COSC422 Harry Dobbs Assignment 1

Outline:

Bezier Surface Modelling: I have developed a Bezier Surface Model of an Elephant (Gumbo) . The bezier surface is generated with 4x4 Bezier patches as the inputs. This model incorporates specular,ambient and diffuse lighting. Depending on the camera view position you will get a different tesselation level. The tesselation level is lower when you are further away from the model, and increases as you get closer. The model is able to play an animation sequence which explodes the bezier surfcae model.

The main problem I faced with this part of the assignment were:

1. After the explosion effect the fragments would partly sit below the ground plane. This was solved by only updating the y-coordinate of the fragment when all vertices inside that fragment will remain above the ground plane.

Terrain Surface Modelling: I implemented a terrain mapping program that allows the user to choose between two height maps. The maps contain dynamic levels of detail in which the tesselation decreases as the camera moves further away and increases as the camera gets closer to the terrain. The terrain is rendered with ambient and diffuse lighting. Four textures are included in this terrain (Water, Grass , Rock and Snow) . Additionally this program prevents cracking that can occur between different tesselation levels. This program also includes adjustable snow and water levels.

The main problems I faced with this part of the assignment were:

1. I was incorrectly using the geometry shader to output colour values of those textures at each point as opposed to using the geometry shader to output the coordinates that would be used by the fragment shader to map the texture onto the triangle. This was solved by using the geometry shader to pass cooridnates rather than colour values.
2. The other problem I faced was to solve cracking. Cracking occurs when neighbouring patches have different tessellation levels. To solve this I used the corners of each patch to work out the tessellation level for that edge. This meant that neighbouring patches would have the same tessellation level as they share the same edge.

Gumbo Patches Model

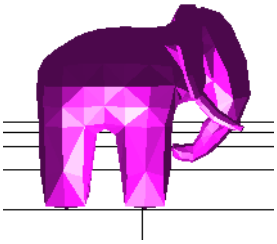

Figure 2: Gumbo being displayed with minimum tessellation level

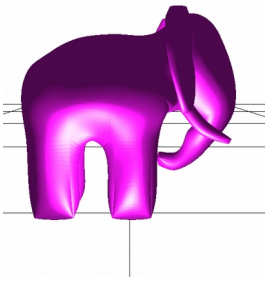

Figure 1: Gumbo being displayed with maximum tessellation level

The tessellation levels of Gumbo vary dependant on how far away the camera is from the object. Firgure 2 shows gumbo when the viewer is close whereas Figure 1 shows gumbo when the viewer is further away. The dynamic level of tessellation levels was implemented in the control shader (Figure 3). The code written in Figure 3 was based off an equation found in the lecture slides (Figure 4).

```
distanceMin = -30;
distanceMax = 200;

Lhigh = 50;
Llow = 2;

tesselationLevel = ((distance - distanceMin)/(distanceMax - distanceMin) * (Llow - Lhigh) + Lhigh);
```

Figure 3: Implementation of Dynamic Tessellation Levels

$$L = \left(\frac{d - d_{\min}}{d_{\max} - d_{\min}} \right) \left(L_{low} - L_{High} \right) + L_{High}$$

Figure 4: Equation from lecture slides used to implement Dynamic Level of Detail [1]

To explode the mesh I did the following steps.

1. Take the normal vector of the centre of the patch (Elephant is centrered around origin)
2. Using the normal vector, I am able to derive the amount to increase the X,Y and Z coordinates by every period.
3. The X and Z coordinates keep on increasing until the Y coordinate is less than or equal to 0 at which point the animation stops. The Y coordinate is effected by gravity. The equations used are shown in figure 5. The time is passed into the shaders as a unifrom variable.

$$P_y = P_y + v_y t - \frac{1}{2} g t^2$$

$$P_{xz} = P_{xz} + d_{xz}\, v_h t$$

Figure 7: Equations used for projectile motion. [1]

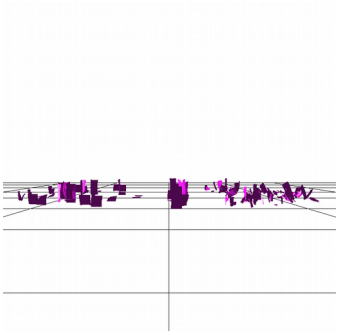

Figure 6: Gumbo Post Explosion

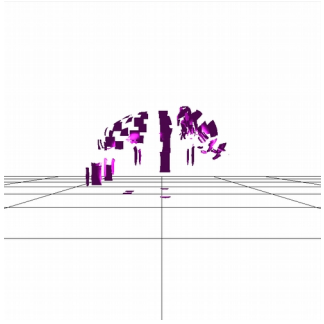

Figure 5: Gumbo Mid Explosion

Terrain Model

The terrain model implements three forms of animation: dynamic level of detail , adjustable snow level and adjustable water level.

Dynamic Level of Detail:

The terrain model uses the camera view position to work out the tessellation levels of each patch. The tessellation levels around each edge of a patch are not the same. The tessellation level of each edge is set uniquely dependant on their distance. This allows for cracking to be prevented as neighbouring patch edges will always have the same level of tessellation. The dynamic level of detail has been set to be aggressive for illustration purposes.

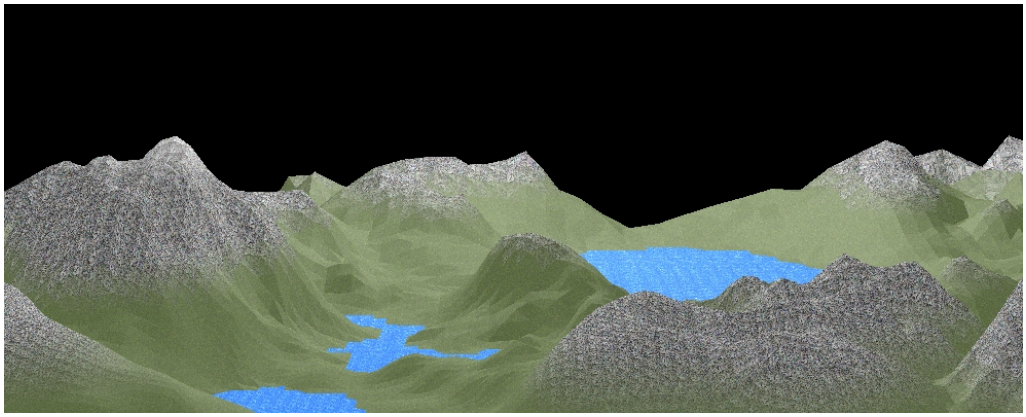

Figure 8: Map 1 With Minimum Water and Maximum Snow

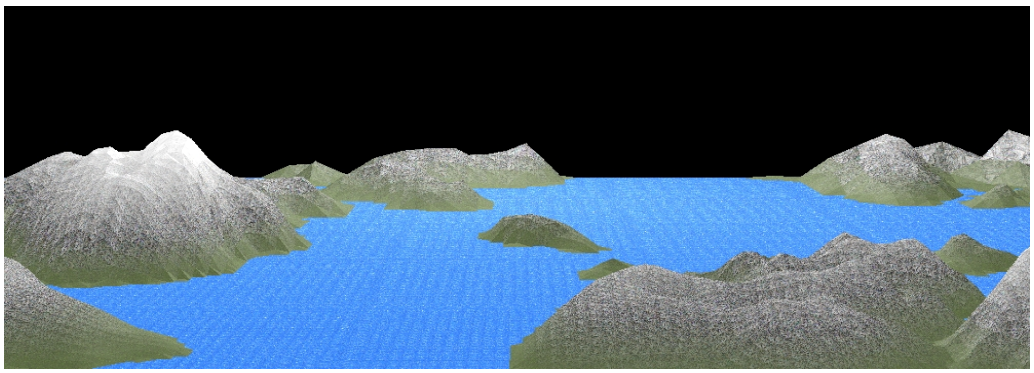

Figure 9: Figure 9: Map 1 with Maximum Water and Maximum Snow

Adjustable Snow Level:

Adjustable snow level has been implemented. To allow for realistic snow rendering there is linear interpolation between the snow and the rock. This means the change between rock and snow occurs gradually. To implement an adjustable snow level,the point in which the interpolation starts needs to change as the snow level decreases. Each texture is given a weight dependant on how much of the particular texture you want to display. These weights are calculated in the Geometry shader and then utilized in the fragment shader to mix the textures.

```
if (fragCentre.y <= snowLevel && fragCentre.y > snowLevel - 3.5)
{
    rockWeight = 1 - (fragCentre.y-(snowLevel - 3.5))/(snowLevel-(snowLevel - 3.5));
    snowWeight = 1 - (1 - (fragCentre.y-(snowLevel - 3.5))/(snowLevel-(snowLevel - 3.5)));
}
if (fragCentre.y > snowLevel)
{
    snowWeight = 1;
}
```

Figure 10: Code from Geometry Shader used to set the weights of each texture

```
 outColor = (texture2D(water,textcoord) * waterWeight) + (texture2D(grass,textcoord) * grassWeight)
+ (texture2D(rock,textcoord) * rockWeight) + (texture2D(snow,textcoord) * snowWeight)+(oColour * 0.30) ;
```

Figure 11: Code extract from fragment shader used to mix the textures together

Adjustable Water Level:

Adjustable water level has been implemented. To implement the water level correctly the water surface must be flat. This requires all the points below the desired water level to become equal to the water level. This is implemented inside the evaluation shader. This will create a flat surface.

```
posn.y = color.y * 10;

if(posn.y <  waterLevel)
{
    posn.y = waterLevel;
}
```

Figure 12: Code snippet from evaluation shader used to flatten the surface.

Controls

Button	Description
Up Arrow	Move Forwards in the Scene
Down Arrow	Move Backwards in the Scene
w	Toggle Wire Mesh
Space Bar	Execute Explosion
Right Arrow	Turn Right
Left Arrow	Turn left
1	Select Map 1
2	Select Map 2
o	Increase Water Level
p	Decrease Water level
l	Increase Snow
k	Decrease Snow

Yellow: Only applies to the patches program. Green: Only applies to terrain program.

References:

Dr. R. Mukundan . (2019) . Computer Graphics COSC422 [PowerPoint Slides]. Retrieved from https://learn.canterbury.ac.nz/mod/resource/view.php?id=803416 [1]

Grass Texture: Surfacecurve. (2014, July 05). Grass Texture,seamless 2d. Retrieved from https://opengameart.org/content/grass-textureseamless-2d [2]

Water Texture: Qubodup. (2012, March 19). 3 live proceduraly generated tiling water textures (512px, running brushes). Retrieved from https://opengameart.org/content/3-live-proceduraly-generated-tiling-water-textures-512px-running-brushes [3]

Rock Texture: Sindwiller. (2011, July 09). Generic rock texture. Retrieved from https://opengameart.org/content/generic-rock-texture [4]

Snow Texture: Downdate. (2012, December 07). Snow Texture - snow_2.png. Retrieved from **https://opengameart.org/node/13587** [5]

Height Map: Heightmap. (2018, October 04). Retrieved from https://en.wikipedia.org/wiki/Heightmap [6]