# VERTEX MANIPULATION

Relevant Files:

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/manipulation_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/manipulation_vs.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/ManipulationShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/ManipulationShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/DetailedPlaneMesh.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/DetailedPlaneMesh.h

**Relevant Screenshots and Diagrams:**

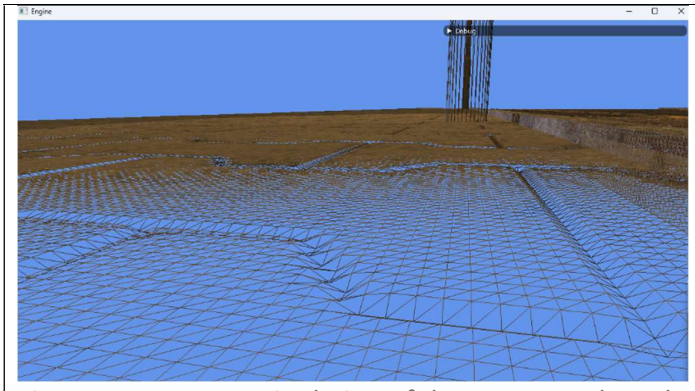



Figure 1: Vertex Manipulation of the pavement based on provided data.

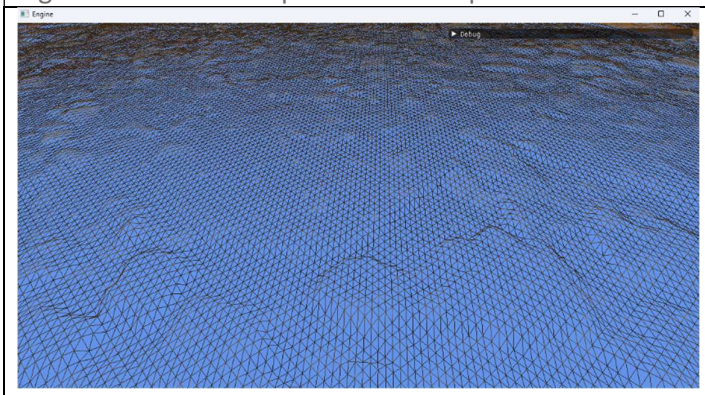



Figure 2: Vertex Manipulation of the road based on provided data..

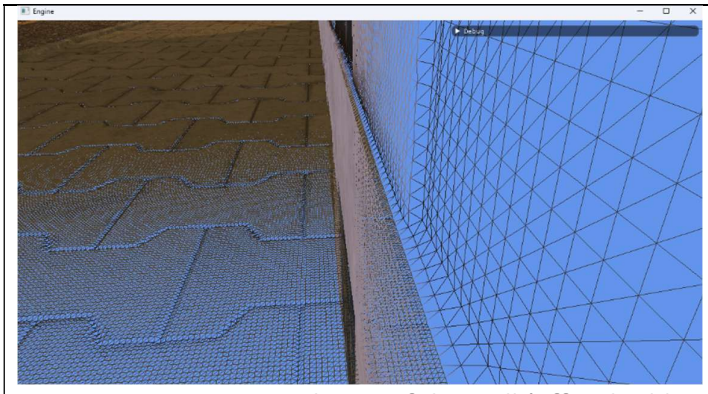



Figure 3: Vertex Manipulation of the wall (office building face) based on provided data.


Figure 4: Proof of correct texturing, correct lighting and proper normal calculations.

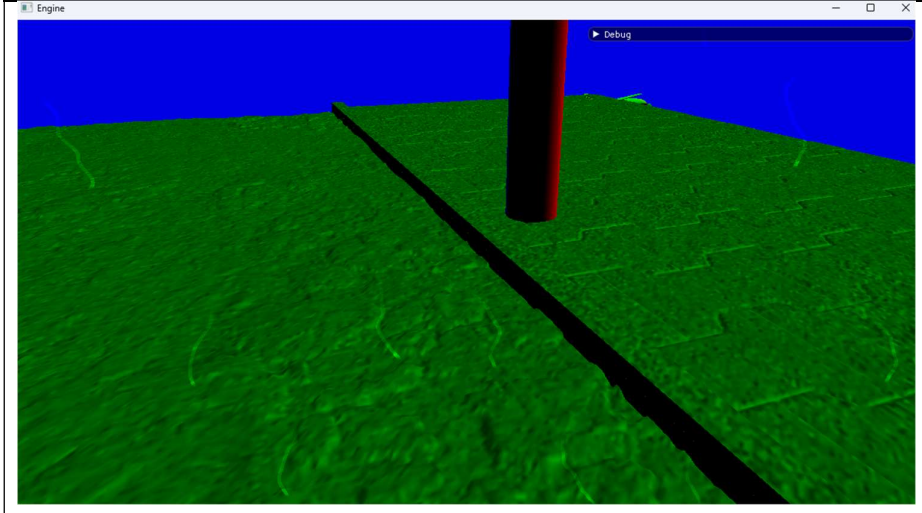


Figure 5: Office Face, pavement 2, road, curb and street lamp model

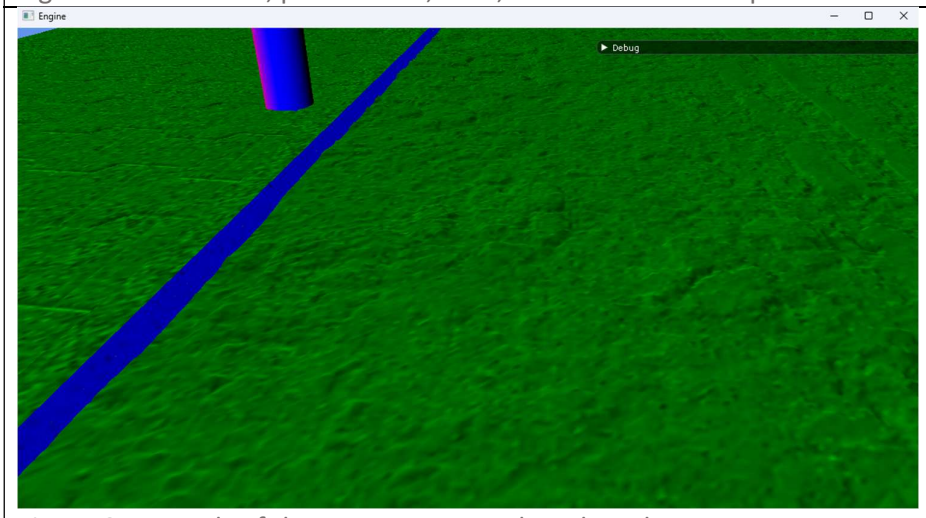


Figure 6: Normals of the pavement 1, curb and road

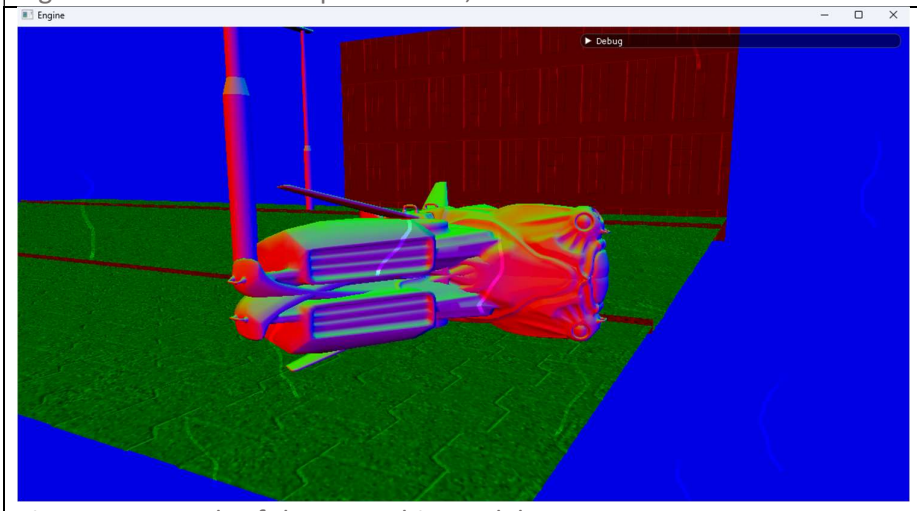


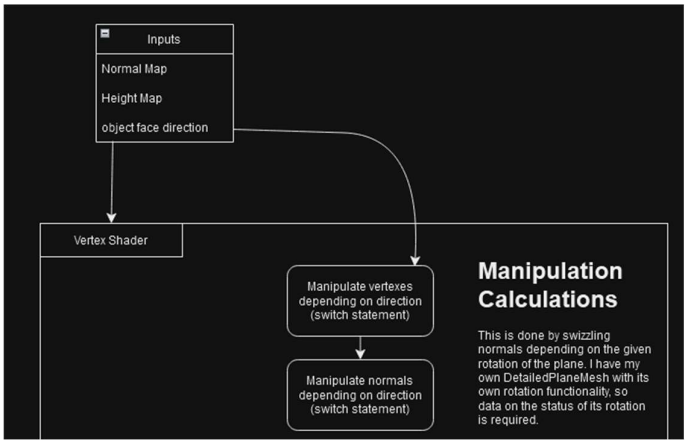
Figure 7: Normals of the spaceship model

**What was implemented:** To implement this vertex manipulation, I used heightmaps. They were passed into the vertex shader to manipulate the vertices. On top of this I used a normal map to map and adjust the normals to correlate with the vertexes that had been adjusted. The pavement is detailed in order to show the accuracy of the vertex manipulation and normal mapping as well as to show a better Graphical output. The models (streetlamps and spaceship) in the scene feature no vertex manipulation. The texture for colour was applied to the texture coordinates. For this, the vertex shader stage and pixel shader stage

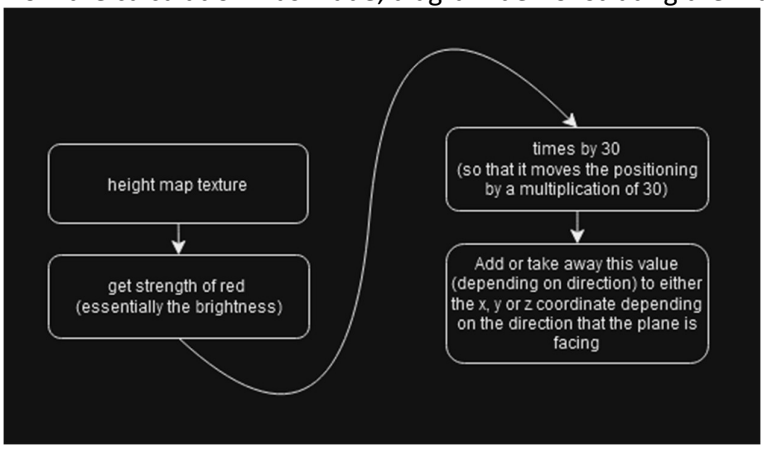
of the graphics pipeline was used. All three of these textures have been passed in to generate these effects.

**How was it implemented:**

My normals were created by taking a float, with a number indicating a direction that my object “DetailedPlaneMesh” is facing in order to appropriately manipulate the normals and vertices accordingly depending on the direction it is facing otherwise it would manipulate and adjust normals when rotated as if it was facing upwards, not giving the effect intended.




How the calculation was made, diagram demonstrating the mathematics needed:




**Critical Analysis:** Initially when I started the project, I didn’t know that there were functions that would scale and functions which would rotate these planes. So in order to solve this issue, I made a new Class “DetailedPlaneMesh” which inherits from the PlaneMesh class with some edits, it takes more parameters, such as size, shape, rotation and position along with the resolution. I wanted the meshes to be high in resolution so that I could demonstrate the pavement grooves well and so that they looked nice since the lower resolution grooves looked extremely unappealing. I set out to create this new class, which worked well, however, I found later when generating normal maps and initially before that, the problem of vertex manipulation, that due to the fact that it was not using the inbuilt XM manipulation functions, it was sometimes, like in the case of my office building face; manipulating upwards, and not upwards away from the face of the plane, which instead just caused a really weirdly manipulated and distorted mesh which had no visible 3D aspect to it. The same was the case for the normals, in that a rotated PlaneMesh thought that the face was facing upwards when it wasn’t. I got around this by pushing integer values into a vector which would correspond to each mesh in the scene, with this number in the according index of the vector correlating to a mesh, I used a switch statement inside the vertex shader to appropriately manipulate the vertex manipulation direction (so I edited, x, y or z in a positive or negative direction depending on the number passed into manipulation_vs.hlsl)

this method solved my problem of having used an unconventional method to rotate, size and position the mesh appropriately. Although this was quite useful for creating special shapes for my scene to then demonstrate the lighting on the pavement curbs with the moonlight and street lights. A potential change that I would've liked to have made is to fully decorate the street with bin models, full buildings and other such things to demonstrate more shadows in order to make it more visually impressive. The diagram that I presented above showing the manipulation code used was pretty simple and effective for getting a numerical value to then manipulate the vertices by.

## LIGHTING AND SHADOWS

**Relevant Files:**

Used for the models lighting:

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/LightShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/LightShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/light_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/light_vs.hlsl

Used for the lighting & shadows of the DetailedPlaneMesh(es):

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/manipulation_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/manipulation_vs.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/ManipulationShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/ManipulationShader.h

Depth Shader for getting shading:

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/DepthShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/DepthShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/depth_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/depth_vs.hlsl

**Relevant Screenshots and Diagrams:**

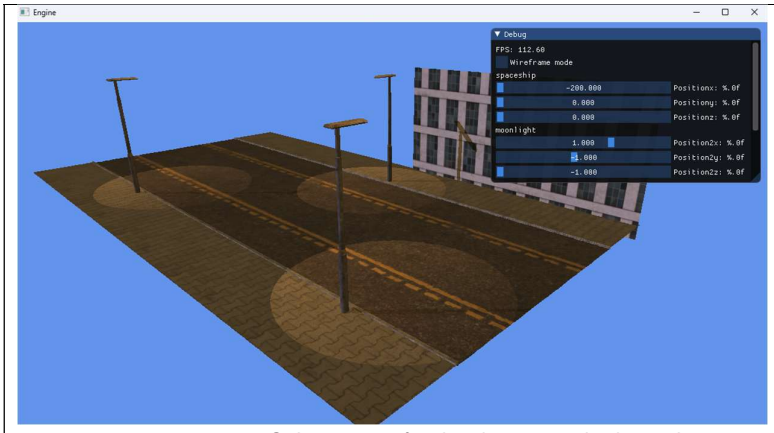


Figure 8: Overview of the scene's shadows excluding the spaceship.

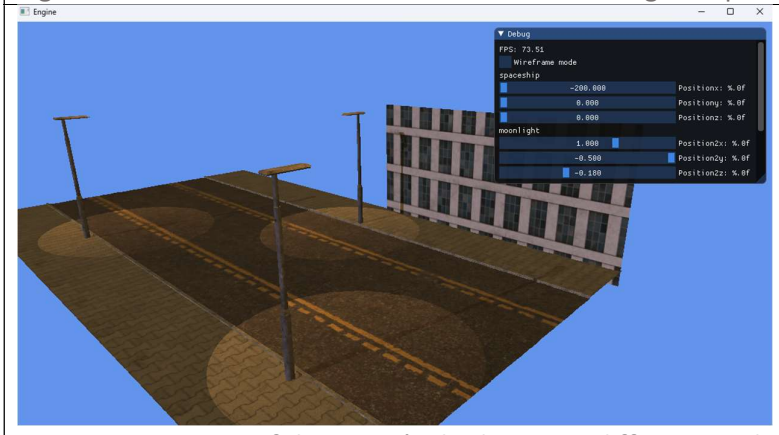


Figure 9: Overview of the scene's shadows at a different angle.

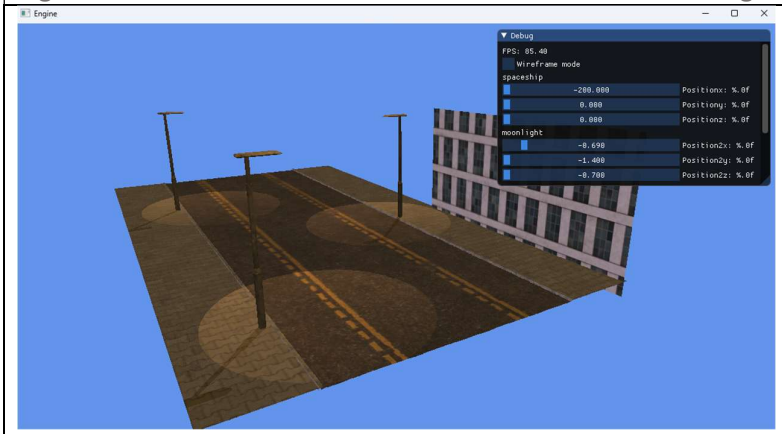


Figure 10: Overview of the scene's shadows at another different angle.


Figure 11: Demonstration of the spaceship casting two shadows from two different light sources. One light being the softer moonlight and the other being the harsher spotlight.

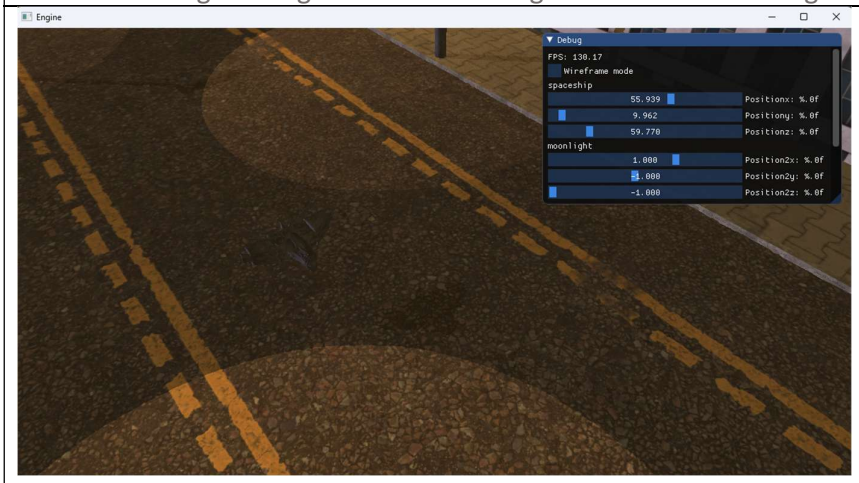



Figure 12: Demonstration of the moonlight light source casting a shadow outside of a spotlight.


Figure 13: Demonstration of the spaceship casting a moonlight shadow on both the spotlight cone area and outside of the spotlight cone area. Appropriately darker shadow outside the spotlight cone and appropriately lighter shadow inside the spotlight cone as would be in real life.


Figure 14: Demonstrating the spotlight cone shadow is only applied within the cone area of effect.

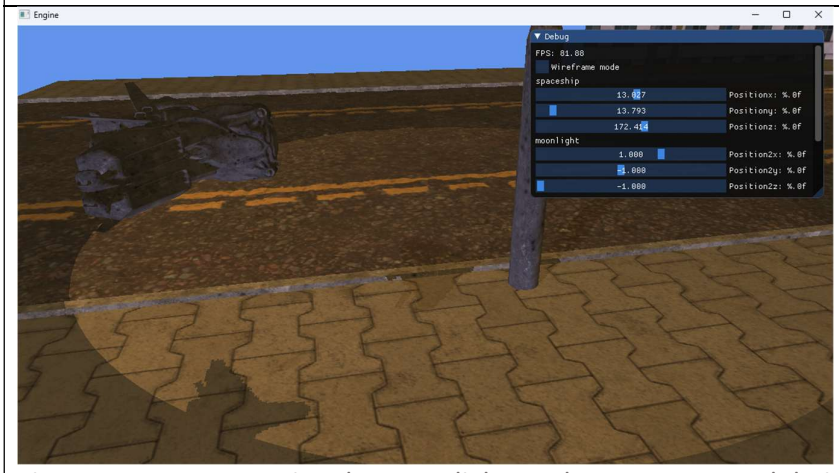



Figure 15: Demonstrating the moonlight on the pavement curb being given a shadow by the moonlight shadow.


Figure 16: Comparing the spotlight cone shadow to the Figure 15 moonlight shadow.

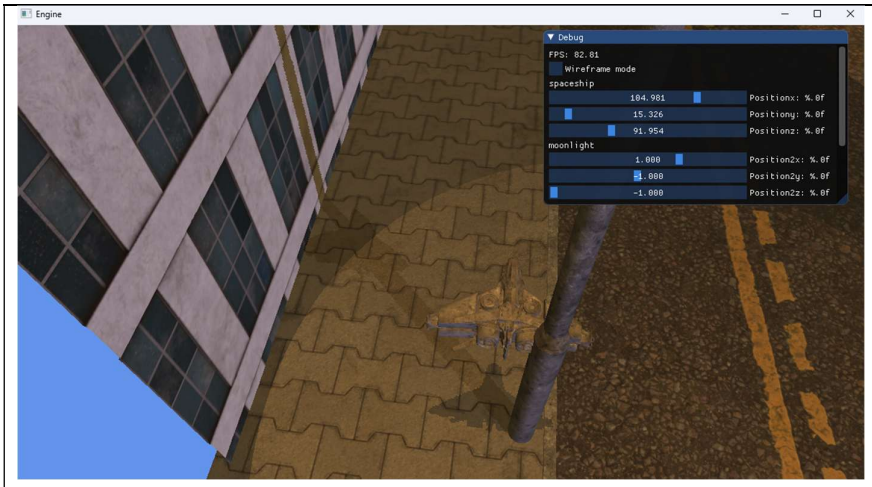



Figure 17: Demonstration of overlapping shadows. Demonstrating the spotlight shadow on the spaceship overlapping with the moonlight shadow on the street lamp. Additionally demonstrating the overlap of the moonlight shadow of the spaceship with the street lamp shadow from the moonlight acting appropriately as a shadow would. Additionally, demonstrating both this overlap both inside and outside of the cone of the street light, showing that the shadows act appropriately with the effects of different lights all at the same time.

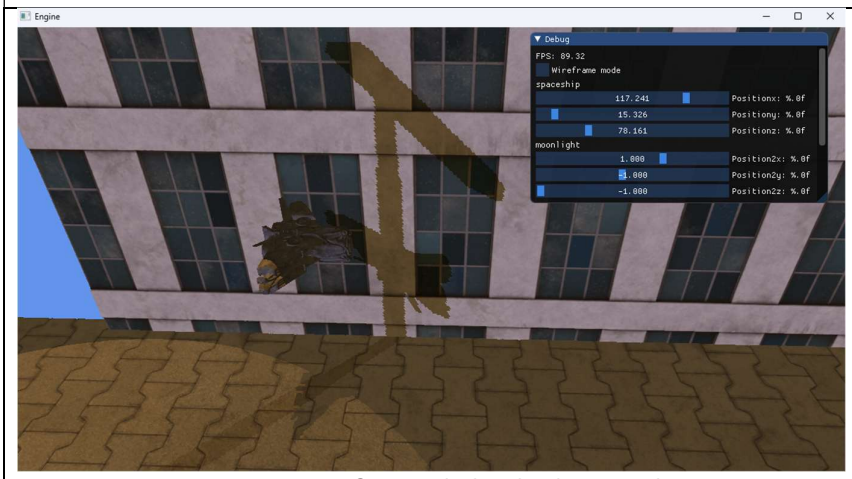



Figure 18: Demonstration of a spotlight shadow on the pavement overlapping with the moonlight shadow of the street lamp on the corner of the spotlight. Additional demonstration of the shadow of the moonlight overlapping with the shadow of the street lamp against the office building face, both overlapping as they should. Additionally demonstrating the correct portrayal of the street lamp profile onto the office building face.

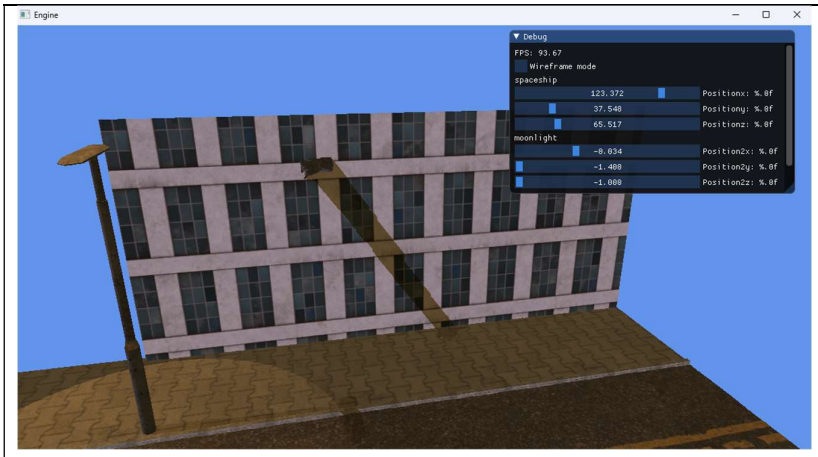



Figure 19: Demonstration of (as you can also see by the streetlamp shadow) a specific direction of moonlight, the spaceship is hallway through the wall and is casting shadows properly onto the pavement and onto the office building face as intended.


Figure 20: Demonstration of the streetlamp shadow both inside and outside the cone of the spotlight, with different levels of shading as appropriate for being both inside and outside the cone of the spotlight, darker outside, and lighter inside as a streetlamp shadow coming from the moonlight would be expected to act.

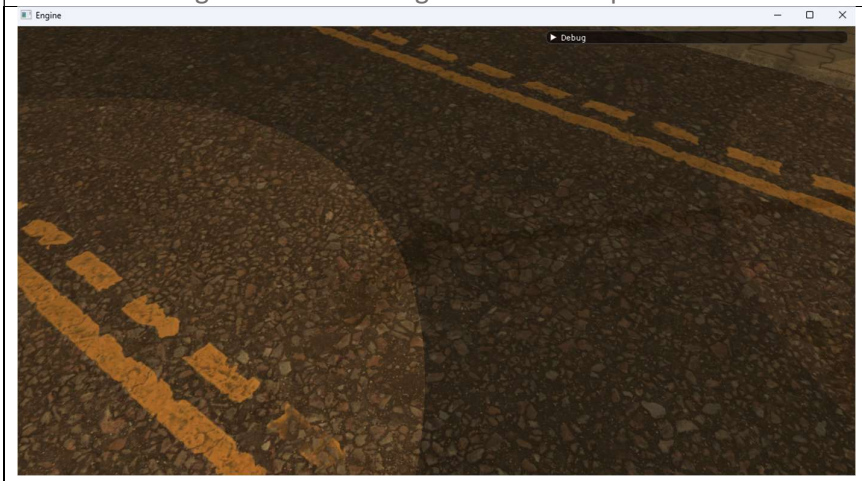



Figure 21: Demonstration of the street lamp shadow cast into another spotlight and acting appropriately (brighter) when cast into the other spotlight cone.

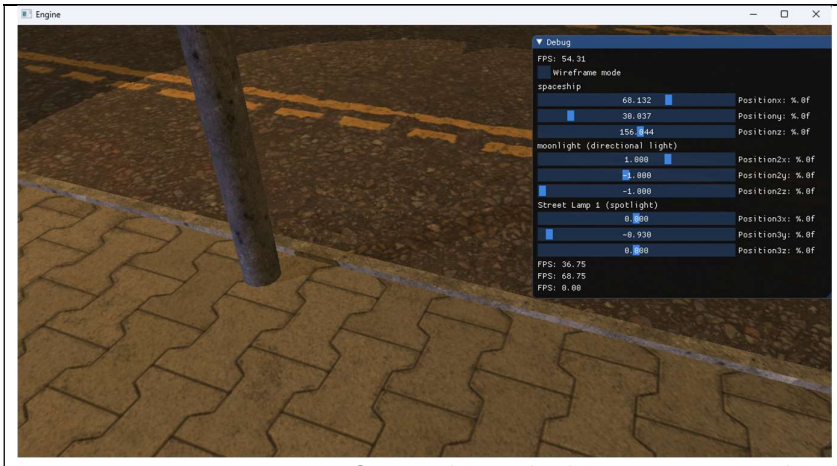



Figure 22: Demonstration of street lamp shadow casting onto the pavement curb.

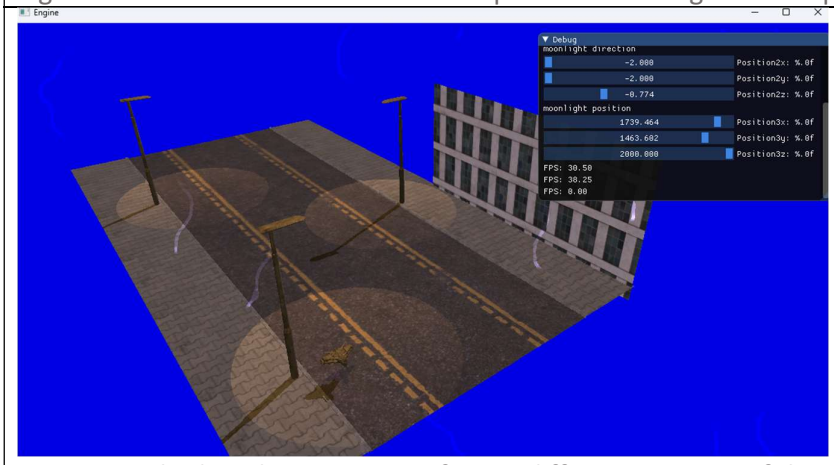



Figure 23: Shadow demonstration from a different position of the moonlight.


Figure 24: Demonstration of direction changed and position moved spotlight.

**What was implemented:** In my scene, I have implemented 3 different spotlights which are all facing downward, casting a cone shape of light. The light is actually above the street lamp for a bigger radius, I could've put it where it would've realistically been placed, but that wouldn't have created the same visual effect that I was going for. I could've also removed the impact as I did in one test of the spotlights on specifically those three street lamp models, but I decided the shading looked more pleasing with them being affected. Very little additional research was done in this department of development. I spent quite a while

trying to figure out how the shadows were being calculated in RenderDoc in order to fix it not working. Only the vertex shader and pixel shader stages of the pipeline were used.

**How was it implemented:** I split my calculations into two different sub-sections (which in the code, I have separated with a dotted comment line. Light "3" is calculated as a directional light separately as the method for calculating a directional vs spotlight's impact is different. The directional light is calculated by adding the ambient colour and the calculate lighting function from a previous week which I very minorly modified. Hence this calculation was pretty simple. However, this is only done if according to the depthMapTexture3 that the pixel is not in a shadow, this is passed into a function to be calculated. My isInShadow function is from a previous week of work. This function gets the depth value by sampling the uv coordinates of the depth map which correlates to the depth from the light view position which was previously calculated in the vertex manipulation shader. In order to be efficient in resource use when making the three spotlight lights, I realised that I could instead just use the same light view position and depth map for all three spotlights as it would not provide a big functional difference to the user when the spotlights are in the same orientation. However, to get these depth maps, I omitted passing in the street lamps since the lights were above the street lamps it wouldn't give the desired effect and would instead cast a shadow rather than giving off the illusion of it coming from the head of the street lamp. However, for the moonlight depth pass I did include these. So ultimately, for each one of these lights, the lightColour on a pixel was calculated of each light, and then the impact of all the lights added together with respect to the texture colour.

**Critical Analysis:** When I was creating the implementation of my shadows, a problem I had with initially the moonlight shadows, I found that my shadows were too black, or in other words, not factoring in the light in the spotlights and instead were just fully setting the pixel colour to black, this was because when there was a shadow I was initially not taking into account the ambient colour. When I implemented the ambient colour, it looked better and more like the output that I was aiming for, however I still didn't find it satisfactory as it wasn't taking into account that those pixels should still be affected by the spotlight and lit up accordingly. I also had a minor issue which I was quickly able to resolve where my shadow was not then showing up within the spotlight. Another issue I had was that the location of my depth map in comparison to my light was off, and it was calculating as if the model was where it was not, causing issues such as a spotlight shadow appearing where the model was not present and not appearing when it was present, I was able to fix this thankfully by changing the lightViewPos being used in the pixel shader, instead basing it off the main street lamp as the depth map was based off of that light. Another issue I had was the layering of shadows and improper calculations, unfortunately I do not have screenshots of this to show, but my shadows were not adding together properly and causing a realistic effect. Thankfully, as you can see in figure 18, my shadow overlaps came to what I intended them to be. I think, that ideally I would have liked to work more on the spotlights so that I could have increased the radius, which I could not figure out how to do without increasing the hight. This would have meant I could have added the lamp models to the depth shader for the spotlights and given off the effect of the lamp casting a shadow on itself. Being also able to have implemented more geometry such as that of a car model.

## POST PROCESS

Relevant Files:

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/texture_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/texture_vs.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/TextureShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/TextureShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/WaterDropShader_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/Shaders/WaterDropShader_vs.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/WaterDropShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-AlexanderPenny/blob/master/Coursework/Coursework/ManipulationShader.h

Relevant Screenshots and Diagrams:

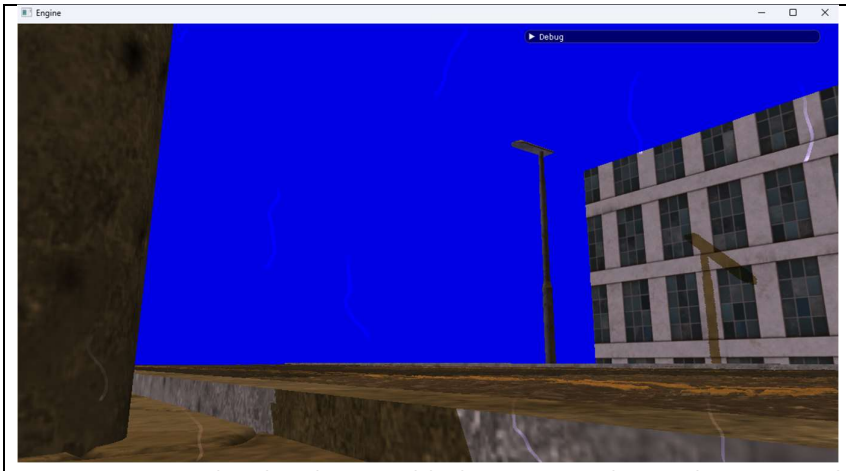



Figure 25: Rain droplets being added onto an orthoMesh texture which is being used to render everything to the screen.

Figure 26: Post processing being shown on a blank blue background.

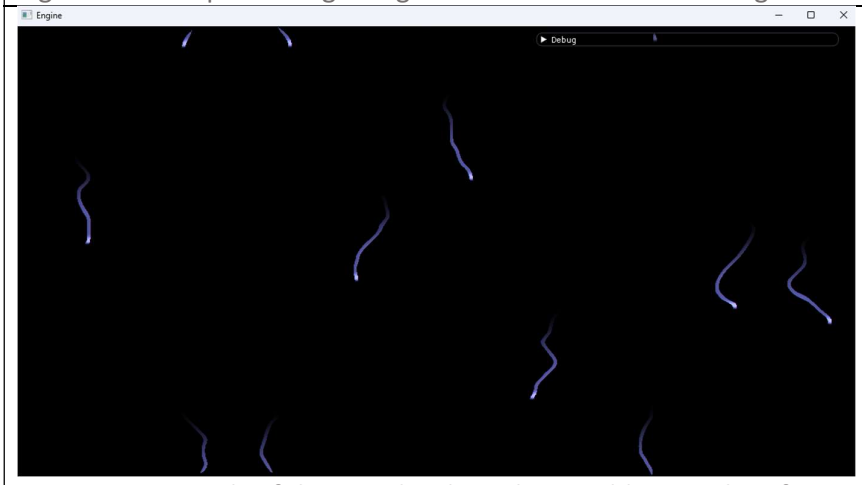


Figure 27: Normals of the rain droplets, this would mean that if it was applied to a texture in the world, light would be affected

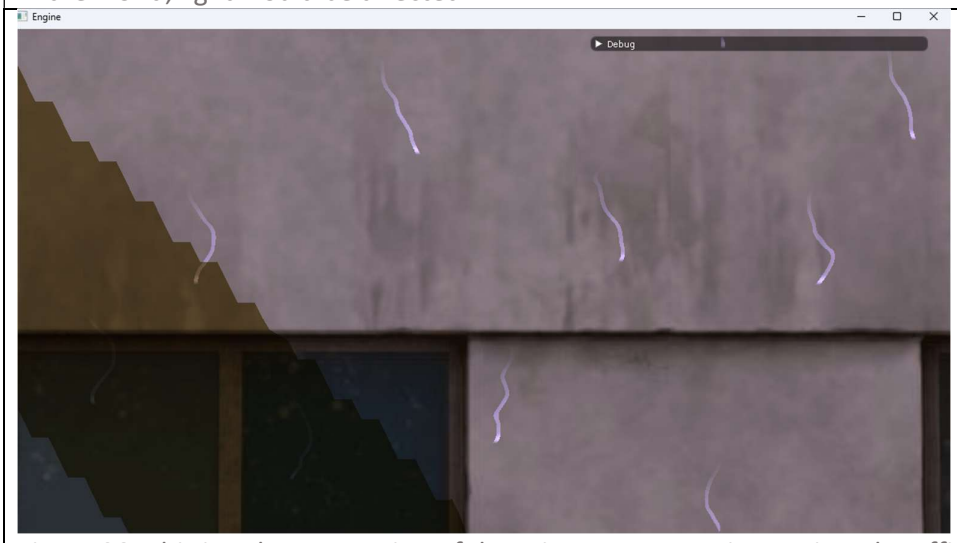


Figure 28: This is a demonstration of the rain post-processing against the office building face in the scene.

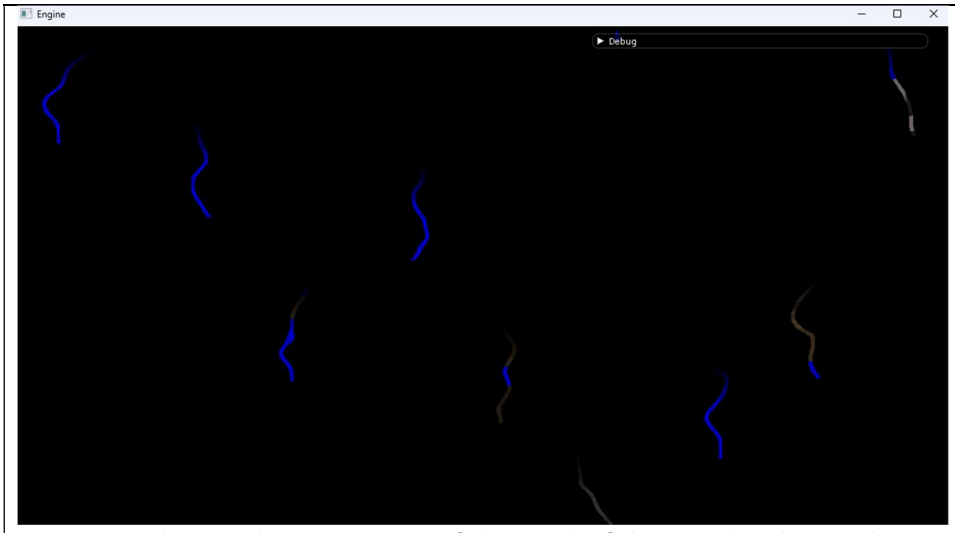


Figure 29: This is a demonstration of the mask of the rain droplets in the scene.

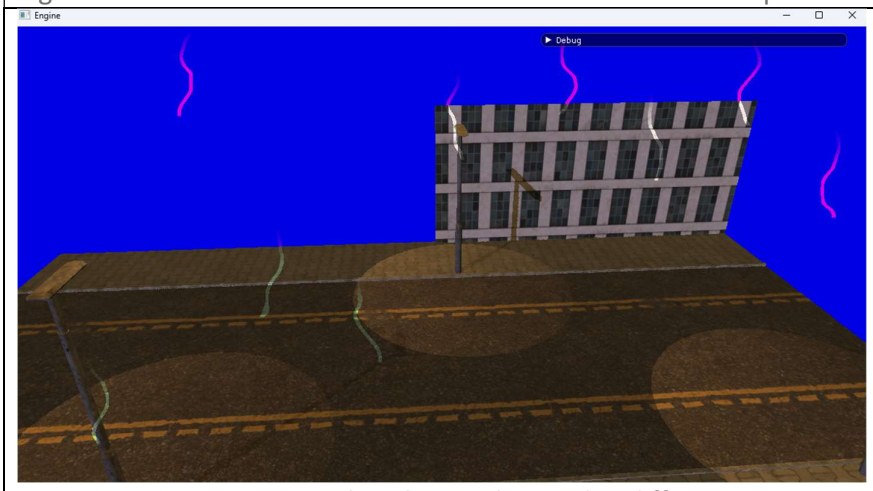


Figure 30: Demonstration that this can be used in different ways, to maybe even create some sort of alien rain by swizzling the values of the input.

**What was implemented:** I have created a rain post-processing effect, which can be toggled on and toggled off in the code by switching to an implementation with it or without it by editing the Boolean value in the render function. As you can see by the mask demonstrated in figure 29, I have created a water drop effect which, in theory could even be used to apply a different image of a swizzled set of values as demonstrated in figure 30. I have also applied normals to these rain droplets as you can see in figure 28, this would mean that if I were to apply this to an object in the world, the normals would be affected by it and therefore would reflect light in a different way. This mask is then applied to the capture of the scene before the post-processing in order to give this post processing effect as desired. This was the part of my project where my research was focused as the effect that I was wanting to create of water running down the screen, I proposed this idea in my project proposal and used the suggestion of a research source to then research how to do this. The research source that I have listed within the references, the YouTube video, which is completing the effect that I have implemented but instead uses Unreal Engine blueprints, I have used to convert into hlsl and to also instead of applying it to an object, apply it to the screen of the player. However, to be able to actually even apply the WaterDropShader, I had to output the whole scene to a TextureShader which I applied to an orthomesh which would then be

used to represent the player's screen by turning off the Zbuffer when I drew it to the screen. The video which I used as my research for the theory of how to be implemented, I used as a guide although there were some discrepancies in use, such as that they were applying this post-processing to an object in the world, so I had to keep that in mind when taking inspiration from their calculations. To apply this post processing, I have passed the data of the screen into the WaterDropShader, which I sample the colour from in order that my rain effect takes into account the colour behind it when calculating the effect.

**How was it implemented:** This was implemented by using a rain drip mask and a rain mask, the first of which was used to give the mask to the WaterDropShader in order to give the illusion of a water-like drop down the screen, the second of which was used in order to tell the shader at what time the raindrops should drop and what path that they should follow. My rain mask, which was taken from my research source, inside the alpha colour data, which is the brightness, I used to correlate to the offset of the rain, and in adding time (and time within a sin() function in order to increase its randomness) it gives the appearance of random droplets. Then using the blue channel, which correlates to the drip locations. I narrowed down the rain drip mask to specific lines, to give a visual "randomness" even if it follows a predictable path to the rain drips. Then using the calculate light function, although just multiplying the colour by the normal provides a similar output, I got the post processing of the scene as shown in figure 29, this was added to the scene view and then returned for being output to the render texture.

**Critical Analysis:** My normals for the water drops seem to be partially redundant in the example of me applying my normals to the screen as they do not affect the output, however, what I could have done is to sample the pixels to the left and right, a bit like the horizontal blur shader from a previous week does and then apply those pixels inside the water drop mask in order to give off a point of view of refraction. I could have also done with applying this possibly to the office face in order to create a water effect and demonstrate the effect of the normals on the plane. Although by using this method I was able to implement something that I found was more challenging than a depth of field where just measuring distances would be done.

## REFERENCES


- Manticorp (no date) Instructions. Available at: https://manticorp.github.io/unrealheightmap/instructions.html (Accessed: 21 October 2024).
- TextureCan (no date) Tile Texture (tiles_0112). Available at: https://www.texturecan.com/details/598/ (Accessed: 29 October 2024).
- TextureCan (no date) Concrete Texture (concrete0006). Available at: https://www.texturecan.com/details/58/ (Accessed: 30 October 2024).
- TextureCan (no date) Other Texture (others0029). Available at: https://www.texturecan.com/details/563/ (Accessed: 31 October 2024).

- TextureCan (no date) Asphalt Texture (asphalt_0004). Available at: https://www.texturecan.com/details/289/ (Accessed: 31 October 2024).
- TurboSquid (2024) Street Lamp Model (StreetLamp.obj). Available at: https://www.turbosquid.com/3d-models/3d-street-lamp-2210777 (Accessed: 20 November 2024).
- TurboSquid (2024) Police Car Model (car.obj). Available at: https://www.turbosquid.com/3d-models/3d-police-car-2260107 (Accessed: 29 November 2024).
- Ben Cloward (2020) Rain Drip Texture (rain_drips.png). Available at: https://www.bencloward.com/youtube/rain_drips.tga (Accessed: 3 December 2024).
- Ben Cloward (2020) Rain Drip Mask Texture (rain_drip_mask.png). Available at: https://www.bencloward.com/youtube/rain_drip_mask.tga (Accessed: 3 December 2024).
- Ben Cloward (2020) *Rain Drip Shader - UE4 Materials 101 - Episode 15*. Available at: https://youtu.be/rhlV4okjkiA?si=kPPGtcuOccSihVBB (Accessed: 12 December 2024).