CMP301- Graphics programming

Overview of the scene

When under taking the course the scene was designed to meet the briefs must completes as well as under taking the additional work as to improve the achievable grade, the brief state that the scene must have contain an example of vertex manipulation which is correctly lit and textured, next is a post processing techniques and finally the scene must contain lighting and shadows. The brief states that an example of tessellation and use of the geometry shader can be demonstrated to improve the grade of the coursework.
Each section relates to the above mentioned techniques.

Vertex Manipulation

The coursework contains two examples of vertex manipulation, the first being the ground which is plane takes in two textures, the first being a displacement map which is used to set the Y-position of the vertex in turn creating and island scene.
The second example is the water waves which is a plane that algorithmically manipulated with data that the user can manually set with UI elements, the water is around 50% transparent so as to the ground below it also referred to as the sea bed.

Post Process

The course work requires that a post process technique be demonstrated this is achieved by the post process bloom being added to the scene. Bloom is a post process that increases the brightness of colours that fall within a set range in turn creating bright spots this is achieved by taking a clean screen shot of the scene and creating a new screenshot but with selected colours mentioned above alone. This screen shot is blurred both horizontally and vertically this creates a ghostly affect when the two are combined.

Lighting and Shadows

To cover the lighting and shadow aspect of the brief the coursework was developed to have 3 lights in which each are used to develop its own depth map of the ground, water and grass objects in the scene. All three lights and depth maps are passed to all the water and ground plane which then calculate the lights intensity within certain spots of scene and casting shadows on the remaining areas.

Tessellation

Tessellation was one of the additional demonstrations that could have been used to improve the achievable grade for the coursework, this was achieved by applying tessellation to both the ground and water plane, each of which has reference to the camera and will activily tessellate based on how close the camera is to parts of the quad plane this will be referred to “Level of Detail” or “LOD” for short.

Geometry Shader

The use of the geometry shader was the last task which was additional to improve the achievable grade of the coursework, for this the geometry shader was used to generate grass for the plane. So due to how the geometry shader works, it gets a vertex position and draws a shape a that point it can be provided a plane of vertices and will draw a shape at each vertex this will draw a flat plane of grass, so make it fit the scene a height map must also be provide so that the grass climbs the islands.

Additional UI

To allow customisation of the scene and properly test different examples to see that they work properly and that they work in real time, additional UI controls are provided to the user so they can

control different aspects of the scene, such as the position and colour of lights, to exact elements that control the generation of waves on the water plane.
Bellow will state what controls the user has active to as well as brief description as to what it does.

Toggles
The grass can be toggled on and off.
The water can be toggled on and off.
Bloom can be toggled on and off.

Grass
Speed of the grass can be adjusted to make I move faster.
Limit of the grass can be adjusted to limit how far it can stretch.
Lower limit of grass can be adjusted to limit how low the grass will render.
Upper limit of grass can be adjusted to limit how high the grass will render.

Water
Amplitude of water can be adjusted to decrease or increase the hight between 0 to the crest of the wave.
Length of water can be adjusted to decrease or increase the distance between two wave crests.
Speed of water can be adjusted to decrease or increase the time it take for peak of the wave to travel the total length of one crest.

Lights
The position of the lights can be adjusted on all three lights
The diffuse(Colour) can be adjusted on all three lights.
The depth map for each light can be toggled on and off.
Only the third light can have its direction changed.

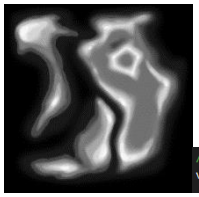
Bloom
Bloom can be toggle on and off.
The amount removed from the colours can be decreased or increased.
The threshold of the colour remove section of the bloom post process can be set.

Detailed explanation of the techniques demonstrated

Below each techniques will be broken down into sections so as to allow a deep dive on each component without losing a readable structure. Each section will relate back to the must completes and the additional examples of the brief.

Ground Island

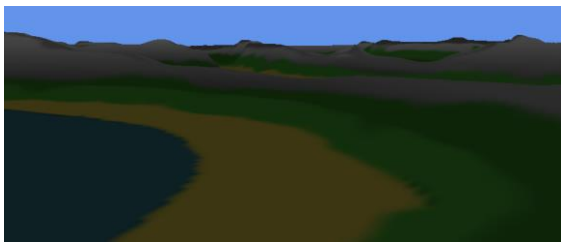
Vertex Manipulation
The islands on the ground are formed through using a displacement map, this method uses a image which is loaded into the program and is sampled with the texture coordinates of the quad plane, this will generate a float4 colour value at that particular vertex this can then be multiplied with a pre set value or a user set value, this particular program uses a pre set value.



```
//Sets vertex y position based on the textures colour value
vertexPosition.y = texture0.SampleLevel(sampler0, output.tex, 0) * 4;
```

For vertex manipulation using this technique the normal of each vertex need to also be calculated as not every vertex the normal is pointing directly up. This is done by using the above method but instead of using the value to set the y coordinate its stored in a local variable. Each value is calculated at each surrounding vertex, with theses values we can calculate the tangent and bitangent of the normal and by getting the cross product of these values the normal for each vertex is generated. In a normal plane this is handled within the vertex shader but in this case its handled within the domain shader.

```
float cellSpcae = 1.f / 100.f;

//Calculate vetex postions
float orgin = ((texture0.SampleLevel(sampler0, output.tex, 0.f))).r;
float up = ((texture0.SampleLevel(sampler0, output.tex + float2(0.f, -cellSpcae), 0.f)));
float down = ((texture0.SampleLevel(sampler0, output.tex + float2(0.f, cellSpcae), 0.f)));
float left = ((texture0.SampleLevel(sampler0, output.tex + float2(-cellSpcae, 0.f), 0.f)));
float right = ((texture0.SampleLevel(sampler0, output.tex + float2(cellSpcae, 0.f), 0.f)));

//Get tangeent and bitangents
float3 tangent = normalize(float3(2 * cellSpcae, (right - left) * 3, 0.f));
float3 bitangent = normalize(float3(0.f, (down - up) * 3, (-2.f) * cellSpcae));

//Get the cross product
float3 temp = cross(tangent, bitangent);
//Times by worldmatrix and return
temp = mul(temp, (float3x3) worldMatrix);
return normalize(temp);
```



When rendering a normal plane with a displacement map the texture coordinate stay the same but when the plane is tessellated the texture coordinates need to be properly calculated so as to be properly textured as this section talks about texturing and not tessellation, the process to calculate texture coordinates will be discussed further down.
A second texture is passed to the pixel shader this is the texture that will be applied to the plane so again by sampling the texture the program will generate a pixel colour value this is then stored and then used to generate a colour with the 3 lights colour.

<u>Lighting and shadowing</u>
This section doesn't cover the generation of depth maps although it does mention the use of depth maps, the generation of these depth maps will be discussed further down the document.
Before moving into the pixel shader the correct light view matrix for each light must be calculate so that shadows can be properly rendered this is handled within the domain shader. This is calculated by first multiplying the vertex position with the world matrix, this product is then multiplied by the light view matrix followed by light projection matrix.

VP = vertex Position
WM = World Matrix
LVM = light view matrix
LPM = Light projection Martix

$$LightViewMatix = ((VP \cdot WM) \cdot LVM) \cdot LPM$$

These three light view matrices are then sent through to the pixel shader, within the pixel shader these matrices are used first to make projected texture coordinates and then later on to check if a point lies within a shadow.
The if statement that checks wither the pixel is not within shadow is part of the original shadow code that was provided in one of the labs this is marked in the code to state who originally wrote. Within in each statement is a light calculating function. The first two are point lights with attenuation being applied with the third being normal point light.

```
// Calculate lighting intensity based on direction and normal. Combine with light colour.
float4 calculateLightingAtten(float3 lightDirection, inputType input, float4 ldiffuse, float3 position)
{

    float intensity = saturate(dot(input.normal, lightDirection));
    float4 colour = saturate(ldiffuse * intensity);

    float d = length(position - input.worldPosition);
    float c = 0.5f;
    float l = 0.125f;
    float q = 0.0f;
    float attValue = 1 / ((c + (l * d)) + (q * pow(d, 2)));

    colour *= attValue;
    colour = saturate(colour);
    return colour;
}
```

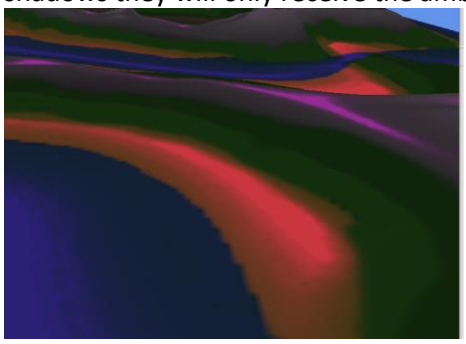
Above is a snipped of code that is used to calculate the lighting colour at each pixel. This code first calculates the intensity by getting the dot product of the normal provided and the lights direction(This is calculated by getting the normalisation of the lights position – world Position). To calculate the base colour the diffuse of the light is multiplied by the intensity and making sure It

stays within 0-1 so as not be instantly turned to bright white. The attenuation value is then calculated and applied to the colour of the pixel calculated above. The formula in which is used is shown below.

CF = Constant Factor
LF = Linear Factor
D = Distance
QF = Quadratic factor

$$Attenuation = \frac{1}{CF + LF \cdot D + QF \cdot D^2}$$

Each new calculated colour is store in a empty local variable this means the three colours that have been generated can now be added together along with the ambient, so if there are areas that are in shadows they will only receive the ambient colour and the texture colour



Tessellation

The tessellation of the plane is handled within the hull shader, for this to work the cameras position is pushed in to the hulls constant buffer. From here the average position of the four patches, this is then split down into a float 2 to capture only the X and Z position the height of the camera doesn't factor into the LOD calculation.

The distance is calculated by simply taking the cameras X and Z position away from the average position of the patches, this is then halved to reduce the distance and is check to make sure it's a positive value if not then its made positive. This value is then used to divide a value this means the lower the distance the higher the tessellation factor the higher the distance value the lower the tessellation value.

```
//take average input
float3 patchAdv = (float3)0;
for (int i = 0; i < 4; i++)
{
    patchAdv += inputPatch[i].position;
}
patchAdv /= 4;
//average the float3 into a float 2
float2 pos = float2(patchAdv.x, patchAdv.z);

//Find distance from vertex to camera
float2 distance = (pos - camPos.xz);
//find distance average
float distanceAdv = (distance.x + distance.y /*+ distance.z*/) / 2;
distanceAdv += 1/2;

//Makes distance positive
if (distanceAdv < 0)
{
    distanceAdv *= -1;
}

//Calculate tesseleation factor so that the smaller the distance the higher the facter and clamp between 1 and 5
float tessellationFactor = clamp(100 / distanceAdv, 1, 5);
```

Water Waves

Vertex manipulation

The waves on the water plane are formed by algorithmically, this method takes in pre-set or user set values which can be past to the domain shader through a constant buffer that will be filled into an algorithm. This algorithm comes from the "NVIDIA Effective Water Simulation" and a copy of the formula is shown below.

$$W_i(x, y, t) = A_i \times \sin\left(\mathbf{D}_i \cdot (x, y) \times w_i + t \times \varphi_i\right).$$

The above formula is slightly altered in terms of what the result is used for this, in terms of this program the algorithm is used to generate the Y position of the vertex. Below is a snippet of code that is used within the course work.

```
//Sets values needed to generate Waves Y position
  float l = length;   //length
  float w = 2 / l;    //Width
  float a = amper;    //Ampplatude
  float s = speed;    //Speed
  float d = distance; //Distance between waves
  float AE = s * w;
  float currentTime = time;

  //gets dot product of the xz of the vertex and the disatance
  float DotProd = dot(d, vertexPosition.xz);
  //calculates the Y position
  vertexPosition.y = (a * sin((DotProd * w) + (currentTime * AE)) + 0.75);
```

For calculating the normal of these vertices follow a similar formula but instead of sin being used within the algorithm its switched out for -cos. Below is a snippet of that formula using the exact same variables as shown above.

```
output.norm = float3(a * -cos((DotProd * w) + (currentTime * AE)) + 0.75, 1, 0);
```

Lighting and shadows

The lighting and shadowing of the water plane is the exact same technique as shown on the ground plane.

Tessellation

The tessellation of the plane follows the exact same LOD techniques as mentioned in the ground plane further up the document.

Grass Plane

for building the grass, the shader is passed a plane with the resolution of 800*800 this is because the geometry shader renders a pre-defined shape at every vertex its provided, this means a 800*800 plane reduced down to the dimensions of 100*100 this means the vertex density is going to be 8 times larger than a standard plane meaning the grass will be close to each other.

Geometry Shader

With in the geometry shader, this position, height and the rotation the blades move at. First the rotation of the X position is calculated, this is a simple sin calculation, this allows the blade of grass to move similar to being blown in the wind. Below is a copy of mentioned formula.

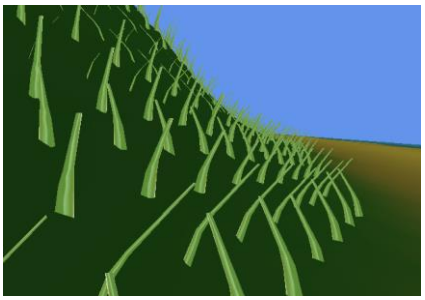
$$rotation = \frac{sin(time \cdot speed \cdot rand(0))}{limit}$$

The next rotation that's calculated is for the blade of grass Z position to make it a little less static within the scene, this formula is the reverse of the above shown formula, this is a modified cos sign wave.

$$rotation = \frac{cos(time \cdot speed \cdot rand(0))}{limit}$$

These rotation values are then incorporated into the vertex position along with a few other values which set the shape of a blade of grass.

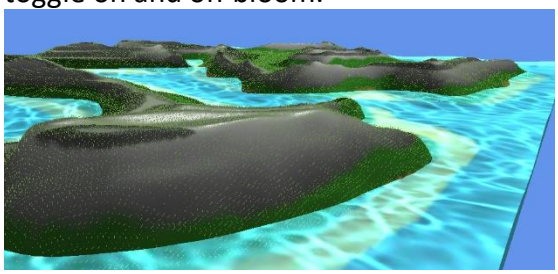
```
float height = 0.2;
float hHeight = height / 2;
float bWidth = 0.04;
float m1Width = 0.03;
float m2Width = 0.02;
float tWidth =  0.01;
float hWidth = bWidth / 2;

float3 g_positions[8] =
{
    float3(-bWidth / 2, start, 0),
    float3(bWidth / 2, start, 0),
    float3(-m1Width / 2 + sitx / 3, start + height / 3, cos(time * speed) / (limit * 6)),
    float3(m1Width / 2 + sitx / 3, start + height / 3, cos(time * speed) / (limit * 6)),
    float3(-m2Width / 2 + sitx / 2, start + height / 2, cos(time * speed) / (limit * 3)),
    float3(m2Width / 2 + sitx / 2, start + height / 2, cos(time * speed) / (limit * 3)),
    float3(-tWidth / 2 + sitx, start + height, cos(time * speed) / (limit * 1)),
    float3(tWidth / 2 + sitx, start + height, cos(time * speed) / (limit * 1))
};
```

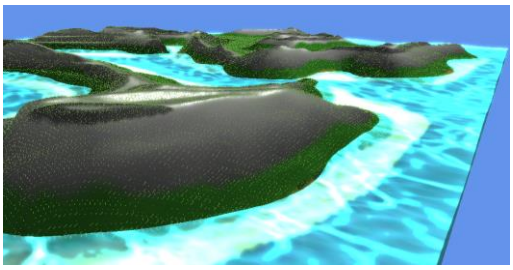


When each vetex is being generated by multiply it with the World, View and project matrix it is also give a rotaional value, this is done to make sure each blade of grass is facing a different direction so that it look far more relistic than it all facing the same direction. The function that is used to generate this rotation is a snipped of code found online. This has been refferenced[4].

Bloom

The bloom postprocess is first started by taking a clean render to texture of the entire scene will all lights and shadows generated. This texture is then duplicated and passed through to the colour remover shader, this is a simple shader the vertex shader passes default position and texture coordinates in which the pixel shader uses to generate the image. From here the background colour is removed so as to not affect the background colour when it comes to reapplying the decreased colour to the original. The remaining colours are then reduced by a set amount, the colour value is then checked to find the brightest values within a pre set range and these colours are kept and the remaining colours are then set to black.
This new render to texture is then passed through a vertical and horizontal shader these shaders each check 4 neighbours either side of the processed pixel this blends each colours so that when the blurred colour is combined with the original render to texture the brightest colours will be increased and will have a glow effect. As bloom is calculated after the entire scene is rendered with lights and shadow, this means the user by choice can toggle on and off bloom.





Lights

Depth Maps

For each light within the scene a depth map must be generated so that shadows can be properly generated, this is done by rendering each object of the scene by passing them to the depth shader, this depth shader generates an image which shows the depths in the form of a colour value this means it can be passed to the another pixel shader to be used to generate the shadows of each light.
For every light we much generate a depth map so 3 depth maps are created.
The depth shader has been specifically designed to accommodate each object, this is to reduce the file count of the of the course work.

```
DepthShader::DepthShader(ID3D11Device* device, HWND hwnd, int typeIndex) : BaseShader(device, hwnd)
{
    //Check what hlsl files we need
    type = typeIndex;
    switch (type)
    {
    case 0: //Ground files
        initShader(L"depth_vs.cso", L"depth_hs.cso", L"depthGround_ds.cso", L"depth_ps.cso");
        break;
    case 1: //Water files
        initShader(L"depth_vs.cso", L"depth_hs.cso", L"depthWater_ds.cso", L"depth_ps.cso");
        break;
    case 2: //grass files
        initShader(L"depth_vs.cso", L"depthGrass_gs.cso", L"depth_ps.cso");
        break;
    }
}
```

During the initialisation of the depth shader an identifier is set so that when each item is rendered the correct hlsl files are used.

Critical reflection of the application

Looking back on the process of completing the course work there are areas that could be improved upon as well as there things that went extremely well and aided in the completion of the course work.

Negative

During the proposal of the course work the first flaw was thinking that a program that had a lot of items was going to grab a high grade but after receiving feedback on this, it was discussed that I would be better to restructure and think about producing a really good example of each technique instead of having a few not so good features. Although this isn't a project breaking flaw it caused a change of thinking when it comes to thinking about planning out of a project, the main thing being not to underestimate a project based on the brief and to properly think it out so that in cases where there wont be feedback on the proposal, its good to get it right from the start.

A lot of time during the production of the course work was spend researching on different techniques and solutions to problems, thinking forward to future projects this could have been better handled. More time could have been spent at the beginning of the project researching and testing, this means when it comes to implementation less time during development will spent of research allowing for more refinement and polishing of the final product.

Although the coursework is fully completed and meets all must completes as well as the additional task, there is multiple areas which could have been further polished even thought it might not improve the grade by itself it would help other areas get an improved grade, a good example of this would be to have a 4^{th} light but this would be a more complex light such a spotlight with better attenuation and tied to something that constantly moving such as the camera, this would better show shadows fully working within the scene.

Positive

During the planning of the coursework, highest priority was to start early so as too allow the most possible time for development and any hitches which might cause a time sink, this took some pressure off as there wasn't any point in which there felt like there wasn't enough time. This is something that was a problem in previous projects and now having proof it will be something that will be continued going forward.

During the production of the course work submission one thing that made a massive difference was the fact that during the lab session a lot of additional time was spent further developing on tasks that was given, One example of this was tessellating a plane and modifying it to work with displacement maps, this allowed for a faster implementation allowing more time to be spent in other areas of the project as well as additional projects separate to this one. This type of example is not always a common thing, so the natural take away is before undertaking the task would be spend

time on building example of potentially implementable objects or the time working on a particular API if that's what the project requires.

Conclusion

Looking back at the project, overall it went very well and didn't stray to far away from the original concept of the project, even though the negatives mentioned did cause time being spent unnecessarily on wrong items, they did show areas on which could be improved on. With positives showing what went well and would be extremally beneficial if it were to be carried on into future projects.

References

2021. [image] Available at: <https://icons8.com/wp-content/uploads/2020/07/water-texture-sun-in-water.jpg> [Accessed 1 January 2021].

2021. *Islands Of The Sentinel*. [image] Available at: <https://www.deviantart.com/jacelaughingwolf/art/Islands-of-the-Sentinel-662281123> [Accessed 1 January 2021].

anon, 2021. [image].

Finch, M., 2021. *Chapter 1. Effective Water Simulation From Physical Models*. [online] NVIDIA Developer. Available at: <https://developer.nvidia.com/gpugems/gpugems/part-i-natural-effects/chapter-1-effective-water-simulation-physical-models> [Accessed 1 January 2021].

Stack Overflow. 2021. *Rotate Quad Made In Geometry Shader*. [online] Available at: <https://stackoverflow.com/questions/28124548/rotate-quad-made-in-geometry-shader> [Accessed 1 January 2021].