as a test. While the program has been coded to handle a change in dimension size, this cube practically froze the program.

If the user wants more room to start placing blocks, there is a hole in the roof which allows access to outside.

Code Structure

The programs base was a build from one of the classes tutorials. By default, a sand block is 12 mini cubes per dimension, with the first and last cube in each dimension being glass. Due to transparency issues on certain axis’, this was changed to being a single block of glass that encompasses the sand block, typically invisible. All blocks are stored as vector4’s in a vector called cubes. Each vector4 contains co-ordinates as well as an ID for the blocks type. In the render section of game.cpp, if the block type is sand, a multi-layered loop system (three nested inside each other, one for each dimension) generates mini sand cubes based on the sand block’s location. Due to this design, only eaten sand cubes need to be stored. All the original blocks in the scene are created in the initialize section. Most sections of the scene are also created in loops.

Update

Update is one of the two main sections of my demo. Amongst the input controls, there is commented out code for the failed gravity system, along with jump and crouch functionality. Afterwards, “block hand” is updated.

This is the floating half block in front of the player. The blocks co-ordinates have been rounded; this snaps the block to the centre of each potential cube. Originally the idea was to mimic the players hand in Minecraft with a plan on implementing a ray cast function to place a block where the camera is aiming, but due to bugs, time constraints and an injury this was changed to what it is now.

Finally, there is the update ant’s section. For all living ants: their lifespan is shortened, their new position is calculated and applied via Ants.cpp and a check is applied to see if the ant is stuck. When an ant has the same co-ordinates for more than 0.1 seconds, a new digging direction is calculated. Additional checks were added beforehand that would kill the ant if it was stuck for more than two seconds, though this was mainly for debugging and realistically it won’t trigger anymore.

In Ants.cpp, when a position of an ant is being sent, it will check if each axis co-ordinate is out of bounds. To keep it in bounds, some clamping is used.

Render

For some reason, adding ants to the sand block is the first thing that is handled here. It is a simple if statement that could be handled in Update, but there isn’t any time left to move it.

The first main section is rendering all living ants. Ants are scaled down by 0.5, before being repositioned and scaled down again by 0.0833 to fit inside a mini cube. After the ant has been rendered, a check is done to see if it is eating any mini cubes. By taking the rounded position co-ordinates of the ant, these are compared against the position of every eaten mini cube (eaten cubes will always be smaller in quantity compared to non-eaten cubes). If the cube hasn’t already been eaten, then it is stored in eatenCubes.

Next is rendering all blocks in the scene. Depending on the block type, a unique texture is applied. If the block is sand, it will either show the glass casing or the sand cubes depending on the user’s choice. If sand is being displayed, it will show either the non-eaten sand cubes or only the eaten sand cubes – rendered as glass. Because the transparent eaten cubes are being rendered first, there is a bug with transparency where the user can see the skybox through the floor.

After all opaque blocks are rendered, glass blocks are then rendered afterwards to improve the transparency look. Looking back at how this is coded, some obvious changes come to mind. The program loops through every cube twice rather than storing all glass cubes to be used afterwards, this would save substantial processing time, especially if this was scaled up. Rearranging the render order of blocks should fix the transparency issues.

The “block hand” is the last block to render, as the transparency bug was the most obvious with this block. Afterwards is the bloom / monochrome post-processing effect being applied. The last part of this section is rendering the GUI in the top left. This was used for debugging issues where the cameras position and the first block being collided with is being displayed.

Collision

Handles collision detection for the player and every block. The player is made of two imaginary blocks, “head” and “feet”. These are calculated by using the cameras position, with feet being one block below head. While it does work, there is some logic issues either in here or in its application in Update that caused the player to be trapped on a block while gravity was being applied. The only way for the player to move beyond the box they were standing on was to jump then move mid-air. This is why the gravity was removed.

Evaluation / Reflection

While I am happy with the completed demo, it does not meet the criteria of what I hoped to achieve. While the ability to place more sand blocks were implemented, it was shortly scrapped due to performance issues. Perhaps if more time was spent investigating ways to reduce this (perhaps by creating a function that only renders a mini cube if all 6 sides are visible) then it would meet my own standards. As mentioned previously, the code hasn’t been refined to an ideal state, as Vector cubes is looped through too many times unnecessarily. I am quite disappointed with not being able to implement mouse movement – and ray cast block placement – due to time constraints caused by an injury to my dominant hand.

Procedurally the ants are almost identical to what I set out to create. The only main difference is that instead of digging direction being a vector, I wanted to use randomized angles for each axis (or quaternion rotation applied to the ‘front’ of the ant, rotated around the centre of it). This would have allowed for spiral effects in 3D space to be created. Some flies could have been hard coded to do so as part of the demo.

Initially the plan was to have the cubes be more dynamic, rather than just visible or invisible. There was a cube object created with a vertex design to allow partial consumption. It looks like a 2x2 cube with diagonal lines cut across each face. This would have allowed for more dynamic looking tunnels created by the ants.

As stated, several times already, a hand injury and severe time constraints (caused by the injury, other modules, and life) led to several features being reduced in scope or flat out removed. Despite this, I am happy with the work that I have been able to create for this module.

References

Developer.mozilla.org. 2022. 3D collision detection - Game development | MDN. [online] Available at: <https://developer.mozilla.org/en-US/docs/Games/Techniques/3D\_collision\_detection> [Accessed 10 May 2022].

Walbourn, C., 2022. BasicPostProcess · microsoft/DirectXTK Wiki. [online] GitHub. Available at: <https://github.com/Microsoft/DirectXTK/wiki/BasicPostProcess> [Accessed 10 May 2022].