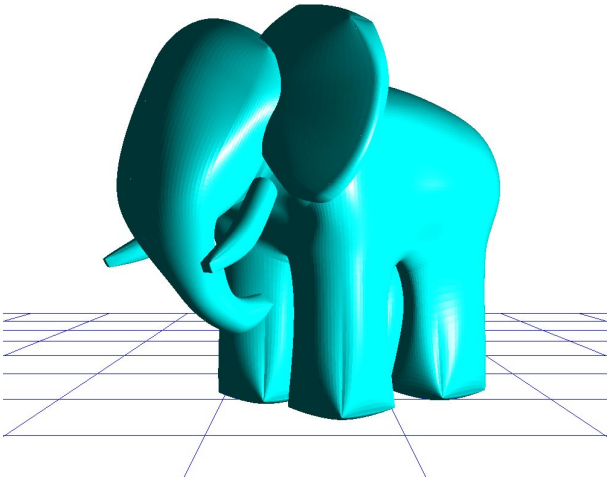
COSC422 Assignment 1 Report

Andrew French - ID: 11147452

Bezier Surface Modelling

In this program, Gumbo the Elephant is the generation of a bi-cubic Bezier surface model, using 4x4 Bezier patches as inputs, using the Tesselation Shaders to tessellate Gumbo correctly. These patches are then passed through the Geometry Shader to calculate the ambient, diffuse and specular lighting calculations for Gumbo based on a light source. The scene also displays a floor plane, which uses a set of parallel lines along the x and z directions and runs as another program alongside the program which generates Gumbo. This can be seen in Figure 1 below.

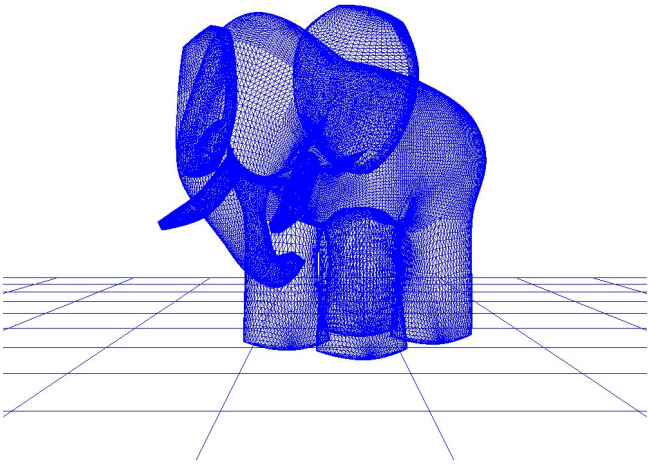


In regards to Dynamic level of detail, view dependent tessellation takes place where, as the camera gets closer to/further away from Gumbo the tessellation levels increase/decrease. This was done using the following equation:

$$level = \frac{(|vertex_z - camera_z| + 100) \times 18}{100} + 2$$

where the tessellation level is dependent on the absolute distance between the z coordinate of a vertex and the camera.

Alongside this, the wireframe of Gumbo can be shown by pressing the ‘w’ key, toggling between the wireframe and solid fill versions of Gumbo. The wireframe model can be seen in Figure 2 to the right.



The scene can also be navigated with the Camera by simply using the arrow keys.

Gumbo can also be ‘exploded’, which is an animation sequence exploding the Bezier surface model. This can be initiated by pressing the space bar. This is shown in Figure 3 below.

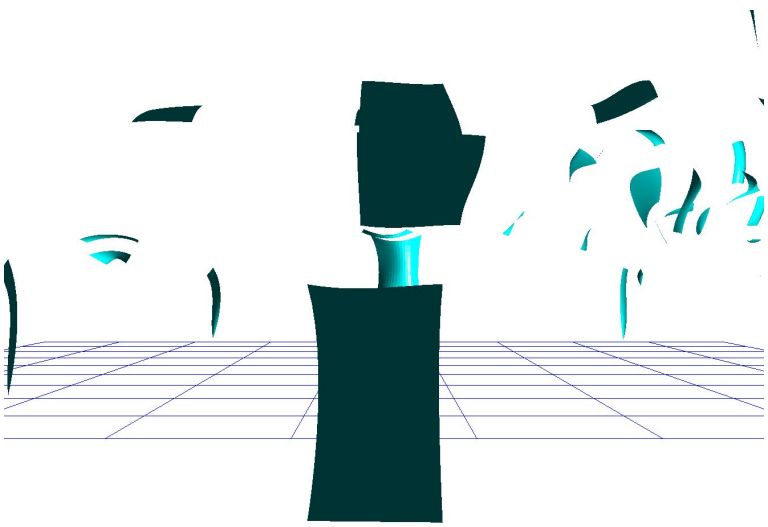
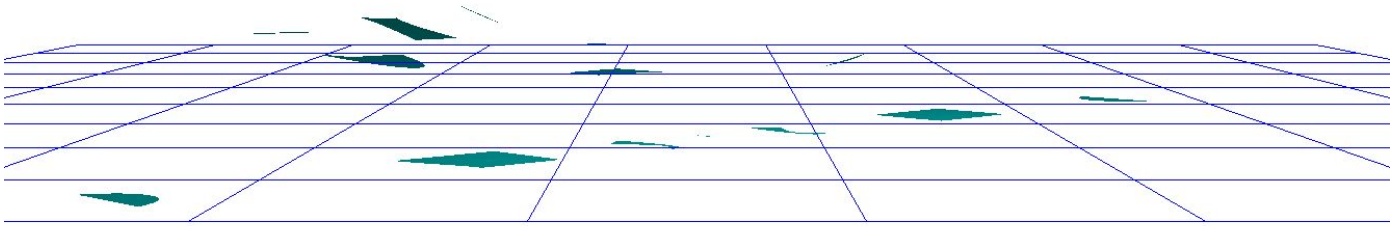


Figure 3 shows Gumbo mid-explosion and Figure 4 (below) shows Gumbo after the explosion has finished, and all of the separate patches are on the floor. In order to do this, each vertex had the following equation [1] applied to it in the Tessellation Control shader:

$$P_y = P_y + v_y t - \tfrac{1}{2} g t^2 \quad P_{xz} = P_{xz} + d_{xz} v_h t$$

This equation is based on simple kinematics and projectile motion, where P_{xyz} represents the input position of the vertex, which is then being updated to output an updated position based on the velocity, v (set as a constant), the time since explosion, t, the force due to gravity, g, and the value of the normal of the patch which the vertex relates to, d.

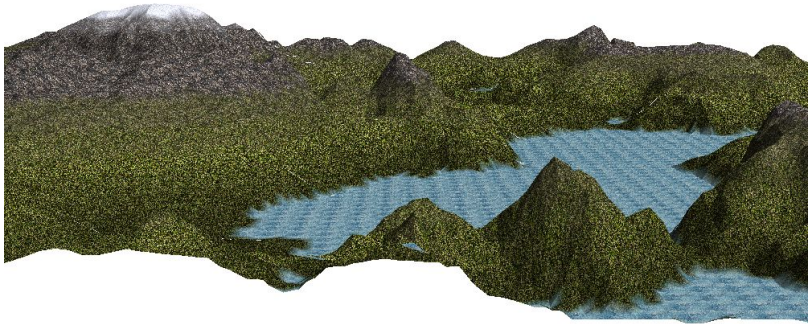


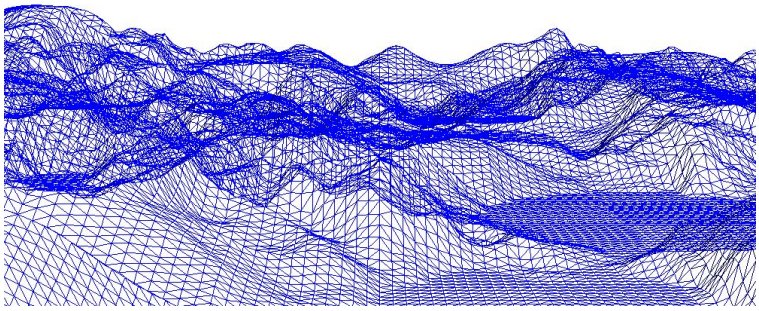
In terms of problems or challenges faced, there were a few. Firstly, having two independent programs which require their own shaders proved to be a difficult task, where you first have to create the floor using another program variable and its own fragment and vertex shader and then create another program for Gumbo with all of its shaders as well proved to be difficult. This was because I found it difficult to distinguish which variables had to be in which program, i.e. the MVP matrix or the Camera Position vectors for the lookAt() function. However, once I split all of the variables up and named them based on which program they belonged to, i found it much easier to modularise and debug my code as it was all contained. I also found it difficult to firstly understand how to get Gumbo showing and tessellated properly, because the input .txt file was different to the one shown in the labs, however, spending some time reading up on input and output in C++ allowed me to understand exactly how the format of the vertices in the .txt file were being mapped to values in the array in the program. It also helped to print out the variables as they were being created in the array.

Terrain Modelling and Rendering

The Terrain is generated through tessellation and mapping the y coordinate of vertices to a height map to create a terrain like output. The textures are then mapped creating a dynamic and visually appealing scene. These textures are snow[2], water[3], rock[4] and grass[5]. The Geometry shader is then used to display ambient and diffuse lighting based on the position of the light source. The output of this can be seen in Figure 5 (below).

The first feature implemented is Dynamic Level of Detail, where as you move the camera around the scene, the tesselation levels of the terrain change based on the camera's position, with the same equation used as shown in the Gumbo program. This is easy to see when in wireframe mode (Figure 6 below), which is seen by pressing the 'w' key, which toggles between wireframe and textured mode.

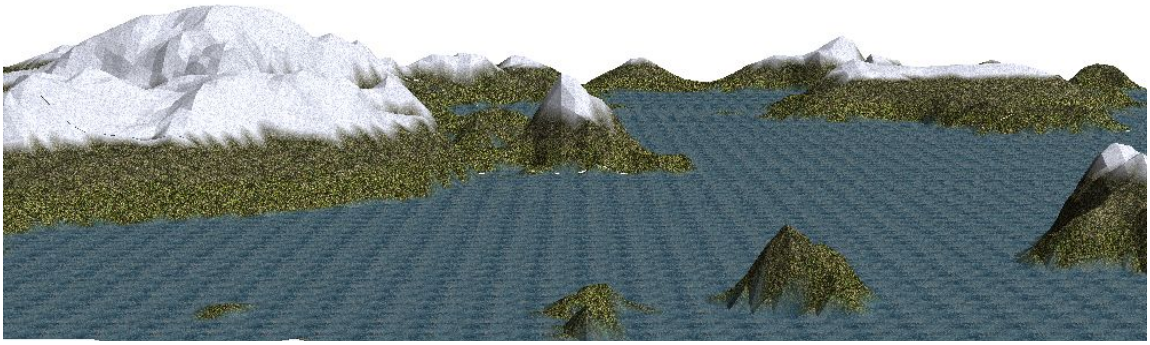




The setting of tessellation levels again used the same equation as the bezier surface model above.

Alongside this, I was able to find another height map [6] which is shown in Figure 7 (below). To toggle between these two height maps use the '1' and '2' key.

In regards to extra features, three were implemented to get the total extra marks of 3. Firstly, adjustable water levels were implemented which allow the user to press the 'p' key and 'o' key to raise and lower the water levels respectively. Also, the snow levels are adjustable which allow the user to press the 'l' key and 'k' key to raise and lower the snow levels respectively. As well as this, the colour of the water gets darker as you increase the water level and gets lighter as you decrease the water level. All of these can be seen when comparing Figure 8 (below) to Figures 6 and 7.



In terms of problems or challenges faced, there were a few. Blending textures nicely was quite difficult, as it meant that you had to try and find a nice gradient of increasing how dark or light one texture is based on the y coordinate of the vertex. To solve this, I ended up setting certain y values at which the textures would change colour, however I think this could have been done more elegantly. Getting water/snow levels to rise and fall properly was also difficult as it meant that you had to update a local height variable every time up or down was pressed and then pass those through to the Geometry shader and ensure that the snow and water level textures were set before any of the others. This was difficult but taught me a lot about how the mapping of the textures works and how it ended up being quite simple in the end, even though it seemed difficult at the start. Finding a good second height map was also difficult, as there were many out there but many didn't produce an output that was smooth between each vertex. Many height maps created an output that was very 'spiky' and had very large increments between the height of each vertex making the output look very disjointed. To solve this, I ended up having to find height maps where the gradient from black to white colours wasn't very severe and that there weren't any height maps which had any extremely black or white elements to it. Also, finding textures that didn't look 'fuzzy' was quite difficult, as often I would have textures which were seamless or low resolution and this would cause this sense of 'fuzziness' to appear where it looked as if the texture was moving when you would navigate around the scene. To solve this, I tried to find textures that had a large amount of detail and were very high resolution and this fuzziness seemed to disappear.

References

1. COSC422 Lecture slides, "2 Shader Applications".
2. Snow Texture https://www.sketchuptextureclub.com/textures/nature-elements/snow
3. Water Texture https://www.123rf.com/photo_39278603_seamless-water-texture-abstract-pond-background.html
4. Rock Texture https://www.pinterest.nz/pin/553590979181905018/
5. Grass Texture http://seamless-pixels.blogspot.com/2015/09/seamless-grass-texture.html
6. Second Height Map https://imgur.com/gallery/NiKS2