COSC422 Assignment 1

James MacKay

Outline

Bezier Surfaces

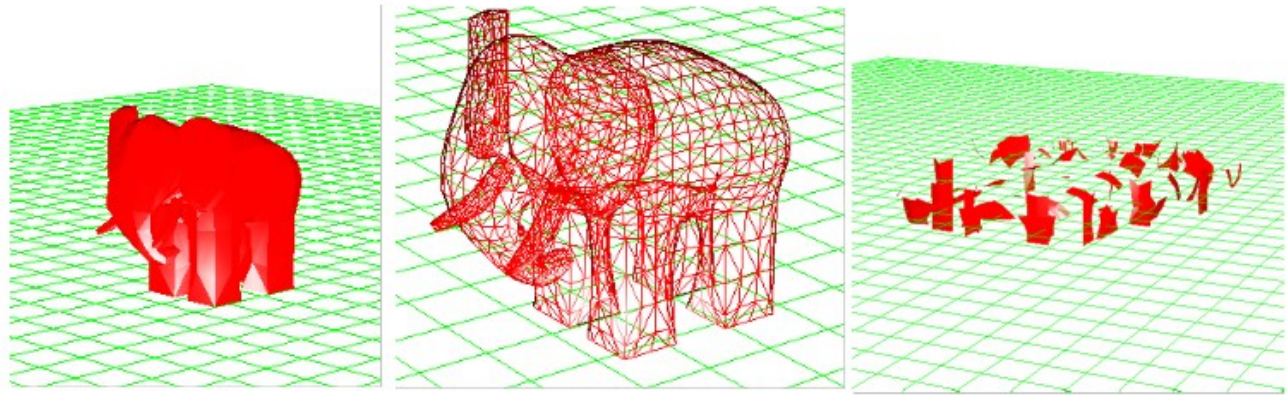

Figure 1: (From left) The Gumbo model seen from afar with filled in triangle primitives showing the low tessellation level, a closeup wireframe of the model showing a higher level of tessellation, and the results of exploding the model.

Stored in the `Bezier` directory, the first part shows a simple floor plane with a tessellated model revolving on top. 'Gumbo' is created from a number of patches which are tessellated according to the camera's distance to the model. The calculation of the level of tessellation is carried out in the tesselation control shader and is reproduced below. Each patch contains 16 vertices and is tessellation according to a bicubic interpolation.

```
tesselationLevel = max(abs(400.0 / camz), 1.0);  Assignment.cont:11
```

The camera can be moved towards and away from the model using the up and down arrow keys respectively. Because the model remains at the origin, we simply use a scaled inverse of the absolute value of the camera's z position to determine the tessellation level. Note that this means the entire model is tessellated to the same depth, and not on a per-patch basis. To more easily view the level of tessellation, the 'w' key can be pressed to see a wireframe version of the model.

The model is also lit using the Phong-Blinn lighting model and displays specular reflections.

Pressing the spacebar causes each patch in the model to explode radially outward and fall to the ground. The direction in which each patch begins moving is determined by the average of the four corner vectors of the patch (line 14 of `Assignment.cont`). While the x and z components of the movement remain constant (ignoring the rotation), the y-component decreases proportional to the square of the time since the explosion (line 20 of the same file).

```
pos.y -= 0.03 * t2 * t2;                                    Assignment.cont:20
```

Line 17 (below) calculates the time at which the center of the patch will lie at y-position 0 (the floor plane). This expression calculates the positive root of the equation for the patch's y position. If this time is less than what has elapsed since the explosion, this value is used to calculate the patch position rather than the actual time elapsed. In this way each patch stops independently of the others according to when its centre has reached the floor plane (see the third image above).

```
t_end = (norm_centre.y + sqrt(norm_centre.y *      Assignment.cont:17
     norm_centre.y + 0.12 * centre.y)) / 0.06;
```

Pressing the spacebar again causes the model to reset to allow for another explosion.

The floor plane and bezier surfaces are rendered using two separate shader pipelines. This is due to the fundamental differences in the way they are tessellated and fragments coloured.

Terrain

Figure 2: From left: The initial terrain map showing no cracks, blended textures and a rippled water level. Next, the same scene without adjusting for cracking with a crack circled. Finally the alternative scene showing the ice texture on the water.

The second program, located in the `Terrain` subfolder, renders a terrain and provides some options to manipulate the scene. The camera can be moved using the arrow keys.

The scene shows three main textures. Snow at the highest point, followed by a rocky texture and finally grass or bush. These textures are linearly blended with each other to enable a smooth transition. The blending is performed in the fragment shader on lines 29 to 36.

The snow level can be dropped by pressing the ‘s’ key and increased by pressing shift+s. Similarly the water level is controlled with ‘+’ and ‘-’. If the snow level drops below the water level, the rippling water is replaced with an ice texture, shown in the third screenshot above.

Two different height maps are provided which can be toggled between by pressing keys ‘1’ and ‘2’.

```
z3 = (mvpMatrix * ((gl_in[2].gl_Position +          Terrain.cont:13
gl_in[3].gl_Position) / 4)).z;
```

Like the bezier model, the scene can be viewed as a wireframe by pressing the ‘w’ key. Similarly as well, patches closer to the camera are tessellated to a higher degree than those farther away. In order to prevent cracking where two patches which share an edge are tessellated to different levels, patch *edges* are tessellated according to their distance from the camera position rather than the distance of the entire patch. An example of calculating this centre z value is shown for the far vertex of a patch above. This z value is then converted into a tessellation level in a similar manner to the bezier surface. The effect of this compared in the first two screenshots above.

The rippling effect on the water is achieved using a combination of trigonometric functions on the coordinated of the fragment and the time since execution began.

```
ripple = 0.1 * sin((position.x + sin(position.z + t  Terrain.geom:42
/ 10.0) + 0.4 * sin(t / 4.0)) * 1.5) + 0.9;
```

The overall effect of the above is to vary the intensity of the water colour from 80% to 100%. The `sin(position.z + t / 10.0)` term results in the ‘wavy’ appearance and movement in the z direction. The `0.4 * sin(t / 4.0)` controls the oscillation back and forth of these waves according to the time elapsed. Finally, the outer sine function determines the repeating nature of the waves along the x direction.

The scene is lit similar to the bezier surface with the Phong-Blinn model but there are no specular reflections.

Texture Sources:

https://www.emanueleferonato.com/2011/07/12/how-to-turn-your-own-photos-into-high-quality-seamless-textures-with-genetica/

https://hannahloughridge.wordpress.com/tag/texture/

https://forsakenaskyrimmod.wordpress.com/tag/skyrim-heightmap/

https://www.goodtextures.com/category/seamless-snow-ice

https://www.the3rdsequence.com/texturedb/category/mountain/