Alex Tompkins (ato47)

79004580

COSC422 Assignment 1

Bezier Surface Modelling

The Bezier program demonstrates the rendering of a model from 4x4 Bezier patches. I chose to use the provided Gumbo model by default, however I also provided support for the Teapot model. The selected model can be changed by simply modifying the *MODEL_FILENAME* define at the top of *Bezier.cpp* and recompiling.

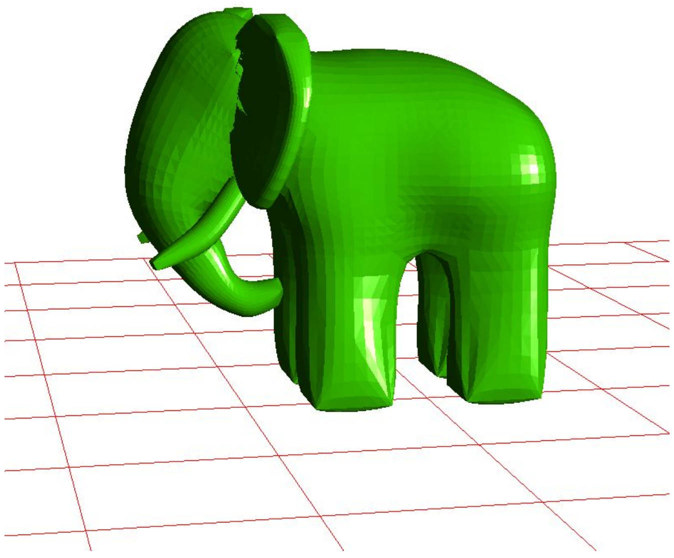


Figure 1: bezier model is displayed with ambient, diffuse and specular lighting

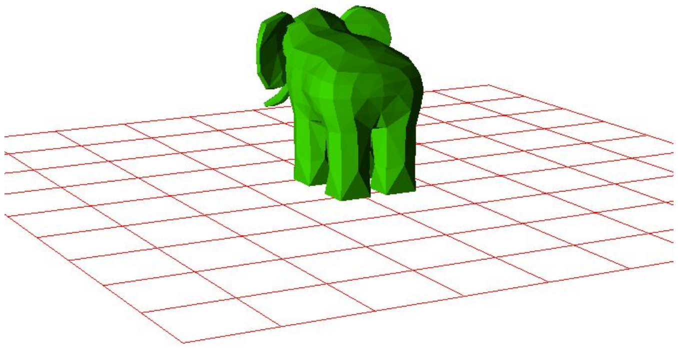


Figure 2: dynamic tessellation level dependent on distance between model and camera

Controls

UP/DOWN: Moves the camera towards/away from the model.
LEFT/RIGHT: Rotates the camera left/right around the model.
A/Z: Moves the camera upwards/downwards.
SPACE: Begins/resets the explosion.
W: Toggles wireframe mode on/off.

Features

Lighting

As shown in figure 1, the bezier model is lit with ambient, diffuse and specular lighting components relative to a light source. All lighting is calculated in the geometry shader stage of the pipeline.

Dynamic Tessellation

The patches are tessellated via the use of a tessellation control and a tessellation evaluation shader. The control shader is responsible for setting the current inner/outer tessellation levels. The evaluation shader uses bi-cubic Bernstein polynomials to reposition the patch vertex. Figure 2 shows that the model's tessellation level decreases with distance, whereas more detail is shown in figure 1 when the camera is near. This is done by finding the distance between the camera position and the model (at the origin), then scaling the tessellation level by using the following relationship:

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

Where L is the tessellation level and d is the distance between the camera and the model. I used 10 for d_{min} and 150 for d_{max} because these seemed reasonable values to have the tessellation levels scale between. I set the tessellation level to scale between 10 (L_{low}) and 50 (L_{high}).

Explosion Animation Sequence

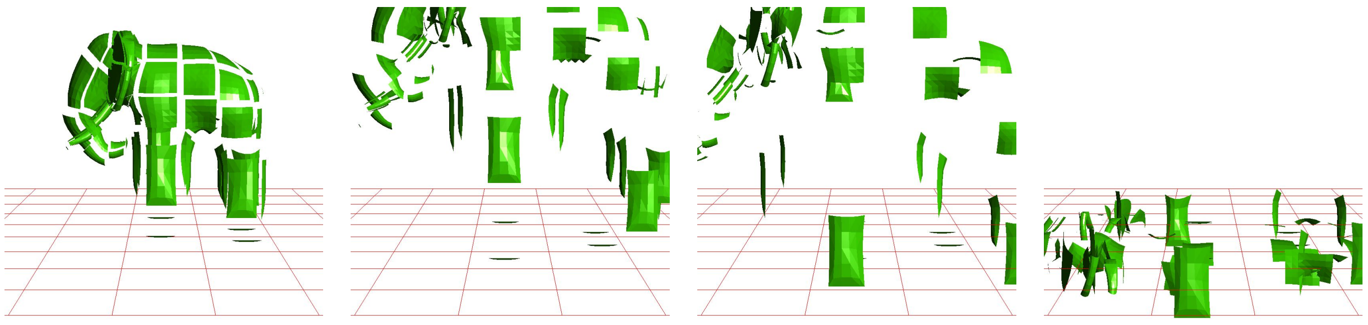


Figure 3: Gumbo exploding, each patch follows its own projectile motion path

To allow for an explosion animation, a uniform variable *timeSinceExplosion* is passed into the tessellation control shader. This value is incremented at a predetermined interval once the spacebar has been pressed. This value will then allow for the patch vertices' positions to be modified according to some kinematic equations of motion (adapted from the lecture notes).

$$P_{y_{new}} = P_y + v_{vert}t - \tfrac{1}{2}gt^2$$

$$P_{x_{new}} = P_x + \frac{P_x}{abs(P_x + P_z)} * v_{hori}t$$

$$P_{z_{new}} = P_z + \frac{P_z}{abs(P_x + P_z)} * v_{hori}t$$

I adjusted the calculation of the new horizontal position to take into account the original proportion of the patch along each horizontal axis.

One issue with this simple animation is that the patches eventually fall through the ground. I tried the naïve approach of simply taking the max of the patch's new y position and 0, but realised that although this stopped the patches from falling below the floor, they would still continue sliding horizontally. Therefore I realised I needed to calculate the time it would take for the patch to reach the ground, and use that value as a maximum bound for *timeSinceExplosion* when calculating the patch's new position.

Finding time *t* when patch hits the ground:

From original position to apex of its path:

$$v_f = v_i + at$$

$$0 = v_{vert} + gt_1$$

$$t_1 = \frac{v_{vert}}{g}$$

From apex to y = 0:

$$d = v_i t + \tfrac{1}{2}at^2$$

$$h = 0 + \tfrac{1}{2}at_2{}^2$$

Find h using equation for p_y at point t defined earlier

$$t_2 = \sqrt{\frac{2h}{g}}$$

$$t_{total} = t_1 + t_2$$

Floor Plane

I used the *generateData()* function from *Terrain.cpp* to generate the vertices/elements for my floor plane. I then set up a second VAO with two VBOs to buffer the data for this second program. The floor program uses only a vertex and fragment shader as it doesn't need to do any lighting calculations etc. I also realised that the floor program needs to have its own location for the mvpMatrix when it is passed in as a uniform variable.

Ear clipping

There is currently an issue with the program where the patches for Gumbo's ears partially clip with the patches defining his head, which causes some visual artifacting around this area. I was unable to resolve this issue as I could not find what was causing it.

Terrain Modelling and Rendering

The terrain program demonstrates the rendering of a terrain model generated from a height map. I coloured this terrain model using grass, rock, snow, water (and ice) textures.

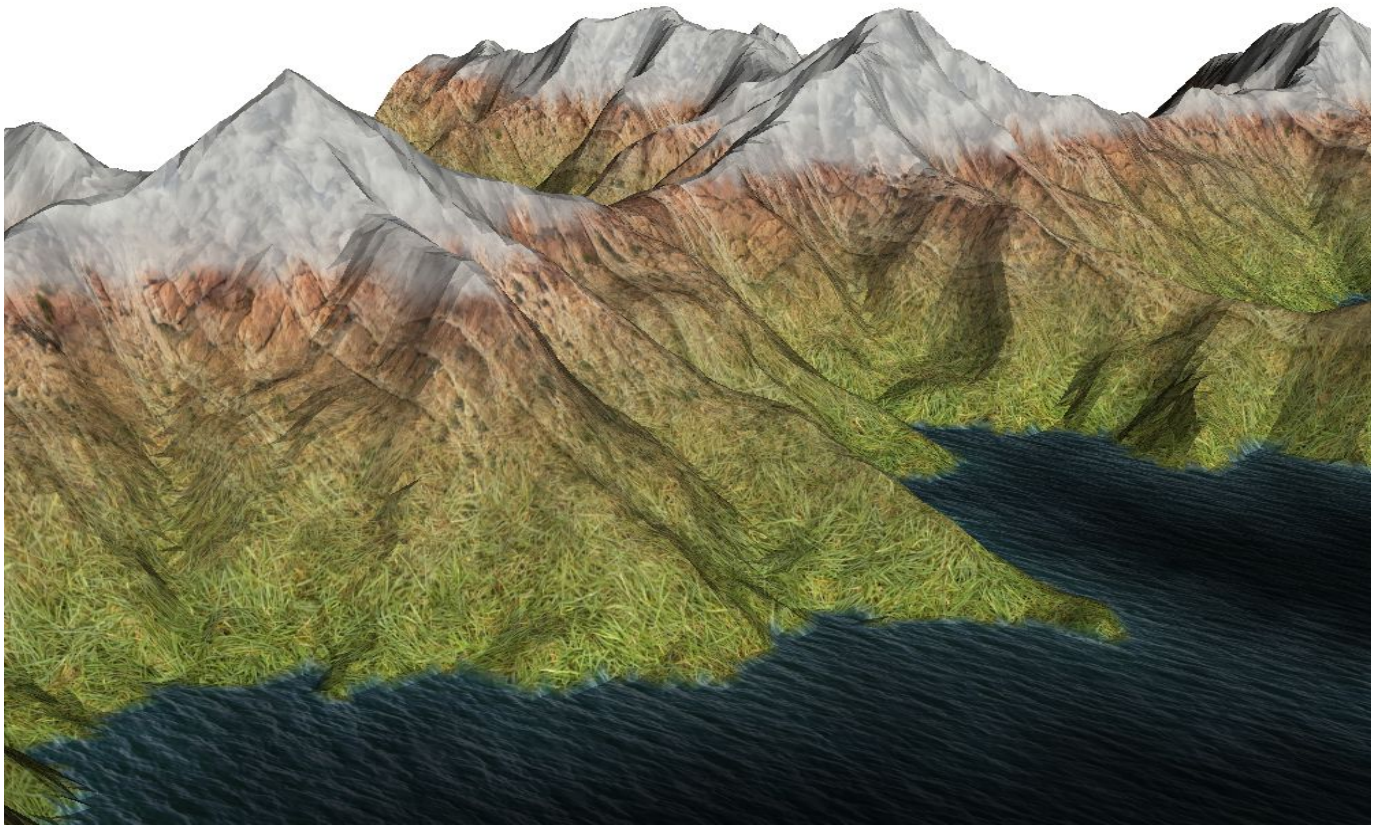


Figure 4: The terrain model; showing grass, rock and snow textures blended together, with a flat water region at the bottom

Controls

UP/DOWN: Moves the camera forwards/backwards (along the Z axis).
LEFT/RIGHT: Moves the camera left/right (along the X axis).
SPACE/X: Moves the camera upwards/downwards. (along the Y axis).

I/K: Tilts the camera up/down.
J/L: Tilts the camera left/right.
Y/H: Changes the water level higher/lower.
T/G: Changes the snow level higher/lower.
Q: Toggles day-night cycle on/off.
1: Switch to primary height map.
2: Switch to secondary height map.
W: Toggles wireframe mode on/off.

Requirements

Dynamic level of detail (per patch)

Whereas for the bezier model the tessellation level for the entire model was dynamically scaled, here each patch's tessellation level is decided individually depending on its distance from the camera. To calculate this distance, the eye position is passed to the tessellation control shader as a uniform variable and the difference between it and the average position of the patch is found. The length of this difference is then passed into the same equation to scale the tessellation level:

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

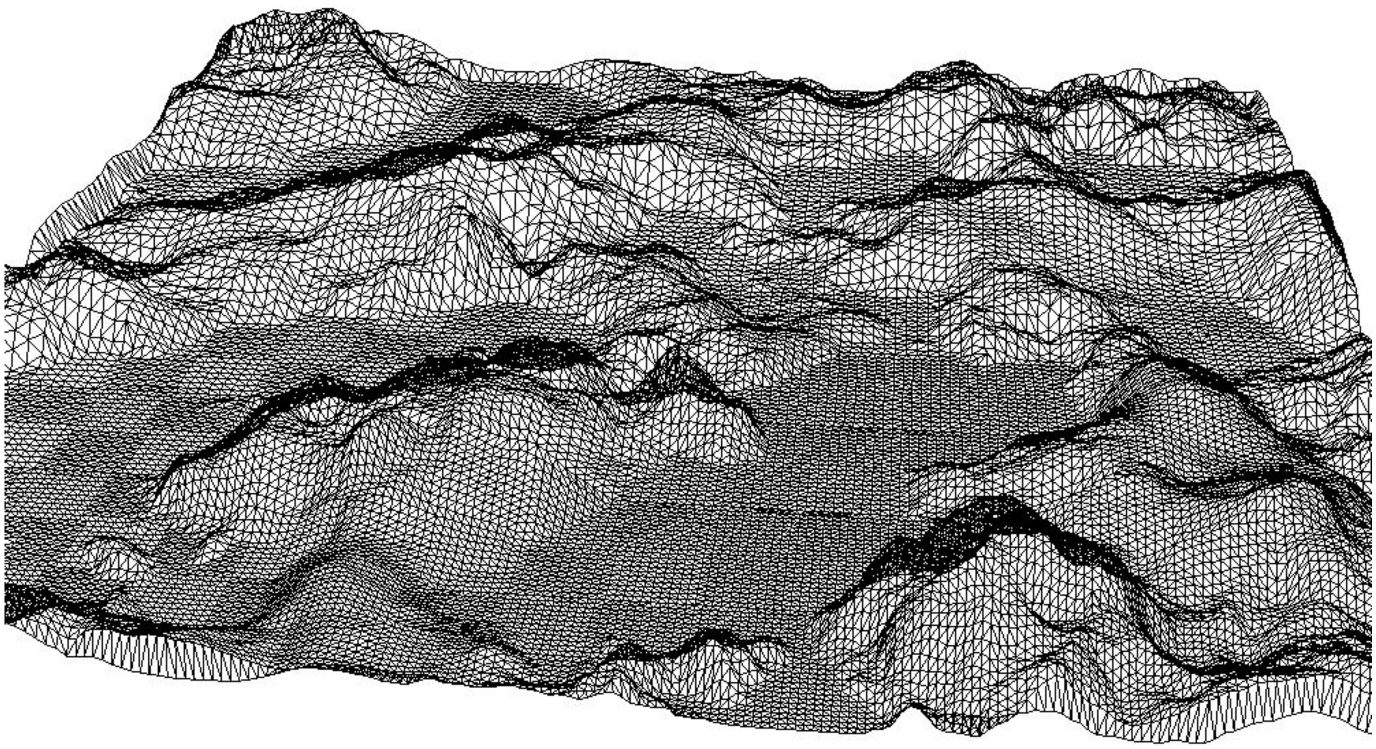


Figure 5: nearer patches of terrain have a higher tessellation level than those further away

Lighting calculations

Lighting calculations for ambient and diffuse terms on the terrain are carried out in the geometry shader *Terrain.geom*.

Two height maps

A second height map has been added. This height map was extracted from real geographic data of the area around Akaroa, i.e. the south side of Banks Peninsula. This secondary height map is viewable by pressing **2**. The user can then press **1** to return to the primary height map.

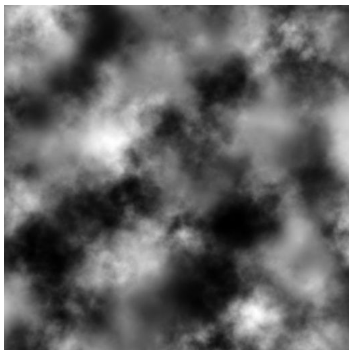


Figure 6: the provided height map

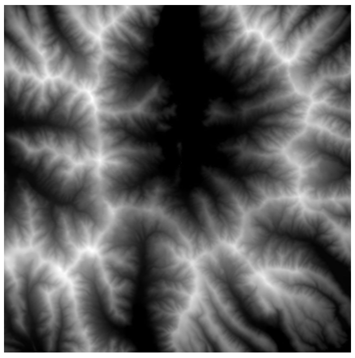
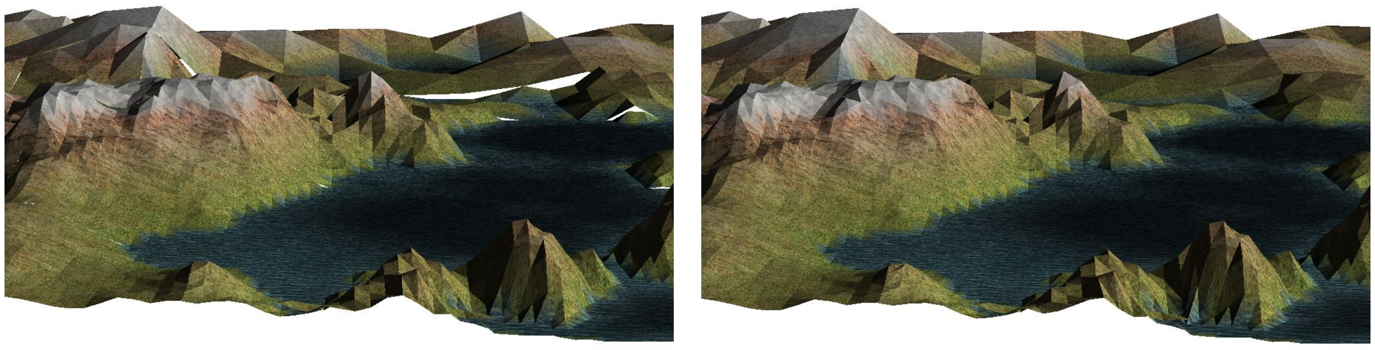


Figure 7: the secondary height map, of the area surrounding Akaroa on Banks Peninsula

Extra Features

Resolve cracking

The issue of cracking can be resolved by ensuring that an edge always has the same tessellation level, no matter what patch it belongs to. Therefore, instead of deciding a tessellation level on our whole patch, we can decide outer tessellation levels individually. To do this, we find the average position of the edge using the two vertices on that edge, then use that average edge position's distance from our camera to determine the tessellation level (using the same equation as earlier).



Before fix: cracking clearly visible on edges between distant patches

After fix: no cracking because shared edges have same tessellation level

Figure 8: demonstration of cracking using exaggerated dynamic level of detail

Adjustable water level

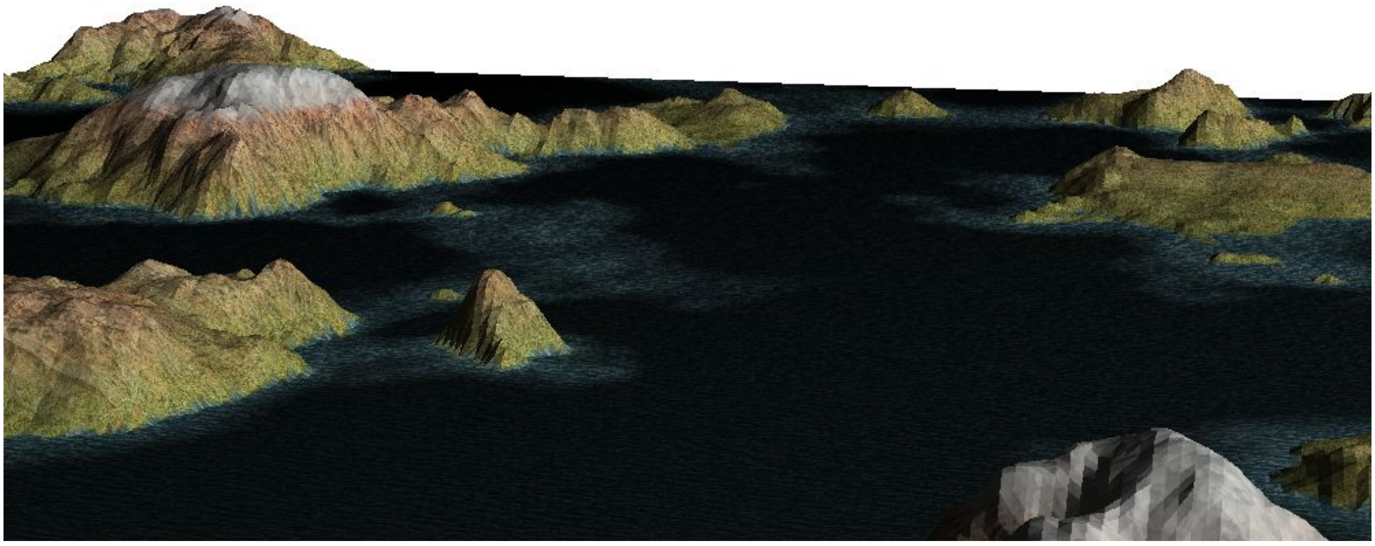


Figure 9: the water level has been raised so that the majority of terrain is submerged

I made the water level adjustable by passing it into the geometry shader as a uniform variable. **Y and H** can be used to change it higher/lower.

Adjustable snow level

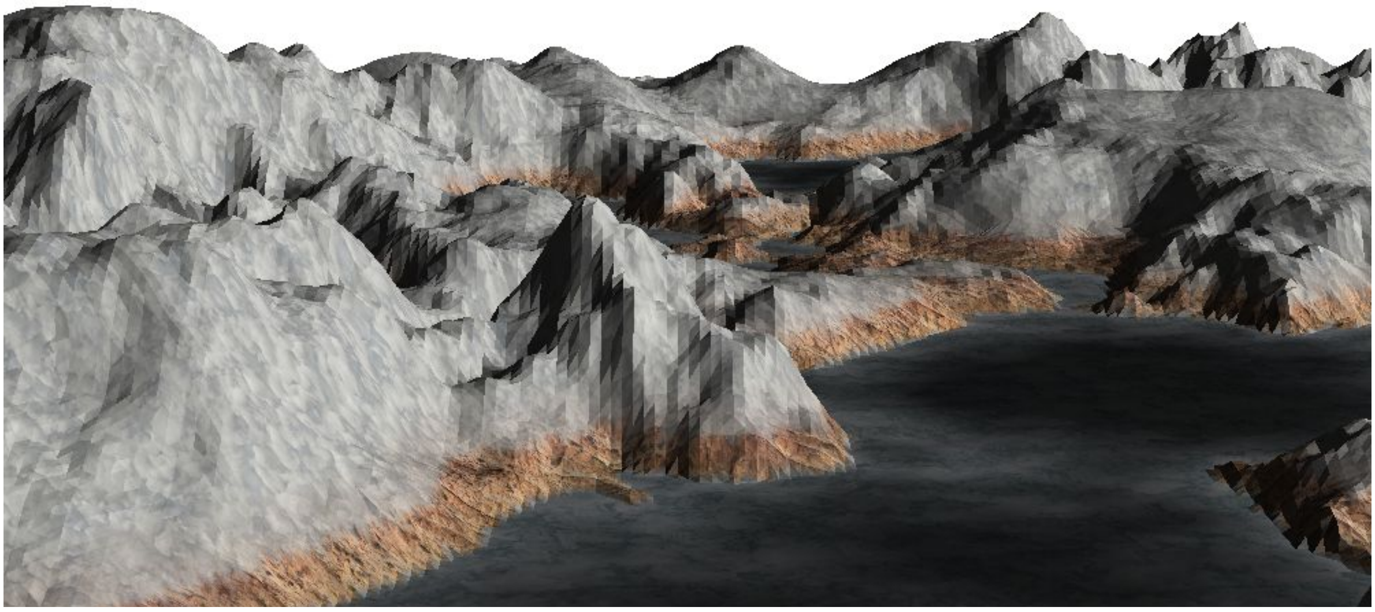


Figure 10: the snow level has been lowered until it nearly reaches the water, which has been frozen to ice

I made the snow height adjustable by passing it into the geometry shader as a uniform variable. **T and G** can be used to change it higher/lower.

Furthermore I added an effect for when the snow level gets too close to the water level: use an ice texture instead of the regular water texture to simulate lakes being frozen over.

Water features (colour variation with depth)

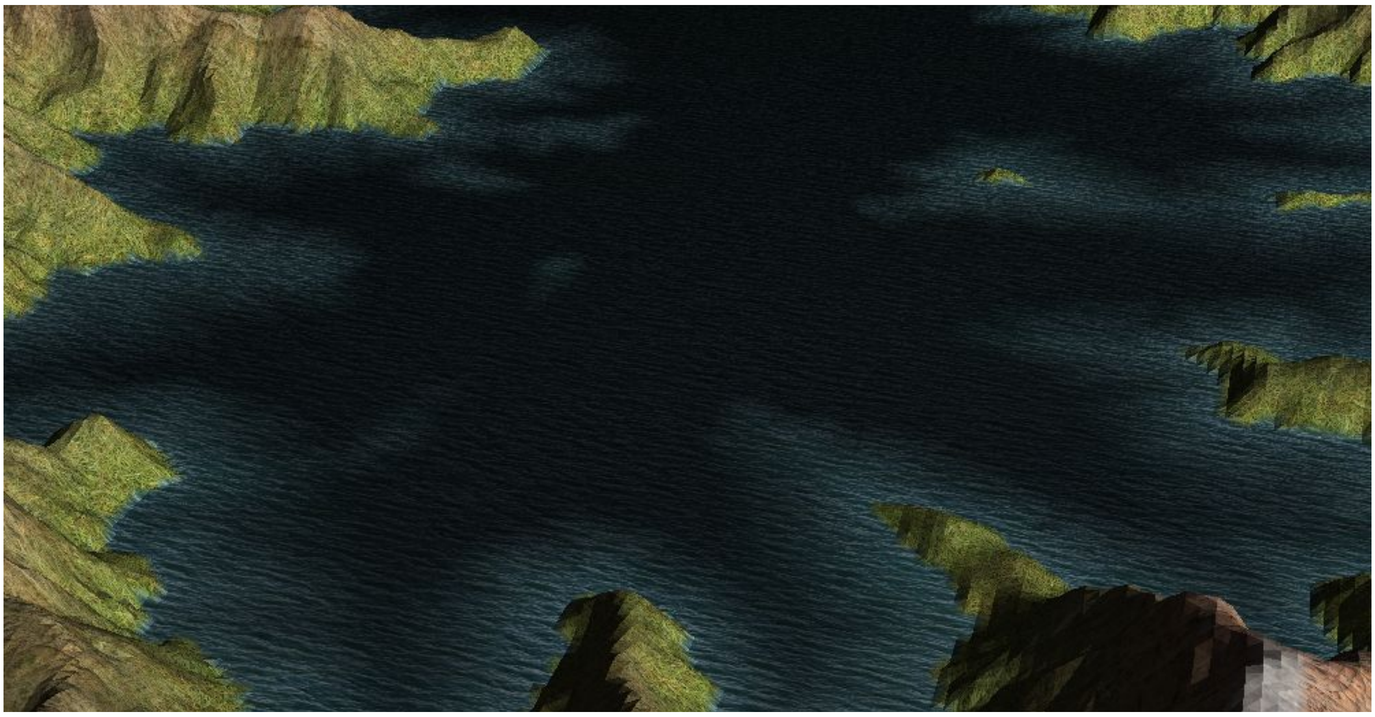


Figure 11: variation in water colour based on depth clearly visible

I decided to implement a water feature: namely, to make water appear darker the deeper the terrain beneath it. This was easily done by figuring out the difference between the original y value on the heightmap and the water level (which the point is adjusted to), then using that distance as a scale factor for a negative brightness term.

Day/night cycle

I also decided to implement a day/night cycle to demonstrate the different lighting situations possible on the terrain based on the time of day. The day/night cycle can be toggled on/off by pressing **Q**. This simply moves the light source in a circular path around the terrain.

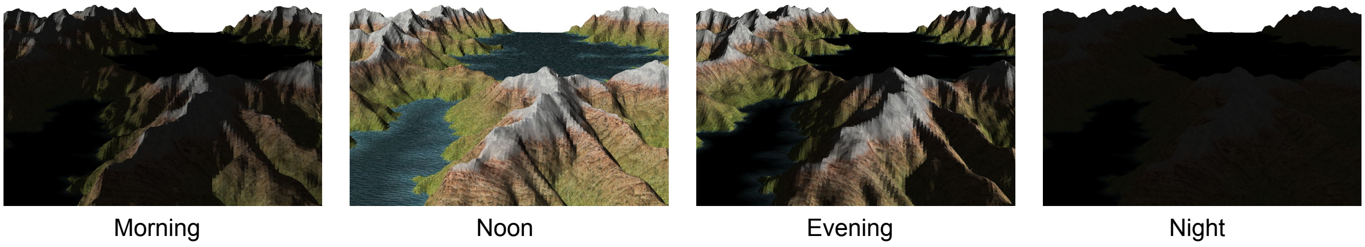


Figure 12: lighting during different phases of the day/night cycle

Resources

The seamless grass, rock, snow, water and ice textures were all obtained from textures.com, an online repository of texture files. According to their website, “textures may be used for free in 2D or 3D computer graphics, movies, printed media, computer games and 3D models”.

My second height map was acquired from https://tangrams.github.io/heightmapper/, an online tool that can retrieve a height map from any location in the world.

All other assets (including the patch geometry files) were provided with the assignment.

Program Setup

To compile the two programs, run the following commands in the root directory of the project:

```
g++ -Wall -o bezier Bezier.cpp -lm -lGL -lGLU -lglut -lGLEW
g++ -Wall -o terrain Terrain.cpp -lm -lGL -lGLU -lglut -lGLEW
```

Then to run the executables, run the following commands:

```
./bezier
./terrain
```