CITS3003 Scene Editor Project Report

Serena McPherson - 24211939
Zane Meleca - 23663079

Camera Panning

This task introduces a resetSeq boolean to control the update logic in a camera system. The flag ensures that during a reset frame, no other camera updates occur. The panning sensitivity is adjusted based on the camera's distance and window size. Additional modifications include refined control over yaw and pitch updates only outside of reset frames. The view matrix is constructed manually, allowing precise control over camera transformations, which is essential for rendering scenes with potentially unbounded depth, using an infinite perspective projection. Depth clamping is enabled to handle fragments beyond the projection planes effectively.

Entity Rotation

This component required that the sliders for *x*, *y*, and *z* rotation of objects in the scene appropriately and accurately perform said rotation. This is a simple implementation of taking the values set by the *GUI* sliders and using them to perform the rotation calculation. A rotation matrix is created to store each of the three axis' rotation. Then the values for these axes are accessed from the euler_rotation variable to be stored in this matrix. Finally this rotation matrix is multiplied with the existing translation and scale matrices to produce the final result where objects can be rotated about each axis. During the construction of this it was noticed that different results occurred based on the order of the axes rotation calculation. The order *y* -> *x* -> *z* was decided upon as it functioned in all directions and provided expected results. The final set of matrix multiplication has the rotation being the second calculation applied (after scale).

Preventing Clipping

When updating the PanningCamera and FlyingCamera, we implemented several enhancements aimed at addressing and preventing clipping issues. Firstly, we introduced distance and angle clamping, which involves setting specific limits on how close the camera can get and how it angles itself, ensuring that it doesn't accidentally cut through nearby objects. We also adopted an infinite perspective projection matrix, allowing for a depth range without boundaries and mitigating the risk of clipping at both near and far planes, which is particularly beneficial in scenes with considerable depth variations. Another key enhancement is the enabling of OpenGL's GL_DEPTH_CLAMP, which prevents objects close to the camera from being clipped by clamping them to the nearest depth values. Additionally, we made regular updates to the view matrix according to the camera's orientation and position to maintain optimal viewing angles and distances, thus reducing the potential for geometric intersections within the camera's viewing frustum. Lastly, for the FlyingCamera, we implemented a velocity decay mechanism that activates when there is no user input, maintaining a stable camera speed and preventing sudden, disorienting movements that could lead to clipping.

Material Properties

This component required that the intensity of the *ambient*, *diffuse*, and *specular* properties of objects the adjustable via sliders in the *GUI*. This simply required implanting a ImGui::ColorEdit3()element for each, with each accessing and modifying their respective tint's *r*, *g*, and *b* values. This alone is not enough however as

nothing is calling the scene to redraw when changes are made. A small *if*-statement is required under each slider's instantiation that checks if that particular item was active and sets a variable to True, indicating that the scene should be redrawn. Similarly, a slider to access and modify the material's *shininess* property was also implemented.

Texture Scaling

This component requires that textures applied to surfaces can be scaled using a slider in the *GUI*, the vertex shader for entities has a texture_scale variable that can be accessed and modified to accomplish this. Another slider can be added that modifies said variable, the range *0* to *100* seemed appropriate. Adding this to the list of attributes that can be saved out to a file was accomplished by editing the material_into_json() and update_material_from_json() functions to include an entry for *texture scale*.

Light Attenuation

This component required that the light emitted from point-lights decay or *attenuate* as the light reaches further out. This can be done easily using the *inverse square law*. The distance between the light source and the fragment is already calculated and stored in ws_light_offset so the length() function can be used on it with the result stored in another variable. The attenuation (or inverse square) can then be calculated using this value `attenuation = 1.0 / (distance * distance)`. This alone is sufficient but the rate the light attenuated was too great, so a constant can be multiplied with the distance square to alter this effect. After some experimentation, *0.09* ended up being a value striking a good balance for this constant.

Directional Light

We were unfortunately unsuccessful in implementing the directional light, we both attempted this task but did not figure it out in time.

New Feature

Randomisation of the colour of scene objects was implemented as a button on the *GUI*. For the emissive and non-emissive entities this was implemented using the rand() function in the <cstdlib> library. An *ImGui* button was added to both of these components; upon pressing them, nine random values between *0* and *255* are calculated and converted to the decimals stored in the material tint matrices. This completely randomises the *ambient*, *specular*, and *diffuse* tints. The point light colour randomisation was implemented a different way. This used the linearRand() function from <glm/gtc/random.hpp>. The same type of button was implemented but instead pressing it sets the *colour* attribute of the *light* to a vector of three random values between zero and one (and the final intensity value kept the same). This effectively randomises the colour of the light.

Problems Encountered:

We encountered a lighting issue when syncing up our git repository where the lighting was only functional on one of our devices, and the other person would see black. This was stressful as it was nearing submission. It was due to inconsistency in the frag and vert shaders across the project and we managed to fix the issue by completely starting a new git repository with the foundations of the functioning scene editor.