COSC422 Advanced Computer Graphics - Assignment 1

Kanyakorn Kitisopakul 18582950

This assignment required the student to demonstrate how a tessellation shader and/or/both a geometry shader is used when rendering a 3D object/scene. The student was given a choice to do either Bezier surfaces, terrain rendering, or particle systems. For me, I chose to do terrain rendering.

Implemented Parts

Terrain Base

Terrain base acts as a foundation for the terrain. It is generated using an array containing the coordinates of the vertices of a large quad and an array containing the order which will be used to read vertices. The base currently has the size of 30x30. These arrays are passed into vertex shader and then fragment shader for rendering the terrain base(Figure 1).

Tessellation

Tessellation stages help us to model object more accurately. The tessellation level will determine how often the quad will be subdivided. This is set inside the control shader.

Lighting

To make the object/scene looks more realistic, lighting is applied. The calculation for diffuse term is calculated inside the geometry shader. Geometry shader provides mesh coordinates. Mesh coordinates could be used to find normal vector of the primitive(in this case, triangle). Once the normal vector is found, we could use it along with fixed light position to find the diffuse term(E1). Geometry shader then outputs this diffuse term so that the value will be received by the fragment shader. The fragment shader is then used the term along with material colour and light color to calculate diffuse reflection(E2). Figure 5 shows before and after lighting is applied to the program.

$$diffuse\ term = max(0,\ dot(l,\ n)) \qquad \text{[E1]}$$

$$diffuse\ reflection = diffuse\ term * material\ colour * light\ colour \qquad \text{[E2]}$$

Height Map

Height map is a grey image which is commonly used for storing value for height efficiently. In this assignment, heightmap will be used to define height at specific coordinate in the terrain. The tessellation coordinates and patch coordinates are provided in the evaluation shader. These values could be used to calculate coordinates for retrieving height from the height map image(E3). We could also multiply the position by some factor to scale up the terrain height. Figure 6 and 7 show the terrain with height map.

$$position = (1-u)\{(1-v)P_1 + vP_3\} + u\{(1-v)P_1 + vP_2\} * factor \qquad \text{[E3]}$$

Height Based Texturing

Now that the terrain has height, we want to texture it based on the height. The rule we use for this application is that if the height is less than H/4 then we set the texture to water. If the height is less than H/2, we set it to grass texture. If the height is less than 3H/4, we set it to rock texture and if the value is less or equal to H then we set it to snow texture. See Figure 3 for different textures based on height.

The given height map does not have completely flat area however water needs to be flat. To achieve this, the height of the coordinate found inside the evaluation shader is adjusted to certain height if it is below the threshold. See Figure 2 and 3 for water flat surface.

Multi-texturing

The surface of the terrain may look rough when changing from one texture to another. To solve this problem, the concept of texture weight is used. The texture weight vector has value between 0 to 1 for each component. The value for the component is used to determine how much of each texture to use. For instance, if the texture weight is (0,0,0.8,0.2), it means 0.8 of texture should use rock texture and 0.2 of texture should use snow texture. In the geometry shader, we find the texture weight for each vertex based on the vertex height. We then pass the value to the fragment shader where each texture is multiplied by the texture weight. Finally, the textures are summed together. Figure 6 shows the smooth surface when we go from rock to grass. Other than smoothen the surface from one texture to another, multi-texturing could also be used to darken and lighten the colour. An example of this could be found in the fragment shader where the water is mixed with another colour to gain a lighter colour of the water texture.

Terrain Level of Detail (LOD)

The terrain surface should be more smooth up close and rougher further away. In other word, the tessellation level should be higher when up close. To achieve this, we set the camera position to look down the terrain in an angle. The camera position is then passed to vertex shader as uniform variable. In vertex shader, the distance between the camera and the average coordinate of current patch is then calculated. The calculated value(d) along with minimum distance between camera and current patch known as dmin, the maximum distance known as dmax, the lowest tessellation level to render known as Llow and the highest tessellation level to render known as Lhigh are used in the equation(E4) to determine tessellation level. Figure 4 shows the triangles near the top of the scene looks bigger than the quad near the bottom of the scene. The tessellation is dynamically update when we move from one position to another. To see this in the program, use F1 to turn on Polyline mode then slowly moving camera forward with down arrow key and backward with up arrow key to observe the changes in tessellation level.

$$level = \frac{d - d_{min}}{d_{max} - d_{min}} (L_{low} - L_{high}) + L_{high} \quad \text{[E4]}$$

Tessellation Optimisation

We know that the water supposes to be flat. This means high tessellation is not needed. To optimize the program, we could checks the height of the patch that is received in the control shader then if it is higher the threshold, we set the tessellation level to be very low. See Figure 4 for an example.

Screenshots

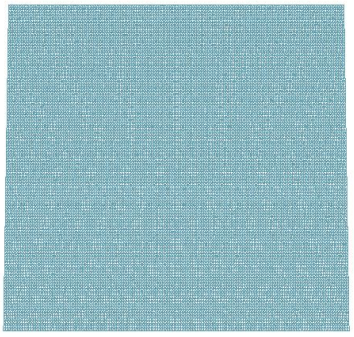

Figure 1: Terrain without height map

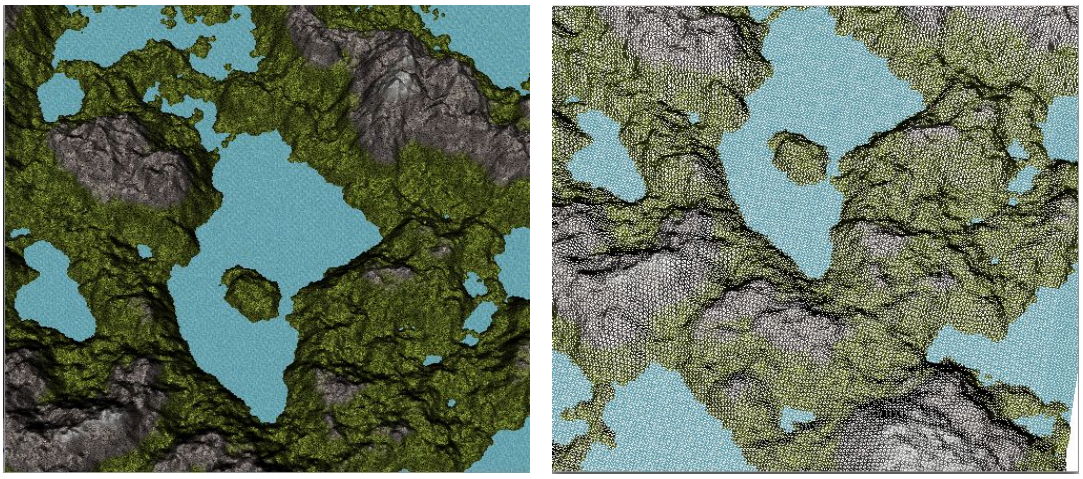

Figure 2 and Figure 3: Terrain with height map with Polyline mode on and off

Figure 5: Terrain with before and after lighting

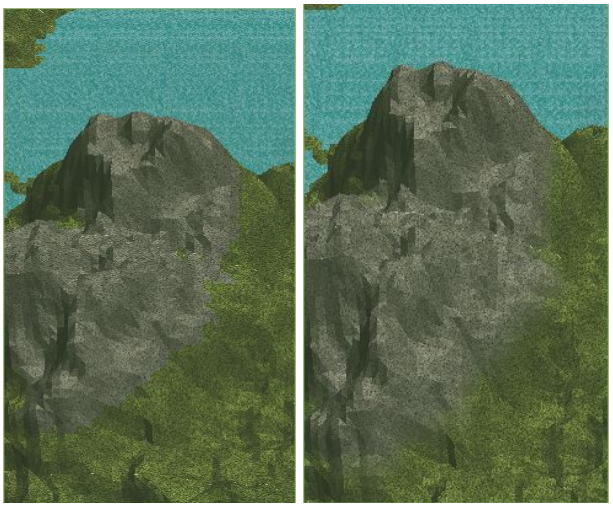

Figure 6: Terrain before and after multi-texturing

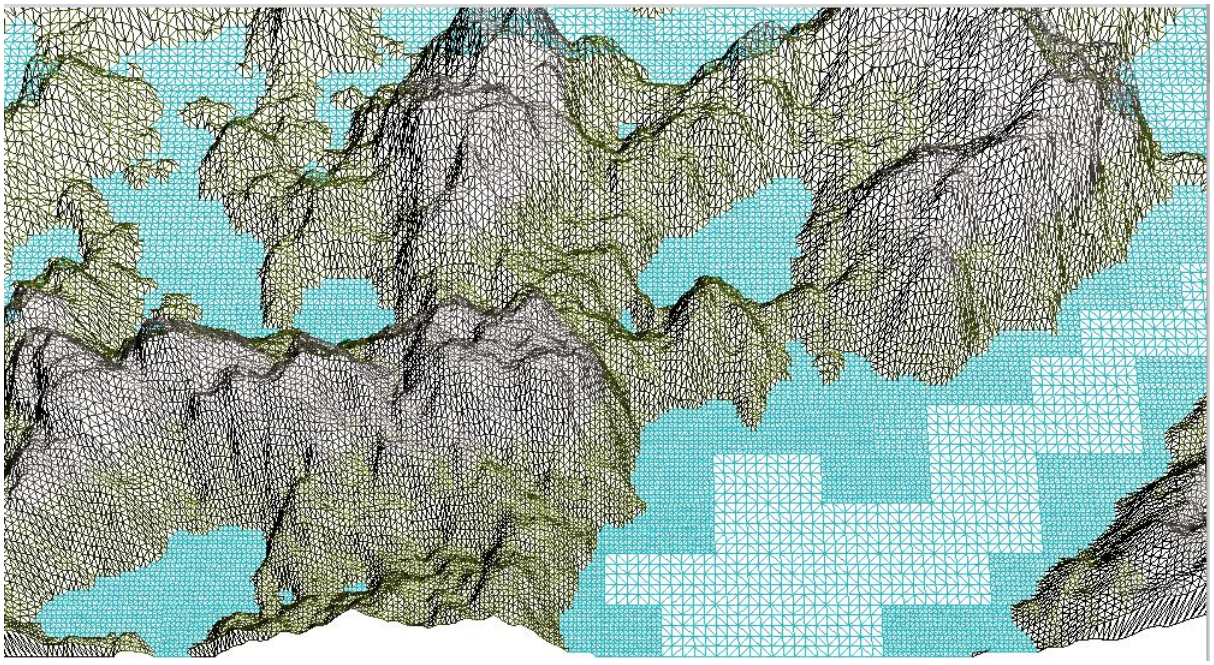

Figure 4: Terrain with LOD and tessellation optimization

Implemented files and keyboard functions

Implemented files
TerrainBaseDraw.cpp
TerrainBase.vert
TerrainBase.cont
TerrainBase.eval
TerrainBase.geom
TerrainBase.frag

Keyboard key	Function
Up arrow	Move camera backward
Down arrow	Move camera forward
Left arrow	Rotate around the terrain
Right arrow	Rotate around the terrain
F1	Switch on Polyline mode
F2	Switch off Polyline mode
F3	Zoom out of terrain
F4	Zoom in to terrain

References

https://www.textures.com/
http://www.informit.com/articles/article.aspx?p=2120983
http://victorbush.com/2015/01/tessellated-terrain/