Scott Owen James

Daniel Phan

**Computer Graphics coursework 2 Report**

**Introduction**

Basic scenery of miniature chapel was produced using groups of multiple objects using complex objects constructed from code. Scenery was simple but was surprisingly complex to put together: this was because of the differences in the initial constructional positioning of objects, y=0 was the floor, but for some objects that means they were right on top, some half submerged, some had initial starting widths of 2 and some had 1 which made scaling more complex than necessary.



Figure 1: Chapel scenery

**Camera Controls**

Camera moves in response to users input on WASD keys in the standard directions as they relate to the current direction of the camera. Q and E can also be used to move directly up and down. Left shift and left control modify the speed by increasing or decreasing the speed. Moving the mouse moves the direction the camera is facing. When the user presses space the mouse is locked in the window and hidden until space is pressed again, this is just because it kept leaving the window when moving the viewport. Camera is entirely frame rate independent based on change in time. To implement this, I adapted implementations found from learnOpenGL, however used opposite camera front/direction values to get the camera to face the correct direction.

Vector and matrix classes have both been fully implemented, as evidenced by camera movements and the number of translations made to arrange the scenery. Added cross product function to assist in calculating camera directions.

**Lighting**

An array of six pointlights are fed into the shader to be used for full illumination. All lighting follows the Blinn-Phong model and appears to be implemented correctly. It was difficult to fine tune this, the model is not 100% accurate and with scenery of limited complexity it was difficult to tell the difference between errors made and uncanny valley, especially considering an error could come from the shader which didn’t produce proper errors. Initially tried to do this using pure mathematics, later found out on learnOpenGL that the shaders had a variety of functions that could be implemented to achieve most basic mathematical tasks without risk of mistakes, so they were used. The normals implemented both standard and smooth shading, the cubes and some building parts used the standard algorithm of calculating the normals of a side, but for the pillars used smooth shading to give it a more cylindrical appearance, this was an accident at the time, and I only found out about it later but kept it due to the better effect.

https://learnopengl.com/Lighting/Basic-Lighting

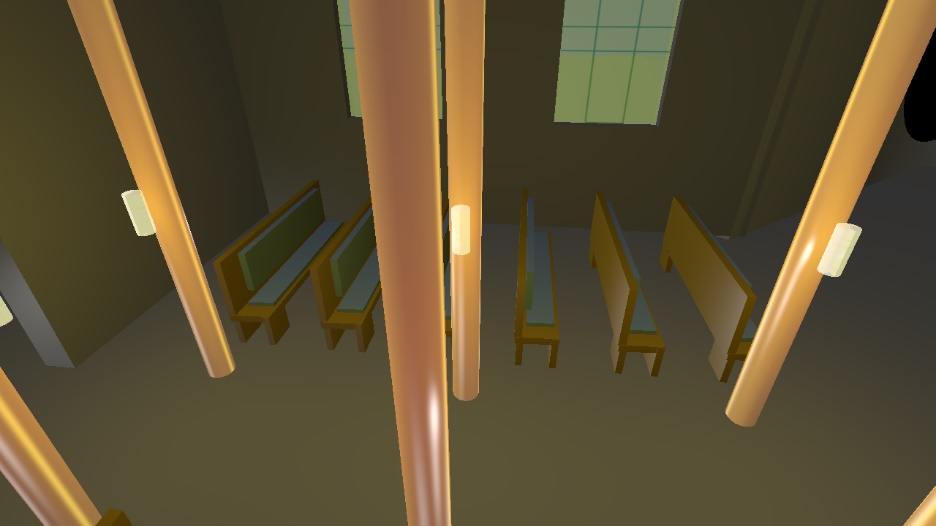


Figure 2: Lighting

**Animated object**

Chapel door opens and closes independent of the framerate, JKL keys toggle the controls to slow down, stop/start, or speed up. Rotating the door required knowing the position of the hinge and fixing the angles of rotation. The normals were also rotated in the vertex shader before being passed to the fragment shader to make the lighting stick.



Figure 3: Animated door object

**Diffuse/Specular/Emissive**

Emissive material is just an untarnished block, mostly unaffected by any light sources, with specular effects just barely being visible and light sources otherwise not affecting it. The diffuse block has an entirely uniform colour across its individual surfaces, but the sides that have no light are near black bar ambience, which is expected. The specular block was replaced with a specular cylinder because it is easier to see the specular effect, but honestly this doesn’t appear to be very impressive, there is a clear specular mark, a higher shiny value might have been better, other objects like the pillars show better specular effects.



*Figure 4: Diffuse, Emissive, Specular objects*

**Texture mapping**

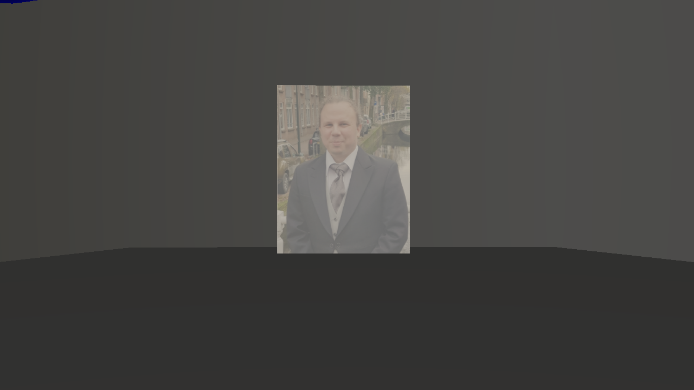


Figure 5: Texture Mapping

Texturing was applied to all objects, attempted to calculate all correctly, but some may be incorrect, however they needed to be there to fill the right spaces in the buffer for other objects. Was successful and able to add the texture that came with the coursework to the scenery with little difficulty. Used the stb\_image.h third party library to achieve this. Made an attempt at multi-texturing, having one for diffuse light and one for specular light, but was unsuccessful. Textures are correct. And are for both the cube and the cylinder.

**Light sources**

Directional lighting was achieved with minimal difficulty. Point lighting and spot lighting was also achieved.

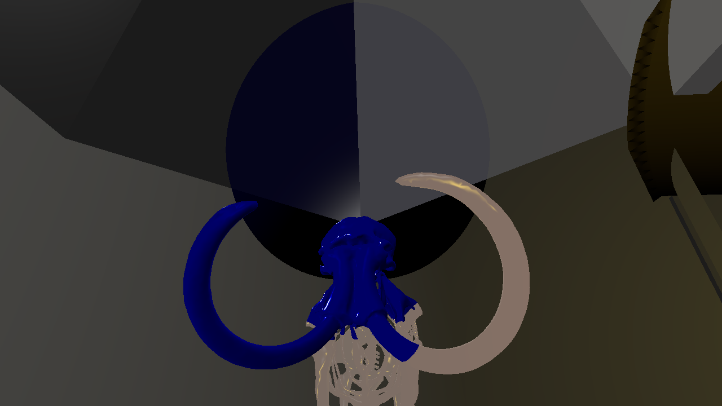


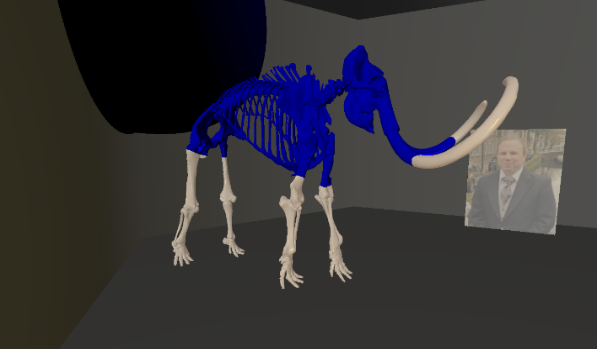
Figure 6: Light sources

Was unsure if the requirement of the coursework was 3 lights or 3 types of light, either way this has been achieved, the directional light affects everything in the scene.

The 6 pointlights are in the main sitting area at the centre of the transparent textured cylinders, the aim was to make lamps, it was good, but would have preferred a stronger white.

The spotlight affects the top of the mammoth skeleton, it’s set with values such that it makes what it touches blue, as the last colour applied it can completely change the previous colours. The spotlight was the easiest to experiment with because it was clear exactly what an effect it was having on its surroundings.

**Model Loading**



*Figure 7: Model loading*

Used two *.obj* files in the project, a woolly mammoth skeleton and chairs. The hardest part was finding free models online that fit the scene. Ultimately used a couple from clara.io, they have all been taken from the file with the correct position, normal, and texture, the chair leg is like that as standard in the file and not a mistake, annoying but still the most fitting chair found.

Texture coordinates were found and functional, though the benches didn’t come with a texture, they are the values from the *.mtl* file. The mammoth skeleton does have its own texture applied and going by how the spotlight is reacting to it, it also has normal values, at a close distance it was surprisingly specular and shiny.

**Transparency and alpha blending/masking**

Transparency was achieved by enabling the transparency function, had some issues with this that only the background colours would show unless the gl\_enable() methods were put in a specific order. Next ordered the transparent objects by distance from the camera, did this by putting it in a map function, checking if space occupied then adding a small amount to map close by. Then drew them in reverse order so the transparent objects could be seen through each other, though would have that same effect if 2 objects were close together. The opacity values of the material is used to produce a final opacity value in the shader. If both opacities are zero the fragment is discarded immediately, tried using both at once to measure for both to have effects, but it didn’t work shown by the stained-glass windows frames, good at a distance but if you get too close you notice the bars are transparent.

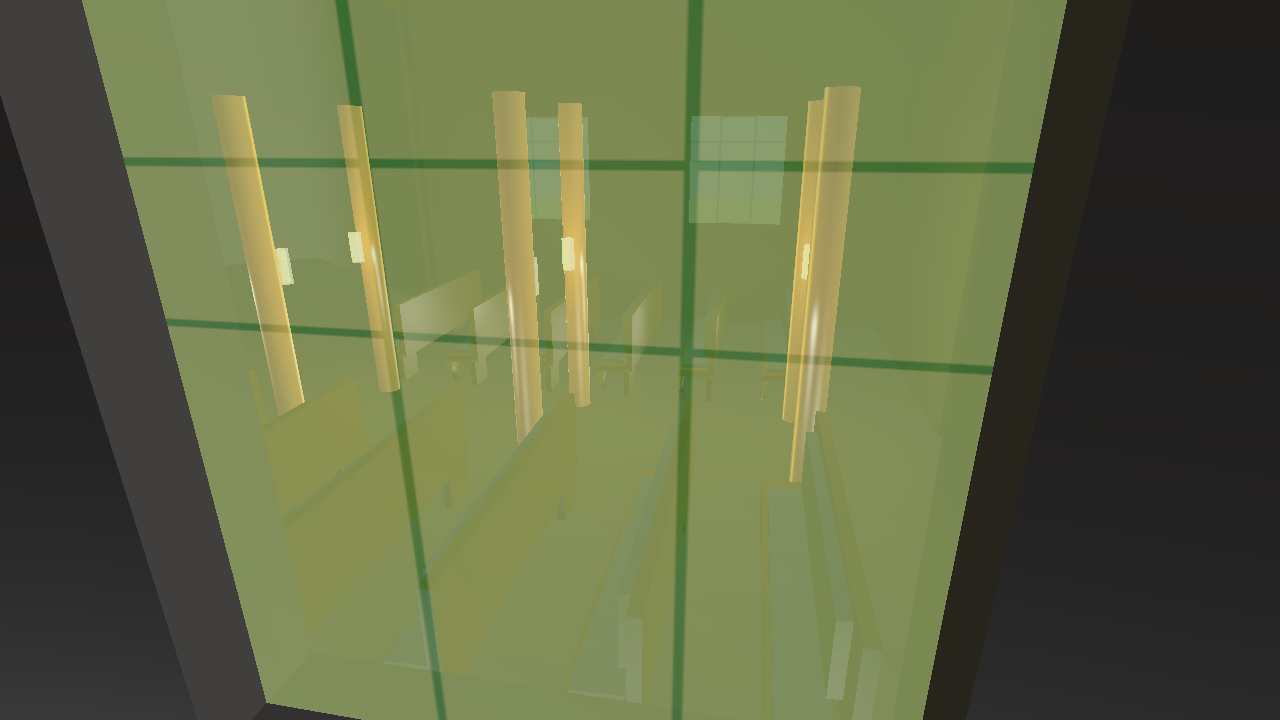


Figure 8: Transparency

**Additional things**

**Jellyfish**

Attempt to make a complex item with multiple moving parts, this comprised of a head that expanded and shrank, a main body, and a predefined number of legs, the whole model is set to complete circuits of the outside of the building, whilst floating up and down, during this the head would cycle through inflating/deflating. This was all successful but failed at segmented leg movement. Throughout the coding of this I had to keep going back and adding new things to the object creation, such as recording the centre point for scaling. If I had to do this coursework again the first thing, I would do is increase the functionality of the SimpleMesh class, so that it had information such as a pointer to materials and textures, but more importantly data concerning the mesh itself such as height/width, centre point and other points of interest that would get translated along with the rest of the shape, and other such things so I did not need to write programs to recalculate them.

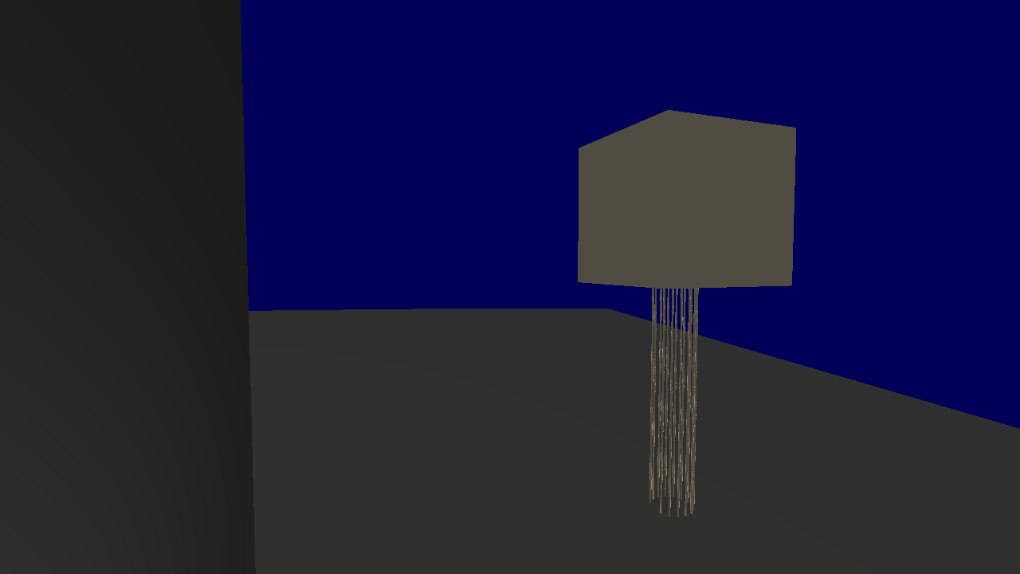


Figure 9: Hierarchical modelling Jellyfish

**Billboarding**

Attempted to get billboarding working, used relative normals and directions between the camera and the object to get it to rotate to face the camera, this was only successful at point (0,0,0), anywhere else it would fail, even shifting the vectors so that the centre was at (0,0,0) was ineffective so I suspect that it was a problem somewhere else. Left the 1 functional sprite in the code.



Figure 10: Billboarding

**Screenshot**

Every image of the system so far has been taken using the screenshot function reads pixel data to buffer, passes it to the custom writer, reverses the pixels so they’re back to normal, creates a name for the image and save to the main area. It’s tied to the enter button, while it works the frame rate does take a noticeable hit when it runs, this happens at pixel reversal, not sure how I could have improved it. Initially didn’t properly perform the reversal, which is why some of the images above are flipped left to right.

Appendix – Credit section

[https://learnopengl.com](https://learnopengl.com/)

clara.io - models

Value for brass:

<http://devernay.free.fr/cours/opengl/materials.html>

|  |  |
| --- | --- |
| Item | Credit |
| Camera movement | Daniel |
| Vec4/vec3 /etc initial files from exercises used | Daniel |
| Scenery layout | Scott |
| Point lights | Scott |
| Spot lights | Scott |
| Blinn-Phong shading implemented | Scott |
| Animated Door | Daniel |
| Diffuse/Spec/Emissive | Scott |
| Texturing | Scott |
| Normals | Scott |
| Loading Models | Scott |
| Transparency | Scott |
| Screenshots | Scott |
| Jellyfish | Scott |
| Billboarding | Scott |
|  |  |