CMP 301 Report

For this coursework submission it was required for the student to build a 3D graphics application which generated a single scene to demonstrate various HLSL shaders. These HLSL shaders had to be capable of showing Vertex Manipulation, Post Processing, Lighting, and Shadows with Tessellation and use of a geometry shader as optional. To meet this brief the student chose to take inspiration from the Dragon Ball series and build the scene based on Master Roshi’s island. This scene was chosen as it had all the elements necessary to meet the brief.

The Vertex Manipulation was covered in two distinct ways. Firstly, to create the Island itself a flat plane mesh was used. A greyscale height map was used in with a Vertex Manipulation shader to move the vertices along the Y axis in accordance with the height map. This means that the height map made the plane change shape and make it look like a realistic island with bumps and dips in it. Secondly a different way Vertex Manipulation was used was to create the waves This was done by using a different height Map on a flat plane mesh to give it the shape of waves much like the previous example. However, the wave Vertex Manipulation differs in that the height map texture moves every frame, this creates the illusion of waves rising and falling as the heightmap just loops creating a seamless pattern of waves.

Lighting and shadows was covered by using three different types of light. The first light was a directional light, this light had no attenuation values and so was used to light the whole scene from the left side. This first light was intended to represent the sun as it is a light source from such a great distance that for a scene this small an endless light in one direction was an appropriate stand in. The second light used was a point light. The point light used in the scene was actually constructed out of 6 directional lights with attenuation. This means the light shines out in all directions but the attenuation prevents it from lighting the entire scene. Doing both of these creates a small realistic light much like a lightbulb. The final light added to the scene was a spot light, A spot light is similar to a direction light however instead of lighting everything in one direction it lights everything within a circle down that direction. This light moved position and direction with the camera to simulate carrying a torch.

The Post Processing Technique used was a depth of field blur. This type of blur was done using two render textures, one of the scene blurred and one that was unblurred. The program would then blur and unblur the scene depending on what the user pointed the camera at and how far away from the camera it was.

Additional GUI elements added were limited to manipulating the lights for example, turning them on and off.

Vertex Manipulation

The Vertex Manipulation of the island begins in the “app1.h” and “app1.cpp” files. It is here that the textures and mesh used for vertex manipulation are loaded including the plane mesh, the sand texture and the height map. This data is then sent to the “Manipulation Shader.h” and “Manipulation Shader,cpp” files via the set shader parameters function. Here a sampler is created and passed along with the Heightmap texture and the plane mesh to the vertex shader used for vertex Manipulation “ManipulationVS.HLSL”. Here each vertex of the plane mesh is compared to an equivalent point on the heightmap. This point on the heightmap is passed into the sampler which returns a value depending on how black or white the point is. This value is used to determine the mesh vertex’s position on the Y axis. The dark the colour the lower the value and the lighter the colour the higher the value. Therefore, the white parts of the height map become the highest points on the plane mesh. The data is then passed on to the Pixel shader file for lighting calculations covered later. Using this technique with the height map the student created results in realistically shaped terrain. The normals of the mesh were also recalculated in the vertex shader however that will be covered in lighting. The water rendered in the scene was done in a very similar way however it required the creation of a time variable and use of the timer class. “Times” is a float variable that is updated every frame by adding delta time, this allows time to pass at the same rate regardless of a change in frame rate. A sampler similar to one previously mentioned is created here as well, this sample is set to “wrap” textures, this means if the coordinate of a point in the texture exceeds one the texture loops. The “Times “ variable along with the water texture and the water heightmap are all passed over to the “WaterShader.h” and “WaterShader.cpp” files where the times variable is placed in the time buffer which contains the time float variable and a float3 padding variable. The padding is necessary as buffers require the use of multiples of four bytes. The time buffer, the two textures and the mesh are pass into the vertex shader. In the vertex shader the sampler is used again however this time the coordinate of the height map mapped to the mesh is offset on the Z axis. This offset is multiplied by the time variable so the heightmap moves, this combined with the sampler looping the texture creates a seamless pattern of waves being generated simulating the movement of the sea. The normal for the manipulated mesh also had to be recalculated and this will be discussed under lighting and shadows. The data for the mesh are then passed into the Pixel shader for lighting calculations to be used.

Lighting and shadows

First how the normal for the manipulated geometry will be covered. The normal were calculated using the example shown in the Frank Luna Direct X11 book (1). In this method two vectors are created which overlap at the point the normal is being calculated. Both of these vectors are normalised, then the normal is calculated from the normalised cross product of both vectors. The student used this technique for both the shader used on the water and the shader used on the island geometry. The Student began adding light with a single direction light casting light diagonally down the Y axis and diagonally across the Z axis with direction values (0.0, -0.7, 0.7) and a diffuse value of (1.0, 1.0, 1.0, 1.0) to be a bright white light and simulate the sun. this light was then passed into the three different shaders to light the scene. The next step taken was to add shadows to the scene. For this purpose, it was necessary to add 3 more shaders for the depth pass. Each of these shaders had to apply the same vertex manipulations to the geometry in order for the depth pass to work. These depth shaders were used to render the scene from the lights perspective and create a black and white image to demonstrate where light would be unable to illuminate. The first made was the depth shader which made no changes to geometry and just rendered what It was given. This was use to render the house during the depth pass. Two more depth shaders were created which applied the same vertex manipulations discussed previously to render the sea and island. New “.cpp” “.h” and vertex shaders were made for this purpose however they all share the same pixel shader file as no modifications for the geometry were needed for that. Then the shaders created for drawing the geometry were modified to improve the lighting. The additions to these shaders were very similar to the example in the shadow’s lab where the black and white depth map texture created during the depth pass is passed into the three shaders to the shared pixel shader where it is then calculated if the light can be cast on the point using the pixel shader of if there is geometry between the light and the point being rendered. After shadows was working the student then built on the lighting by adding another type of light. A point light. A point light renders only a limited area of geometry around a single point, to do this six lights are made which point up and down each axis both positively and negatively. Each of these lights don’t light the entire scene like a direction light because of two factors, a directional light uses an orthogonal matrix for its depth pass which means it has a perspective which can see the entire rendered scene in contrast to this point lights use a projection matrix for the depth pass which renders the scene from a point of view based on a position and a direction.

1. https://files.xray-engine.org/boox/3d\_game\_programming\_with\_DirectX11.pdf