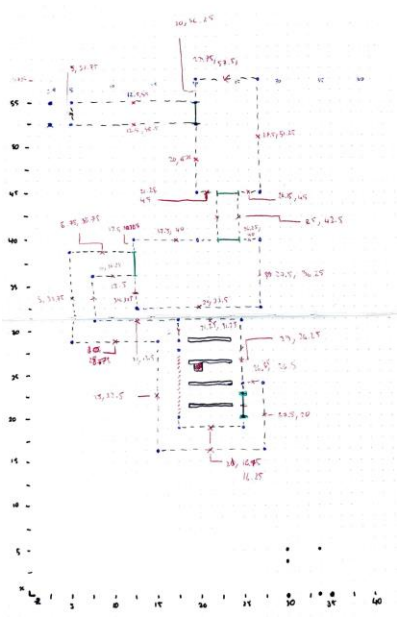
Computer Graphics: Assignment 2

Max Barker,
19624729

The Map and Composite Objects

When the player is loaded into the map, they start in their office or “starting room” where they are greeted with the power going out. After picking up the flashlight, the player traverses into the main hallway only to find the Enemy spawning in front of them, and the exit door shut. The only way for the player to get out is by running to the server room, pressing the button to open the door and escape without getting caught by the Enemy.

Modeling the Map



In order to ensure that the map creation would follow a logical process, I drew out the entire map with every important position noted. With each of these positions I could then step by step draw the entire map. However a problem presented itself: How am I going to render each of the models without creating hundreds of lines of text?

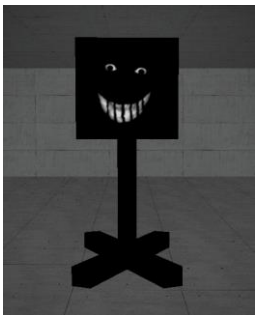
My solution which was later used for my composite objects was to create a struct called Buildings. This struct stored the translation, scaling and texture coordinates for each object.

An array of Buildings thus can store every single object for the map, and can be looped through to draw the environment on each frame.

The only things that were not stored were the textures that represented each object, but this was fixed by having the textures load before every loop.

The biggest benefit of the Building struct is that it allowed me to have an array of Buildings for every single composite object in the game without having wall after wall of text.

The Enemy:



I wanted to create something creepy that didn't make a lot of sense. Since jump scares are forbidden, I felt that an unsettling (or comical) monster would be a good choice for the game. The enemies legs will rotate into each other, giving it a strange gliding appearance across the floor.
The enemy can also phase through walls. To balance this I made it very slow but do not underestimate it as the Server Room is essentially a dead end where it will be trying to get you.

The Exit Door:

The Exit Door is a composite object that is the player's escape given that they have pressed the button. The door has three mat4’s that represent its frame, an inner mat4 that is the door, and a glowing exit sign to the right of the door. When the button is pushed, the door texture changes to a night sky texture.

Starting Room

The starting room is, you guessed it, where the player starts at the first run of the program. The room has 3 composite objects which are a computer, table and a torch. There also exists a photo frame, however it is simply just a flat rectangle with an image on it.

Computer

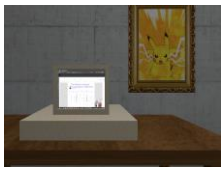
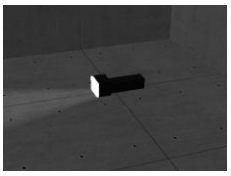
The Computer in the starting room is a composite object made up of 6 parts: The computers base plate, 4 frames for each side of the monitor, and a screen within the frame. The computer will initially be turned on, until the power shuts off in which the screen will go to a skull. The frame also, while not a composite object, will respond to the lights going off.

Table

The table is a simple composite object, consisting of 4 legs and a tabletop. It is used as a desk for the computer, and undergoes no changes throughout the program.

Torch

The torch sits on the floor in the starting room. Initially, the spotlight
will beam out the end of it until the player picks it up. The torch has two models which are its handle and the metal case surrounding
the bulb. The bulb itself is also a model that will glow white. When
the player picks up the torch


Server Room

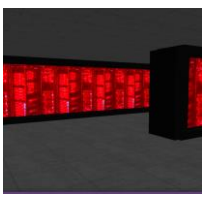
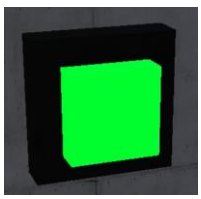
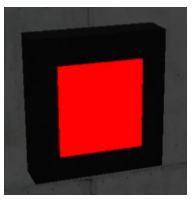
The server room is where the player will find the button which opens the exit door. It contains 4 servers, which will change depending on if the button is pressed. It also includes a button, which when pressed, changes from green to red and will depress and extend.

Servers:

A server is a composite object that consists of four mat4's that are the frame for the servers, and a fifth mat4 that represents the servers themselves. One server is stored as a Building, which then is copied 4 times into an empty array allowing us to avoid code duplication. When the button is pressed, each server will translate to the left or the right based on whether their index is odd or even. The new locations are used to make it a bit more difficult for the player to exit while the Enemy approaches.

The Button:

The button is a composite object that has two mat4's that represent the button holder and the button itself. Initially the button will glow green, but when it is pressed an animation will play of the button being pushed in and out. The button will also turn red along with the servers.

Functions

Function: followPlayer

To translate the Enemy to the Player, the distances between the two are found firstly by subtracting the enemies X and Z positions from the Players X and Z positions. With these two points we can use the Pythagorean Theorem to calculate the hypotenuse. Finally, dividing the X and Z distances by the hypotenuse and adding the directions to the Players current position we can simulate the Enemy moving towards the player at a slow rate. The distances are added to the enemy position however the positions are also multiplied by 0.01, which we can treat as the enemies velocity. The position is then retired and the Enemy's model can use this to update its position.

Function: rotateToPlayer

The function rotateToPlayer takes the distances from the X and Z positions of the Player to the Enemy. Then using atan, we can find the angle between these two points. Then multiplying the rotation by 180/π, we are left with a float that represents the current rotation of the Enemy towards the player.
This is then passed back to the mat4 of the Enemy and is rotated (in the negative direction to ensure it rotates the correct way).

Function: startGame

To implement an ability to restart the game startGame was created to set some of the Boolean conditions and model positions back to their original locations/values. It also sets the time to 0.0f using glfwSetTime(). This allows us to pseudo restart the while loop, without ever exiting it. The servers in the server room must be redeclared, as the locations of the servers change after pressing a button. To do this a nested for loop resets the values of each of the four servers.

Function: bindTexture

This function takes in 3 textures as arguments and binds them. The reason this function exists is to prevent massive amounts of code that arise from the binding of each texture, which in my program takes up 6 lines (diffuse, specular, glow).

Function: detectCollision

This function is quite simple. Given the location of the player and the hitbox of the player, along with the location of the object and the hitbox of said object, we can detect whether the objects collide or not. The YouTube video linked in my references was what helped me with the implementation. Every hitbox is divided by 2.0f, because without this the hitboxes are double the length, which means that walls extend further than they should. DetectCollision will return a Boolean of true or false depending on if there is a collision.

Function: preventPlayerMovement

To prevent the player from walking through walls preventPlayerMovement was created to respond when detectCollision returns true for certain Building arrays. Similarly to detect Collision, we are working with the camera position, camera hitbox, object position, and the object hitbox. This method runs for things in the game that must act as a barrier (i.e. walls, desks, servers). When the camera has collided with a barrier, we need to prevent it from continuing. How? Well in this function when the player collides with a barrier the player's position is sent backwards very slightly to prevent them from passing through. The direction in which the player is sent depends on the side of the barrier they are on. Four separate checks are performed to determine which direction the player must be sent: In the positive X, negative X, positive Z and negative. The amount that the player is sent back depends on the size of the hitbox and location of the barrier and the camera.

Function: collisionLoop

CollisionLoop contains all the necessary checks and updates needed for every frame of the program. This includes checks like whether the player is in range of the button or the enemy. It also controls what happens when the player collides with things like walls. The collision loop contains the hitboxes needed for certain objects in the program,

Animations

There are 4 animations that can be found in my program. They are the starting animation, the button click animation, the enemy animation and the server animation.

Starting Animation

When the game starts, the player is standing in front of a desk when suddenly the lights go out, the computer screen changes, and the player turns his head towards the now dark hallway. To control this animation, I have a check within each frame of the program that sees if the game has been running for under 7 seconds. If it is, then there are a few things that will occur. For 3 seconds, the camera will look at the computer. On the 4th second, the lights will shut off. Then following 3 seconds involve the camera slowly rotating towards the right of the screen. Once the animation is over the player is given control over the mouse. To make the animation smooth and progressive, a float is added to the players front X and Z coordinates progressively each frame.

Button Click Animation

For the button I wanted a realistic button press animation, where the button would quickly but smoothly depress and expand. I felt that this would be a nice subtle addition.

The moment the player clicks the button, the game saves the exact moment in time in a float with an extra second added onto it. Then the blackout Boolean is set to true. Every loop after that the program will check if the current time is less than the time captured during the button press plus the extra second. This essentially allows us to have 1 second to animate the button and no more. Before 0.5 seconds, the button moves inwards via translation and then after 0.5 seconds but before 1 second the button moves back out.

Server Animation

I needed to add some extra difficulty in the server room to make it more likely that the player could get caught in there by the Enemy. I came up with the idea for the servers to turn red and move left and right, forcing the player to run a longer direction to get out of the server room which is essentially a dead end.

When the button is clicked, for the next 5 seconds each server will be translated left and right depending on whether the server is of an even index in the server Buildings array. Each server will slowly glide to the walls to block the player from exiting the same way they came as that spot in the level is very claustrophobic and easy to get caught in by the enemy.

Enemy animation

To make the enemy more unsettling I added an animation in which the enemies weird legs would spin on the spot. It gives the effect of the enemy gliding along the floor towards the player.

To achieve this I made it so that on every frame the enemy would rotate by 45 and -45 degrees multiplied by the current time. I multiplied it by the current time to allow for a easy way to progressively change the rotation of the legs.

Death and Winner Implementation

When the player collides with the enemy, the DEAD Boolean is set to true, and similarly to the Button Click, we store the exact moment plus an extra 10 seconds to give the animation some time to timeout before the game naturally restarts. When this happens, the camera Front is set to 1.0f on the Y and 0.0f on the Z and the X forcing it to point up. It is also translated to floor level.

When the exit door is opened and reached the exact moment plus 10 seconds is stored, and the players camera is translated to a cube that says, “You Win!” on it. This is a sneaky way of having a victory message without adding text functionality into the game.

Lighting

The Lighting in my game comes from 3 sources: Directional lights, point lights and a spotlight. Each of these are from the LearnOpenGL multiple lights tutorial. I also have a special light that glows, however emits no light source. Each light handles attenuation given by the algorithm from LearnOpenGL and the lectures:

$$F_{att} = \frac{1.0}{K_c + K_l * d + K_q * d^2}$$

This allows each light to die off over long distances, which is especially important for the spotlight. I will now go into more detail about the purposes of the three lights:

The Directional Light

The entire purpose of this light is for when the <O> key is pressed, and the scene is illuminated. It essentially works as if it were a sun in the sky that can shine through walls.

The Point Light

I had some trouble implementing this one as the example in LearnOpenGL was not set up for the point lights to face downwards. Each point light in the building will give some lighting around the scene to create a scary atmosphere. The tiles reflect of these lights which give them a shiny appearance. When the button is pressed, every light shuts off and the area goes completely dark. Unfortunately, I could not get the lights to stop passing through walls, so perhaps that can come another day,

The Spot Light

This is the most important light in the game as it is carried by the player. Initially, the pointlight is set to point from the ground at the wall in the starting room next to the flashlight object,

Textures

I made good use of textures throughout the development of this assignment to try to make the game a bit nicer to look at. I used specular and diffuse textures to allow different textures to have different shadings.
In many cases I had to make my own specular maps to get the best result.
Some examples of my different shading techniques used are the wooden table in the starting room vs the tiles on the floor. The tiles are reflective whereas the wood is not as its specular map is the same as its diffuse map to give it a flat look to it.
Some textures are later swapped over to another texture for added effect, such as the photo frame changing when the lights go out or the button/servers changing color.

Sources:

https://learnopengl.com/ - A large part of my code comes from the tutorials
https://www.youtube.com/watch?v=ENuk9HgeTiI – Where I learnt the collision detection algorithm
https://www.textures.com/ - An abundance of my textures came from here
http://www.peroxide.dk/papers/collision/collision.pdf - This thesis on collision detection. Not super useful but definitely helped my understanding