Student Name: Yuezhang Zhu
Student ID: 97245310

COSC422 Assignment1 Report Terrain Rendering

The Scene

1. The program renders two terrain models from the height maps of Mount Cook and Mount Ruapehu, as shown in Figure 1.1 and Figure 1.2.

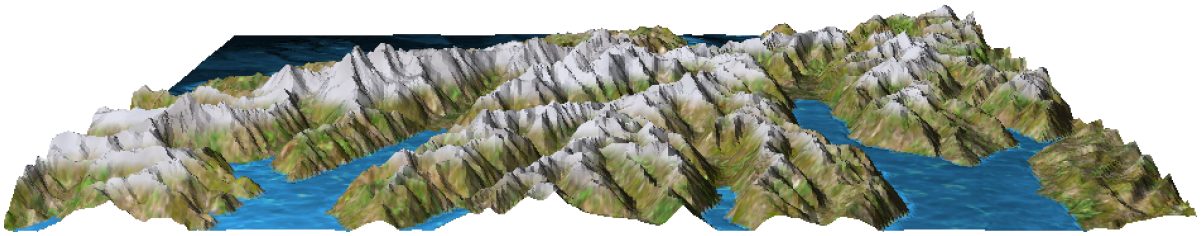

Figure 1.1 Mount Cook Scene

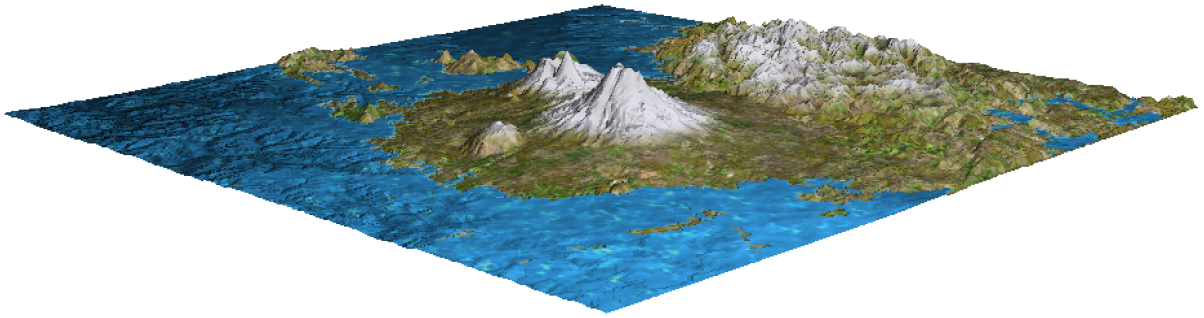

Figure 1.2 Mount Ruapehu Scene

2. Dynamic level of detail is demonstrated in Figure 2, with terrain patches closer to the camera showing higher tessellation levels than those further away. The tessellation levels of a patch are calculated in the control shader, using the following equation:
tessellation level = (d - d_min) / (d_max - d_min) * (l_low - l_high) + l_high
where d is the distance between the camera position and the patch/edge center, d_min and d_max are user-defined minimum and maximum values of patch distance, l_low and l_high are user-defined low and high tessellation levels. Please press the **space** key followed by key **‘ t ‘** to clearly see the tessellation level changes (see section Key Functions for detail).

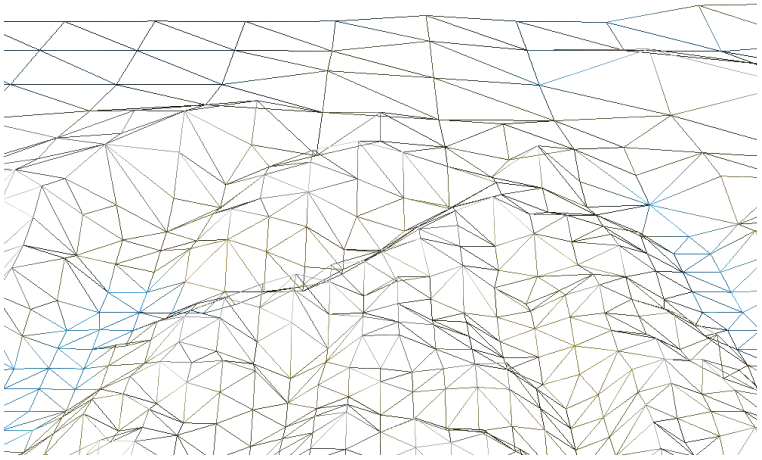

Figure 2. Dynamic level of detail of Mount Cook with l_low=1 and l_high=10

3. Each Terrain is textured with water, grass, and snow, with a smooth transition between the textures, as shown in Figure 3. The level of blending between textures depends on the height of the vertex. For example, if the height of a vertex is in the range where both grass and snow exist, the closer its height is to the snow level, the less the texture weight for grass.

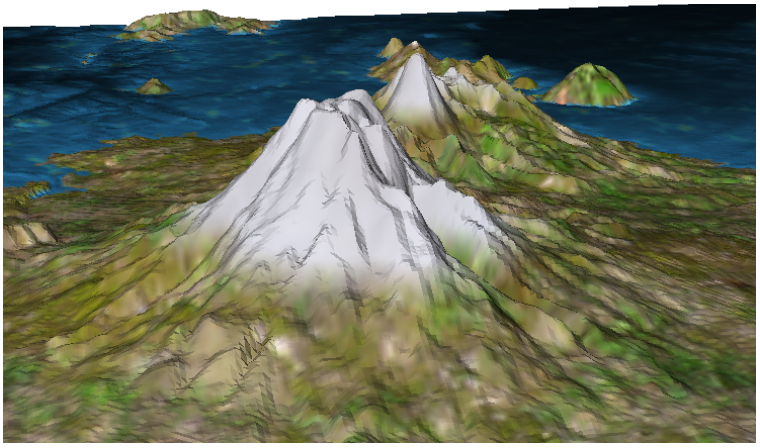

Figure 3. Blending of textures

4. Lighting calculations for ambient and diffuse terms are performed in the geometry shader to render the terrain models under a light source, as shown in Figure 4.

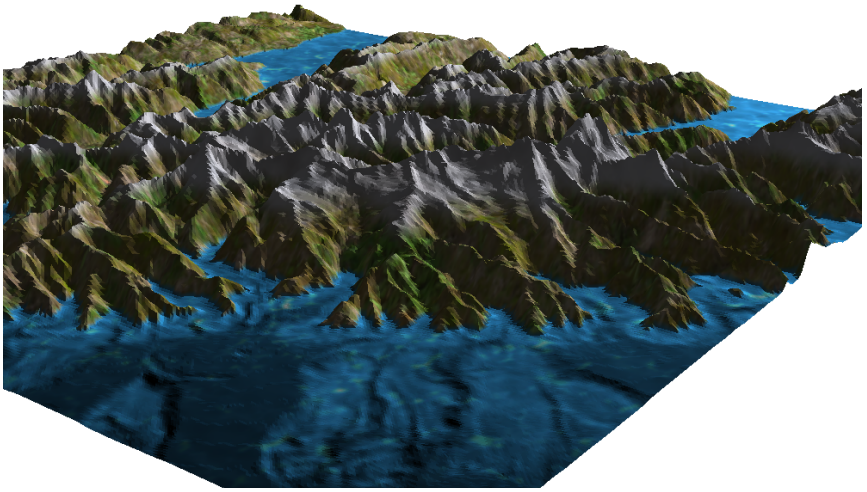

Figure 4. Ambient and diffuse lighting on Mount Cook

Extra Features

1. Cracking is fixed by edge correction (toggle key **'c'** to see). The distance of the center of the patch from the camera is now only used to compute the inner tessellation level of a patch. The outer tessellation levels are computed using the distance of the center of the corresponding edge from the camera. Figure 5.1 - 5.3 demonstrate the solution to cracking.

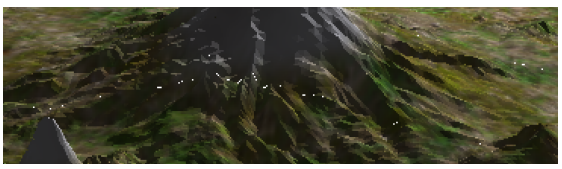

Figure 5.1 Cracking

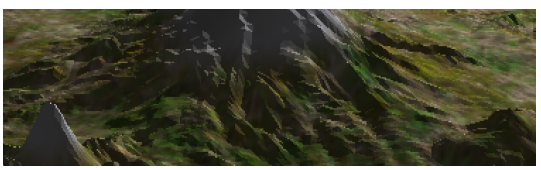

Figure 5.2 No cracking

```
int innerLevel;
int outerLevels[4];

vec4 patch_center = (gl_in[0].gl_Position
                + gl_in[1].gl_Position
                + gl_in[2].gl_Position
                + gl_in[3].gl_Position) * 0.25;

float patch_d = distance(eyePos, patch_center);

innerLevel = getTesLevel(patch_d, d_min, d_max, l_low, l_high);

if (fixCracking){
    for (int i = 0; i < 4; i++){
        vec4 side_center = (gl_in[int(mod(i - 1, 4))].gl_Position
                          + gl_in[i].gl_Position) * 0.5;
        float side_d = distance(eyePos, side_center);
        outerLevels[i] = getTesLevel(side_d, d_min, d_max, l_low, l_high);
    }
}else{
    for (int i = 0; i < 4; i++){
         outerLevels[i] = innerLevel;
    }
}
```

Figure 5.3 Code for fixing cracking

2. The water level can be adjusted by pressing key **'q'** (increase) and **'a'** (decrease). Figure 6 shows the terrain with an increased water level.

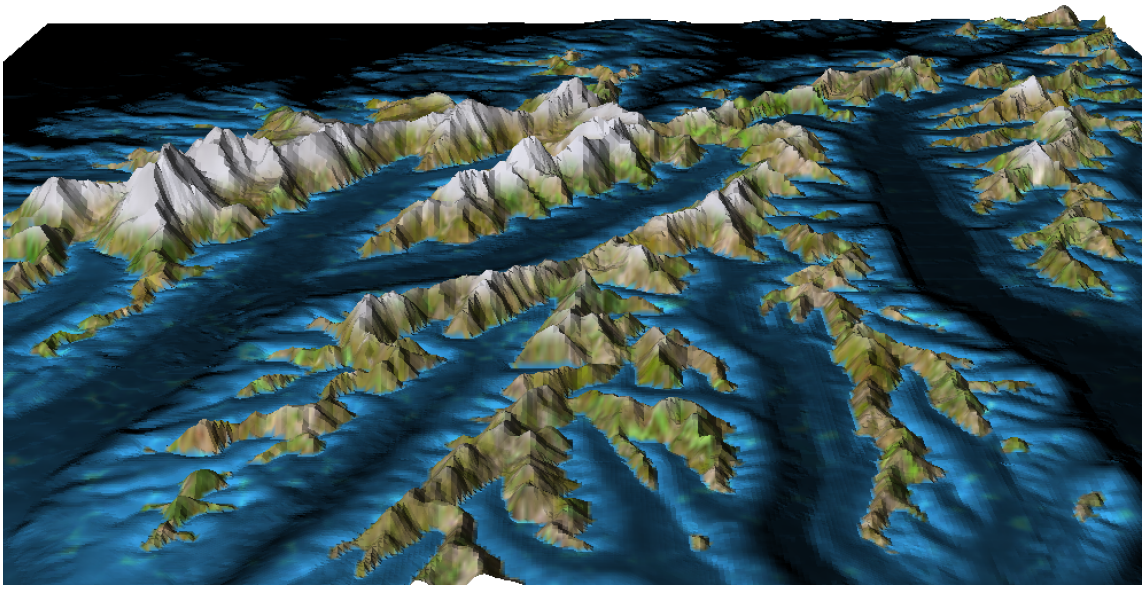

Figure 6. Adjustable water level, water depth variation, and water wave

3. Water depth variation is calculated in the geometry shader. The distance between the original height of the vertex and the water level the vertex is adjusting to is first calculated. This distance is then divided by the maximum water depth to get the depth factor. The depth factor is then subtracted from the lighting term, and passed onto the fragment shader. The depth variation is shown in Figure 6, and the code is outlined in Figure 7.

```
//water depth variation
float depth = newPositions[i].y - oldPos.y;
float depthFactor = depth / dmax;

//sum light, output to frag shader
lightFactor = min(ambient + diffuse - depthFactor, 1.0);
```

Figure 7. Water depth variation code

Student Name: Yuezhang Zhu
Student ID: 97245310

4. Water waves are rendered in the scene as shown in Figure 8.1. The waves are calculated in the geometry shader, and the wave tick is updated by a timer function in the main terrain program. For a vertex belonging to the water region, the wave height is added to the water level to obtain the new vertex height. The original height of this vertex is compared to the water level to select the correct water texture. Figure 8.2 demonstrates the water wave calculation.

Figure 8.1 Water waves

```
float getWaterWavaHeight(vec4 position)
{
    float d = waterLevel - position.y;
    float m = 0.2;  // wave height
    float waterFrequency = 1.0;
    float y = m * sin(waterFrequency * (d - float(waterWaveTick)*0.1));
    return y;
}

vec4 newPositions[3];

for (int i = 0; i < gl_in.length(); i++)
{
    newPositions[i] = gl_in[i].gl_Position;
    if (newPositions[i].y < waterLevel){
        float waterWavaHeight = getWaterWavaHeight(newPositions[i]);
        newPositions[i].y = waterLevel + waterWavaHeight;
    }
}
for (int i=0; i< gl_in.length(); i++)
{
    vec4 oldPos = gl_in[i].gl_Position; //unmodified water position

    //pass texWeights to frag shader for selecting correct texture
    if (oldPos.y < waterLevel){                //water
        texWeights = vec3(1.0, 0.0, 0.0);
    }
```

Figure 8.2 Water wave calculation code in the geometry shader

5. The snow level can be adjusted by pressing key **'w'** (increase) and **'s'** (decrease). Figure 9 below shows an increased snow level.

Student Name: Yuezhang Zhu
Student ID: 97245310

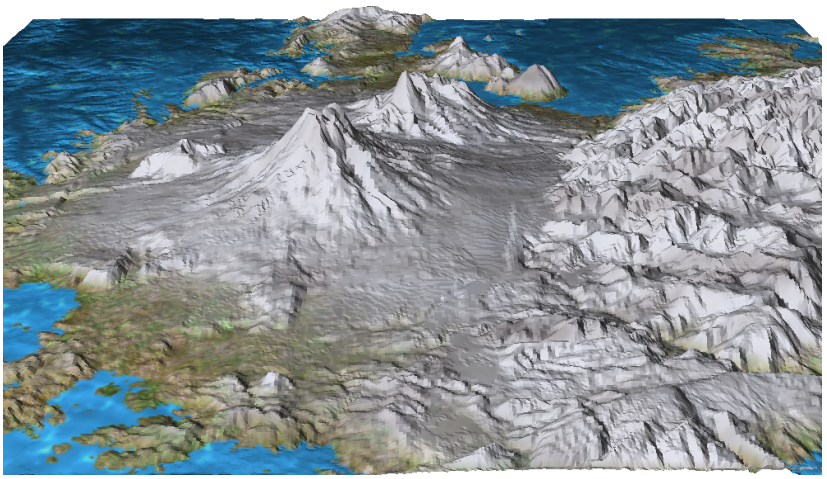

Figure 9. Adjustable snow level

6. Fog can be enabled/disabled by pressing key **'f'**. The fog level can then be adjusted by pressing key **'i'** (increase) and **'d'** (decrease). The fog level factor is calculated in the geometry shader, and passed onto the fragment shader for texture mix. The result is demonstrated in Figure 10.1, and the equations used are shown in Figure 10.2 - 10.3.

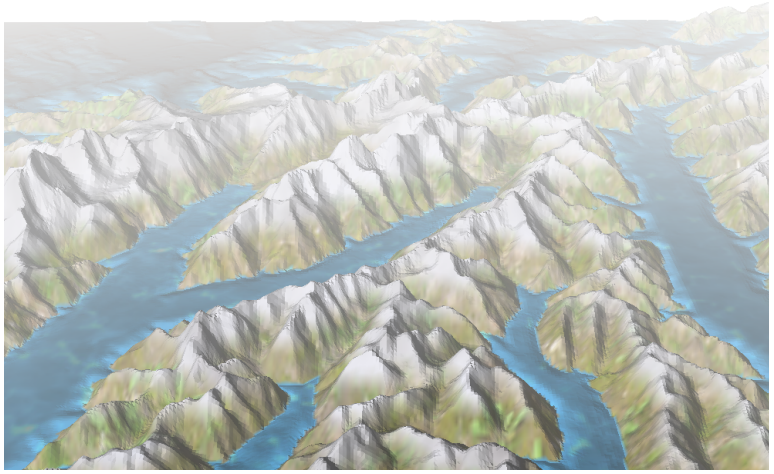

Figure 10.1 Adjustable fog level

```
//fog factor for mix in frag
if (hasFog){
    fogFactor = 1 - exp(-pow(length(newPositions[i] * fogLevel), fogGradient));
    fogFactor = clamp(fogFactor, 0.0, 1.0);
} else{
    fogFactor = 0.0; //no fog
}
```

Figure 10.2 Fog factor calculation in geometry shader

```
outputColor = lightFactor * (grassTexColor + snowTexColor + waterTexColor);

if (hasFog){
    vec4 fogColor = vec4(255/242, 255/248, 255/247, 1);
    outputColor = mix(outputColor, fogColor, fogFactor);
}
```

Figure 10.3 Fog mix in fragment shader

Student Name: Yuezhang Zhu
Student ID: 97245310

Key Functions

Key	Function
↑	move camera forward
↓	move camera backward
←	move camera left
→	move camera right
space	toggle wireframe mode on/off
t	toggle tessellation level range between [30, 100] and [1, 10], default [30, 100]. To clearly see the dynamic level of detail, please toggle to [1, 10]
1	show Mount Cook
2	show Mount Ruapehu
c	toggle cracking on/off
q	increase water level
a	decrease water level
w	increase snow level
s	decrease snow level
f	toggle fog on/off
i	increase fog level, only effective when fog is enabled
d	decrease fog level, only effective when fog is enabled
v	toggle water wave on/off
ESC	exit program

References

1. COSC422 lecture and tutorial materials by Professor Ramakrishnan Mukundan
2. “3D mesh processing and character animation: with examples using OpenGL, OpenMesh and Assimp” by Professor Ramakrishnan Mukundan
3. https://www.mbsoftworks.sk/tutorials/opengl4/020-fog/