Winter 20

Procedural Terrain Generation

(Version II)

CS557 FINAL PROJECT

COMPUTER GRAPHICS SHADERS

Submitted By: **Akash Agarwal**
OSU ID Number: **933-471-097**

Contents

1. Introduction

This file describes the implementation details of CS557 Computer Graphics Shaders- Final Project. I've chosen to perform **Procedural Terrain Generation** for the final project.

This contains implementation of a 3-D mesh such that the vertices are connected in a way that they represent an actual terrain. Procedural terrain implies that the object is not static and is subject to change in term of un-evenness and ruggedness in the real-time by user controls. The elevation of each of the vertex on the mesh would depend upon a pre-defined Noise function value for that coordinate.

The following section contains the requirements mentioned in the Project proposal.

2. Project Proposal

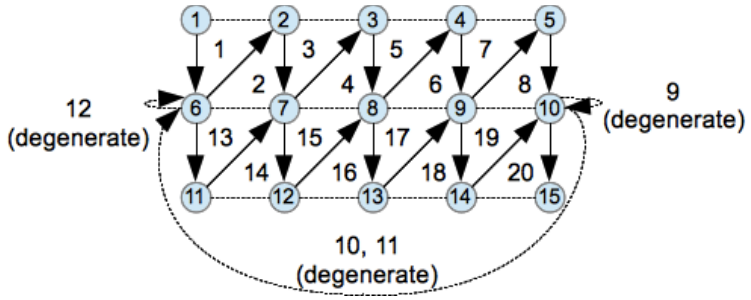
- I had created Terrain Generation project as a part of CS550 Final Project and encountered problems with real-time tessellation speed, lighting and coloring when using the convention OpenGL library. I'll use shaders to solve the aforementioned problems.
- In this model, I'll create a 3-D model of a terrain, that is made up of grid of points arranged in rows and columns and connected using triangle-strips.
- I'll also provide lighting conditions in the scene (mostly ambient light). Now, for the terrain to show effects of lighting, I'll calculate the vertex normal for each of the vertices on the grid.
- I'll provide options to the user to change the elevation and ruggedness of terrain by internally changing the noise function that is generating the terrain.
- Coloring/ texturing vertices based on the height of each vertex. (using smooth-step function)
- I'll be using tessellation/ geometry shader to increase the tessellation of the grid.

3. Current Project Status

The project is fully implemented and meets all the proposed requirements. Each of the implemented things are discussed below:

- **TRIANGULAR MESH**

  This implementation completes the first requirement for the project. For a m x n dimensional size of a terrain, each of the vertex that lies on the terrain is connected in the following manner:



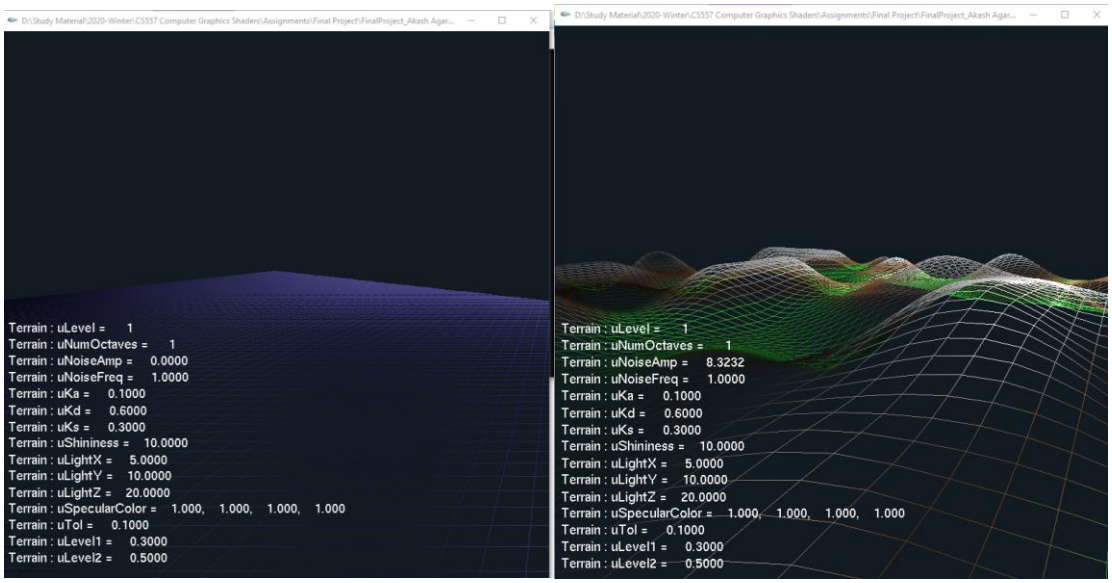
  As shown in the figure, the mesh is made-up of groups of small triangles. The main reasons for connecting the vertices in this manner are:

- Glman program supports Triangle predefined keyword for creating triangle geometry. Although, use of triangular mesh would have been much more efficient, this feature in Glman offers ease in code writing.
- Making real-time changes to the dynamic terrain structure is much faster when terrain is implemented using triangle-strip mesh.
- The lighting conditions would be better with triangular-mesh, especially the flat shading.

I wrote a separate code snippet that prints out the glib mesh information for me to copy into the .glib file. The code snippet is as follows:

```
#include <iostream>
#define size 50

int main()
{
  for(int i = -size; i < size - 1; i++)
  {
    for(int j = -size; j < size - 1; j++)
    {
      // vertices of triangle: (i, 0, j); (i, 0, j + 1); (i + 1, 0, j + 1)
      std::cout << "Triangles [" << i << " 0 " << j << "] [" << i << " 0 " << j + 1 << "] [" << i + 1 << " 0 " << j + 1 << "]\n";

      // vertices of triangle: (i, 0, j); (i + 1, 0, j + 1); (i + 1, 0, j)
      std::cout << "Triangles [" << i << " 0 " << j << "] [" << i + 1 << " 0 " << j + 1 << "] [" << i + 1 << " 0 " << j << "]\n";
    }
  }
  return 0;
}
```

The mesh as implemented in the project is shown below:



- **TERRAIN LIGHTING**

  The script provides the user of manipulating the lighting conditions on the terrain (Ambient, Diffuse and Spectral lighting). It was important to calculate normal for each vertex, to make a provision for smooth shading of terrain. Without normal calculation, the terrain would look like a 2D-blob.

  The following steps were performed to calculate normals:

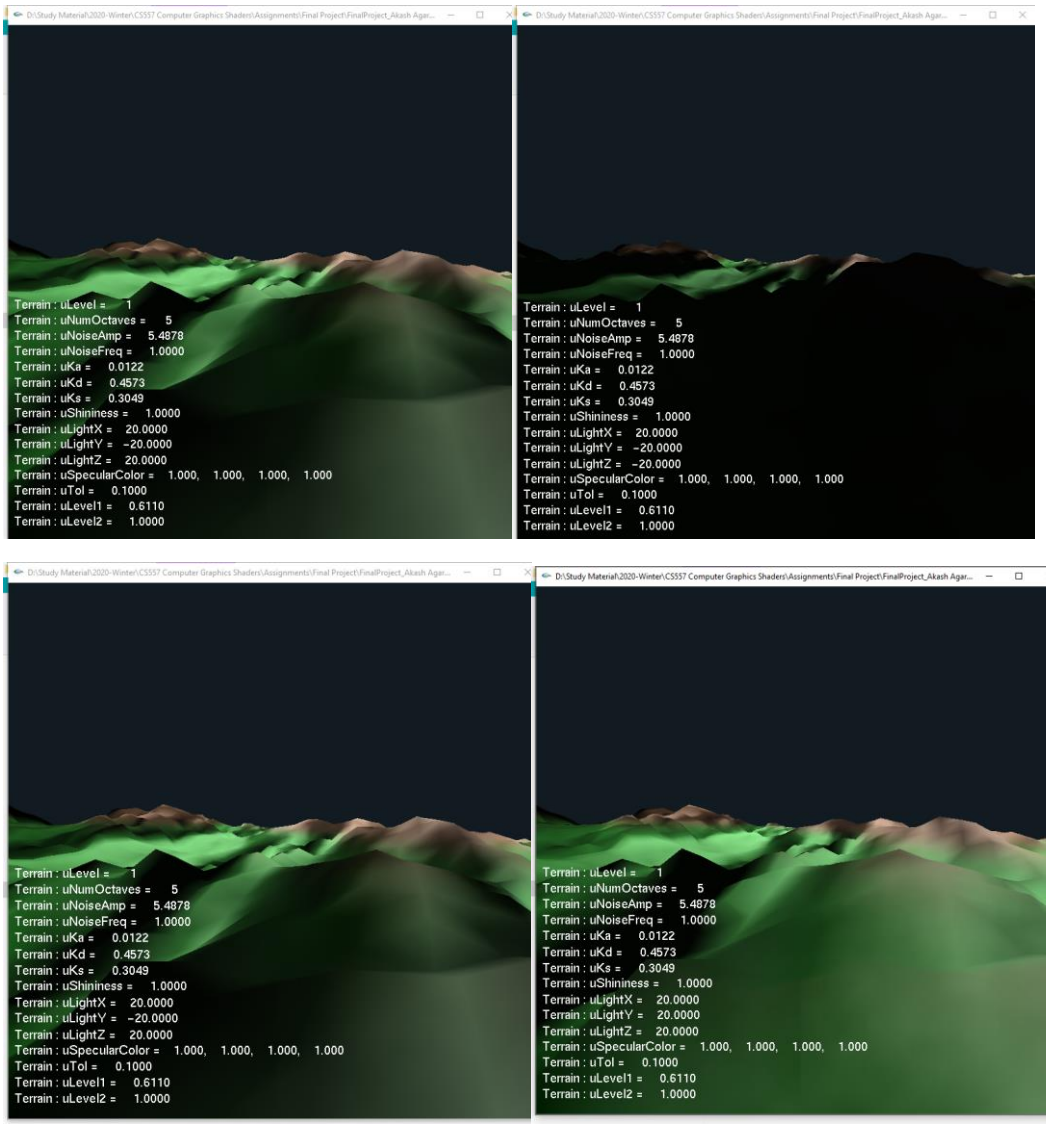
  - Calculation of Vertex Normals: The calculation of vertex normals is more complicated than calculation of surface normal. According to theory, we need to take average of surface normals of all the surfaces common to a vertex and assign that normal to the vertex. When we do that for every vertex, with smooth shading enabled, the terrain object will show lighting conditions.

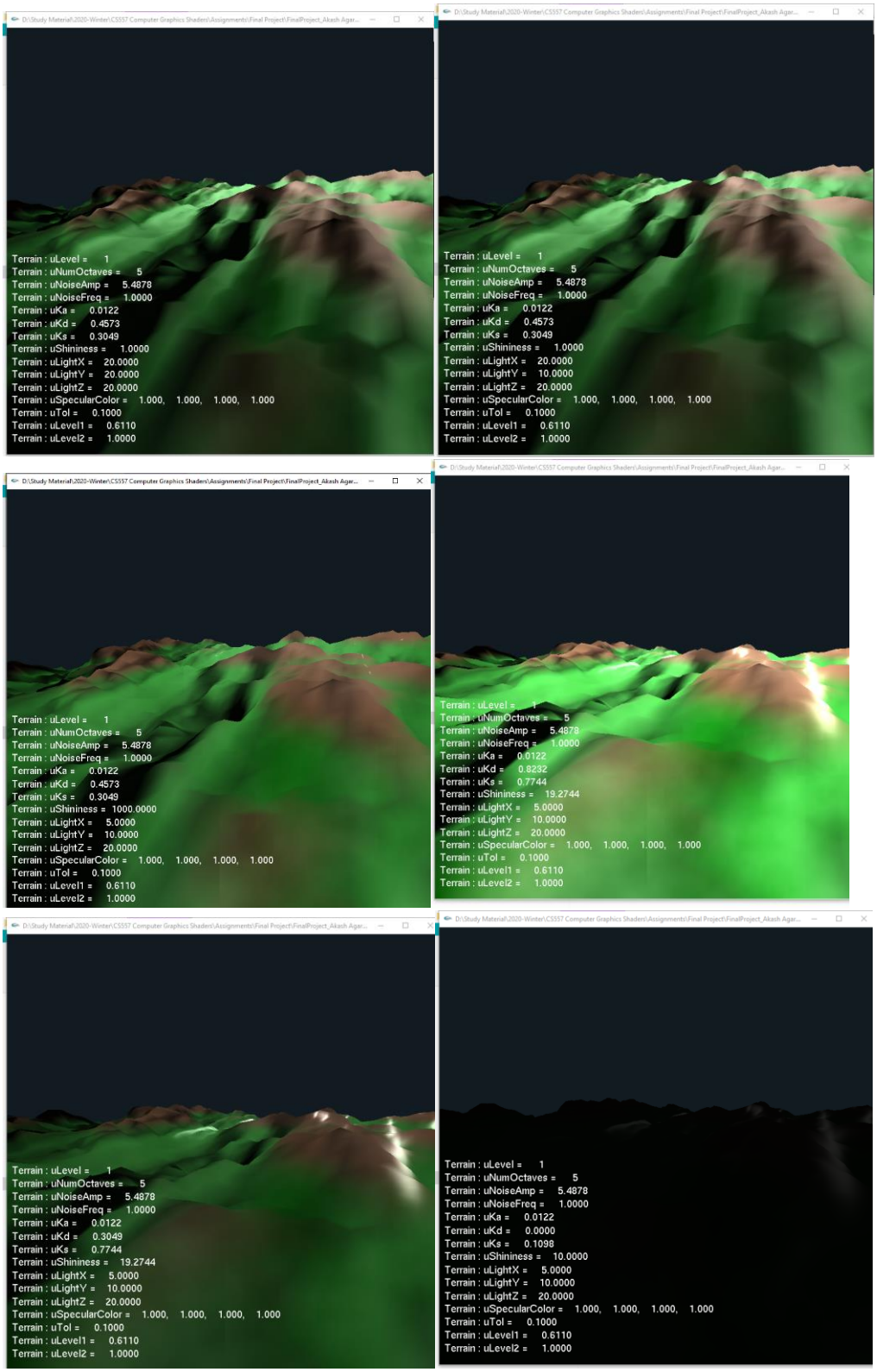
    For the terrain, the surface normal values stand (0, 1, 0) initially, when the height values of each of the vertices is 0 units, but the normals are subject to change because the height of every vertex would be calculated using the noise function. Since all the calculations of normals were done in the geometry shader, it becomes rather unnecessary to keep track of the surface normal. Instead, for each vertex, we can calculate the two vectors, one pointing in X-axis direction and the other one pointing in Z-axis direction. The cross-product of these vector can serve as the required vertex normal. Moreover, to calculate the above mentioned vectors, we can take an offset vector to calculate the same.

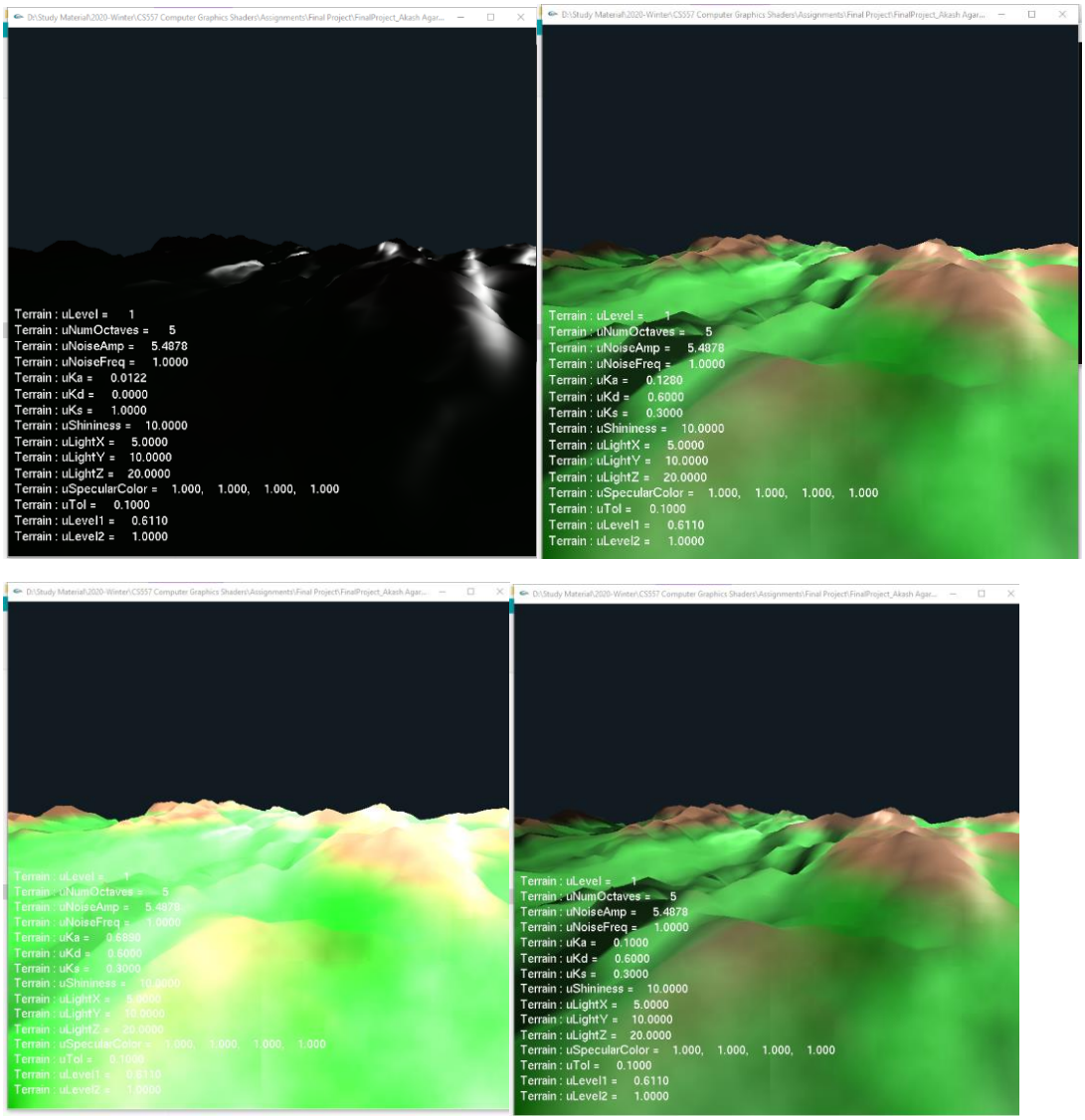
    The following code snippet provides a good approximation for vertex normals:

```
vec3 getNormal(vec3 pos)
{
    vec3 off = vec3(1.0, 1.0, 0.0);

    float hL = octaves(pos.xy - off.xz);
    float hR = octaves(pos.xy + off.xz);
    float hD = octaves(pos.xy - off.zy);
    float hU = octaves(pos.xy + off.zy);

    // deduce terrain normal
    gNormal.x = hL - hR;
    gNormal.y = hD - hU;
    gNormal.z = 2.0;
    gNormal = normalize(gNormal);
    return gNormal;
}
```

Thus, the proposed requirements for lighting are met in the project.


Terrain : uLevel = 1
Terrain : uNumOctaves = 5
Terrain : uNoiseAmp = 5.4878
Terrain : uNoiseFreq = 1.0000
Terrain : uKa = 0.0122
Terrain : uKd = 0.4573
Terrain : uKs = 0.3049
Terrain : uShininess = 1.0000
Terrain : uLightX = 20.0000
Terrain : uLightY = -20.0000
Terrain : uLightZ = 20.0000
Terrain : uSpecularColor = 1.000, 1.000, 1.000, 1.000
Terrain : uTol = 0.1000
Terrain : uLevel1 = 0.6110
Terrain : uLevel2 = 1.0000

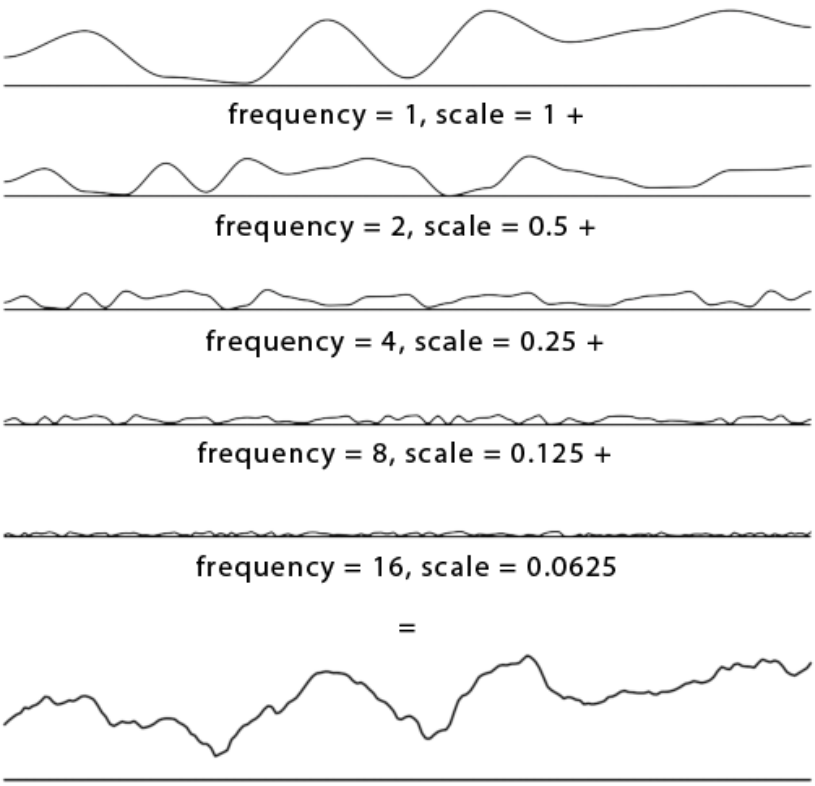

Terrain : uLevel = 1
Terrain : uNumOctaves = 5
Terrain : uNoiseAmp = 5.4878
Terrain : uNoiseFreq = 1.0000
Terrain : uKa = 0.0122
Terrain : uKd = 0.4573
Terrain : uKs = 0.3049
Terrain : uShininess = 1.0000
Terrain : uLightX = 20.0000
Terrain : uLightY = 20.0000
Terrain : uLightZ = 20.0000
Terrain : uSpecularColor = 1.000, 1.000, 1.000, 1.000
Terrain : uTol = 0.1000
Terrain : uLevel1 = 0.6110
Terrain : uLevel2 = 1.0000
Terrain : uLevel = 1
Terrain : uNumOctaves = 5
Terrain : uNoiseAmp = 5.4878
Terrain : uNoiseFreq = 1.0000
Terrain : uKa = 0.0122
Terrain : uKd = 0.4573
Terrain : uKs = 0.3049
Terrain : uShininess = 1.0000
Terrain : uLightX = 20.0000
Terrain : uLightY = 10.0000
Terrain : uLightZ = 20.0000
Terrain : uSpecularColor = 1.000, 1.000, 1.000, 1.000
Terrain : uTol = 0.1000
Terrain : uLevel1 = 0.6110
Terrain : uLevel2 = 1.0000
Terrain : uLevel = 1
Terrain : uNumOctaves = 5
Terrain : uNoiseAmp = 5.4878
Terrain : uNoiseFreq = 1.0000
Terrain : uKa = 0.0122
Terrain : uKd = 0.4573
Terrain : uKs = 0.3049
Terrain : uShininess = 1000.0000
Terrain : uLightX = 5.0000
Terrain : uLightY = 10.0000
Terrain : uLightZ = 20.0000
Terrain : uSpecularColor = 1.000, 1.000, 1.000, 1.000
Terrain : uTol = 0.1000
Terrain : uLevel1 = 0.6110
Terrain : uLevel2 = 1.0000
Terrain : uLevel = 1
Terrain : uNumOctaves = 5
Terrain : uNoiseAmp = 5.4878
Terrain : uNoiseFreq = 1.0000
Terrain : uKa = 0.0122
Terrain : uKd = 0.8232
Terrain : uKs = 0.7744
Terrain : uShininess = 19.2744
Terrain : uLightX = 5.0000
Terrain : uLightY = 10.0000
Terrain : uLightZ = 20.0000
Terrain : uSpecularColor = 1.000, 1.000, 1.000, 1.000
Terrain : uTol = 0.1000
Terrain : uLevel1 = 0.6110
Terrain : uLevel2 = 1.0000
Terrain : uLevel = 1
Terrain : uNumOctaves = 5
Terrain : uNoiseAmp = 5.4878
Terrain : uNoiseFreq = 1.0000
Terrain : uKa = 0.0122
Terrain : uKd = 0.3049
Terrain : uKs = 0.7744
Terrain : uShininess = 19.2744
Terrain : uLightX = 5.0000
Terrain : uLightY = 10.0000
Terrain : uLightZ = 20.0000
Terrain : uSpecularColor = 1.000, 1.000, 1.000, 1.000
Terrain : uTol = 0.1000
Terrain : uLevel1 = 0.6110
Terrain : uLevel2 = 1.0000
Terrain : uLevel = 1
Terrain : uNumOctaves = 5
Terrain : uNoiseAmp = 5.4878
Terrain : uNoiseFreq = 1.0000
Terrain : uKa = 0.0122
Terrain : uKd = 0.0000
Terrain : uKs = 0.1098
Terrain : uShininess = 10.0000
Terrain : uLightX = 5.0000
Terrain : uLightY = 10.0000
Terrain : uLightZ = 20.0000
Terrain : uSpecularColor = 1.000, 1.000, 1.000, 1.000
Terrain : uTol = 0.1000
Terrain : uLevel1 = 0.6110
Terrain : uLevel2 = 1.0000



- **PERLIN NOISE AND DYNAMIC TERRAIN:**

  The project uses the Perlin Noise function to generate the height of each vertex. For every vertex, the x and z coordinates are provided as an input to the noise function and the function return a float value, that can act as the y coordinate, i.e. the height of the vertex. Since, the returned value depends upon the distance, the height values can vary greatly among vertices, leading to generation of crests and troughs. This makes the mesh uneven and provides the feeling of terrain. The noise function was implemented in the geometry shader to manipulate the height values of each vertex and the same noise function is also defined in the fragment shader to change the color of the corresponding pixel.

  One of the proposed requirements for the project was to make the terrain dynamic, i.e. to give the user controls to change the height of the crests and increase/ decrease the ruggedness of the terrain. The noise octaves work as shown in the image below:



Thus, the project also meets the requirement for dynamic terrain.

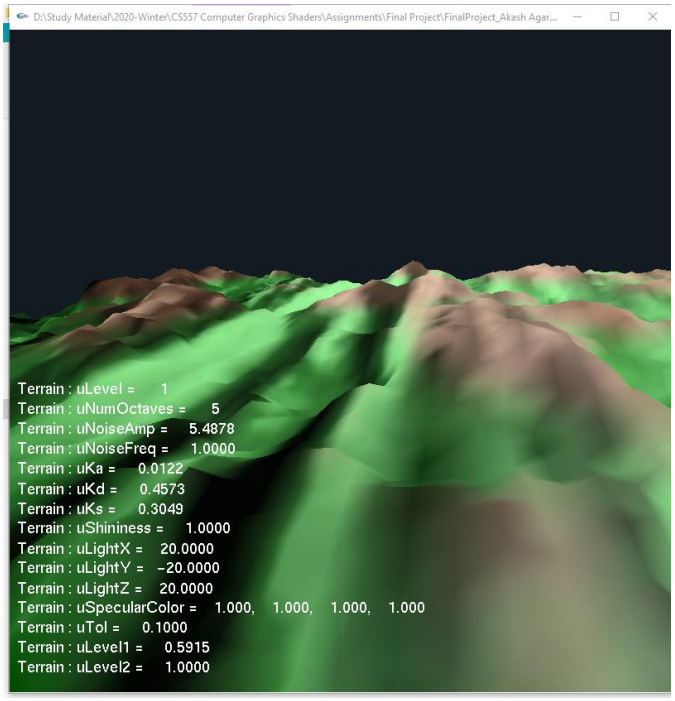
- **TERRAIN COLORING**

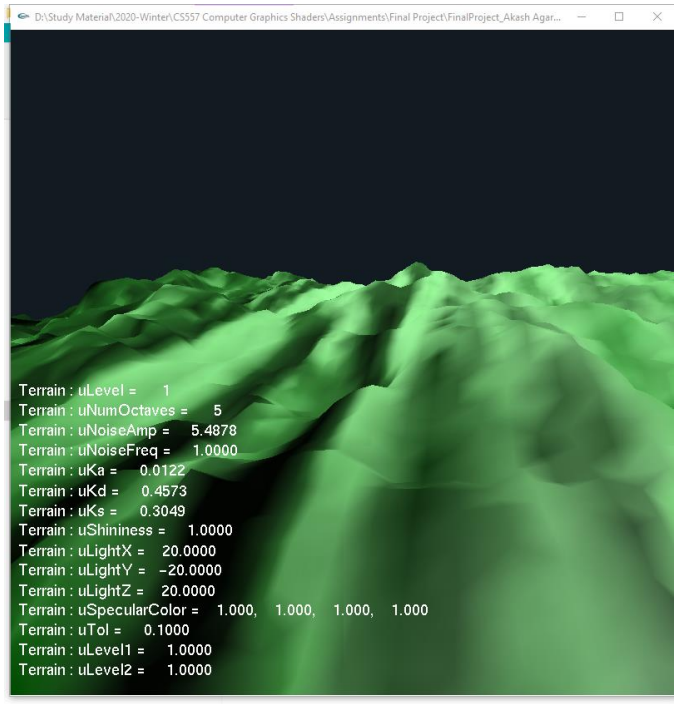
  In the project, the colors are applied to the terrain gradually according to the height of the vertex. The program pseudo-normalizes each of the height values between 0 and 1 when colors for the vertex are calculated. This methodology provides a gradual color shift to the terrain. The color values applied on the terrain according to the height are:

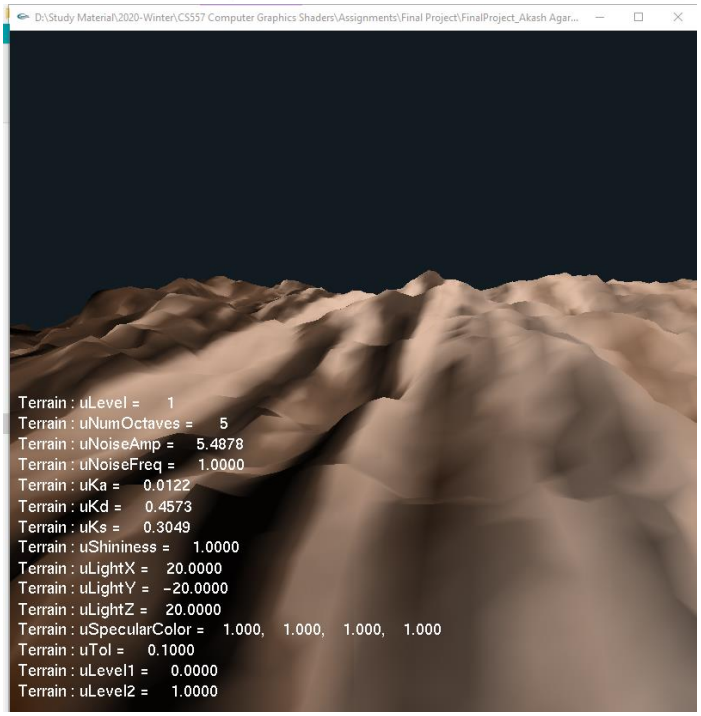
```
const vec3 BLUE = vec3( 0.1, 0.1, 0.5 );
const vec3 GREEN = vec3( 0.0, 0.8, 0.0 );
const vec3 BROWN = vec3( 0.6, 0.3, 0.1 );
const vec3 WHITE = vec3( 1.0, 1.0, 1.0 );

vec3 getColor(float height)
{
    height = height/ 10.;

    vec3 color = BLUE;
    if( height > 0. )
    {
        float t = smoothstep( uLevel1-uTol, uLevel1+uTol, height );
        color = mix( GREEN, BROWN, t );
    }
    if( height > uLevel1+uTol )
```

```
    {
        float t = smoothstep( uLevel2-uTol, uLevel2+uTol, height );
        color = mix( BROWN, WHITE, t );
    }
    return color;
}
```

The results of the coloring method can be seen in the figure below:







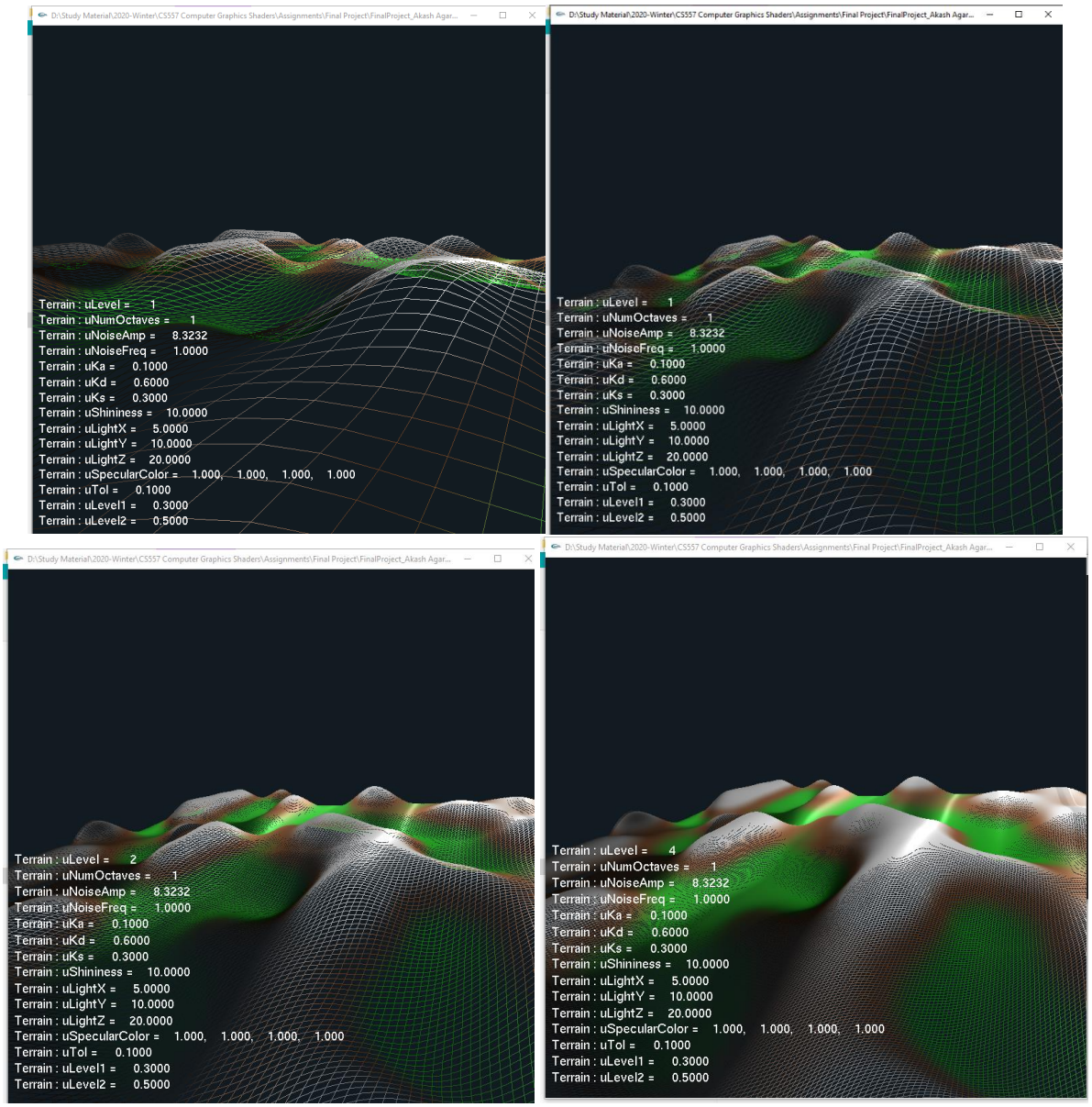
Thus, the project also meets the proposed requirements for coloring.

- **TESSELLATION USING GEOMETRY SHADERS**

In this project, the geometry shader provides the tessellation feature. Generation of new vertices works like the sphere subdivision method we studied in class.

The glib file mentions a uniform variable uLevel that divides each triangle into different levels.

The screenshots are shown below:



4. Differences from Proposed Requirements

Although all the requirements have been met, there are 2 subtle differences.

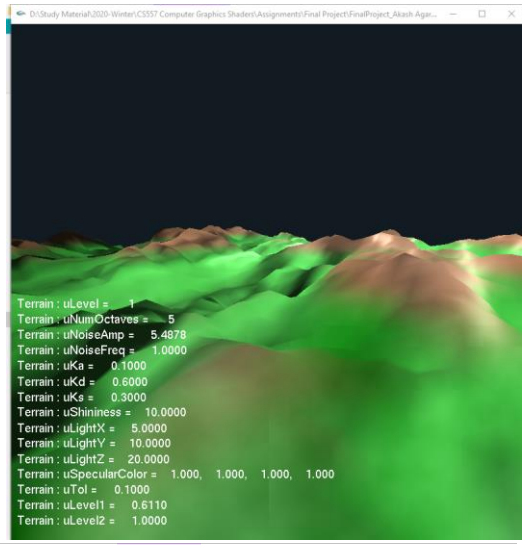
- It was proposed that the tessellation would be done using the Tessellation shaders, but in the interest of time, it was implemented using the Geometry Shaders.
- The implementation was done using the Glman program instead of the proposed GLSL API.

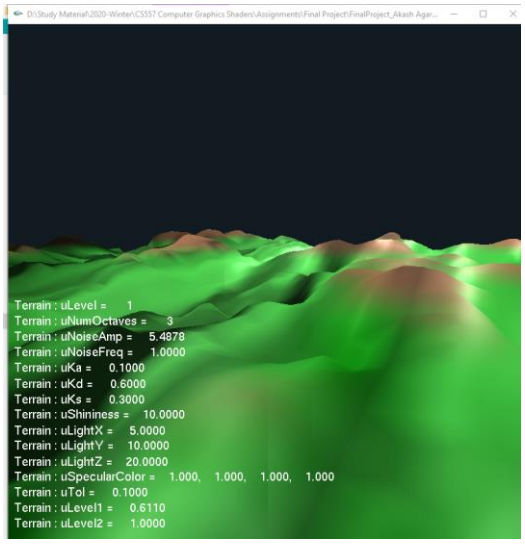
All the features were implemented, and the above mentioned minor implementation changes do not lead to any functionality changes.

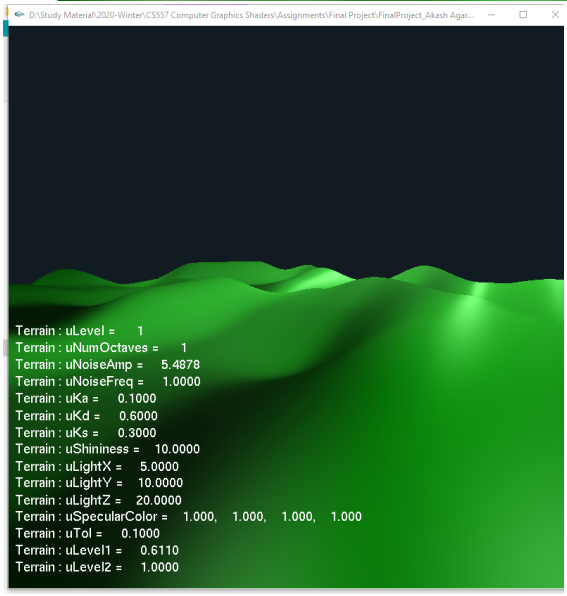
5. Concepts Learnt

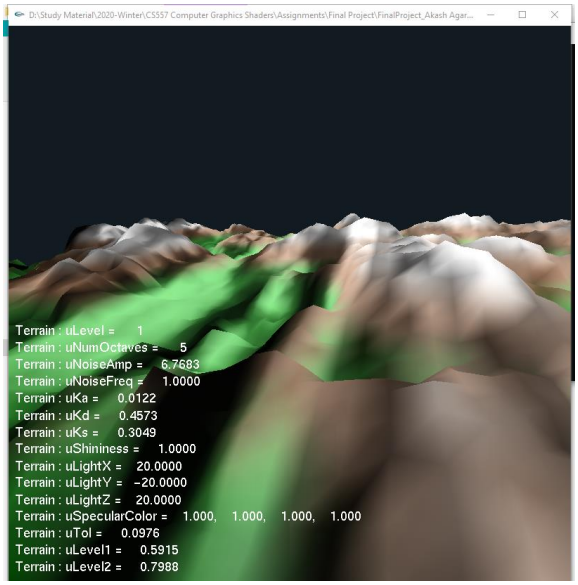
- Use of geometry shaders for generating more vertices.
- Approximated calculations for vertex normals.
- Lighting calculations using shaders.
- Learnt the importance of vertex normals in terrains. The lighting on terrain looked horrible with flat shading as some of the triangular surface would turn black and the others would show color. The actual blend of shading comes using vertex normals only.
- Learnt how to calculate vertex normals for each vertex in a triangle-strip mesh.
- Learnt a little about Perlin Noise.
- Learnt about Noise Octaves and how to control them.

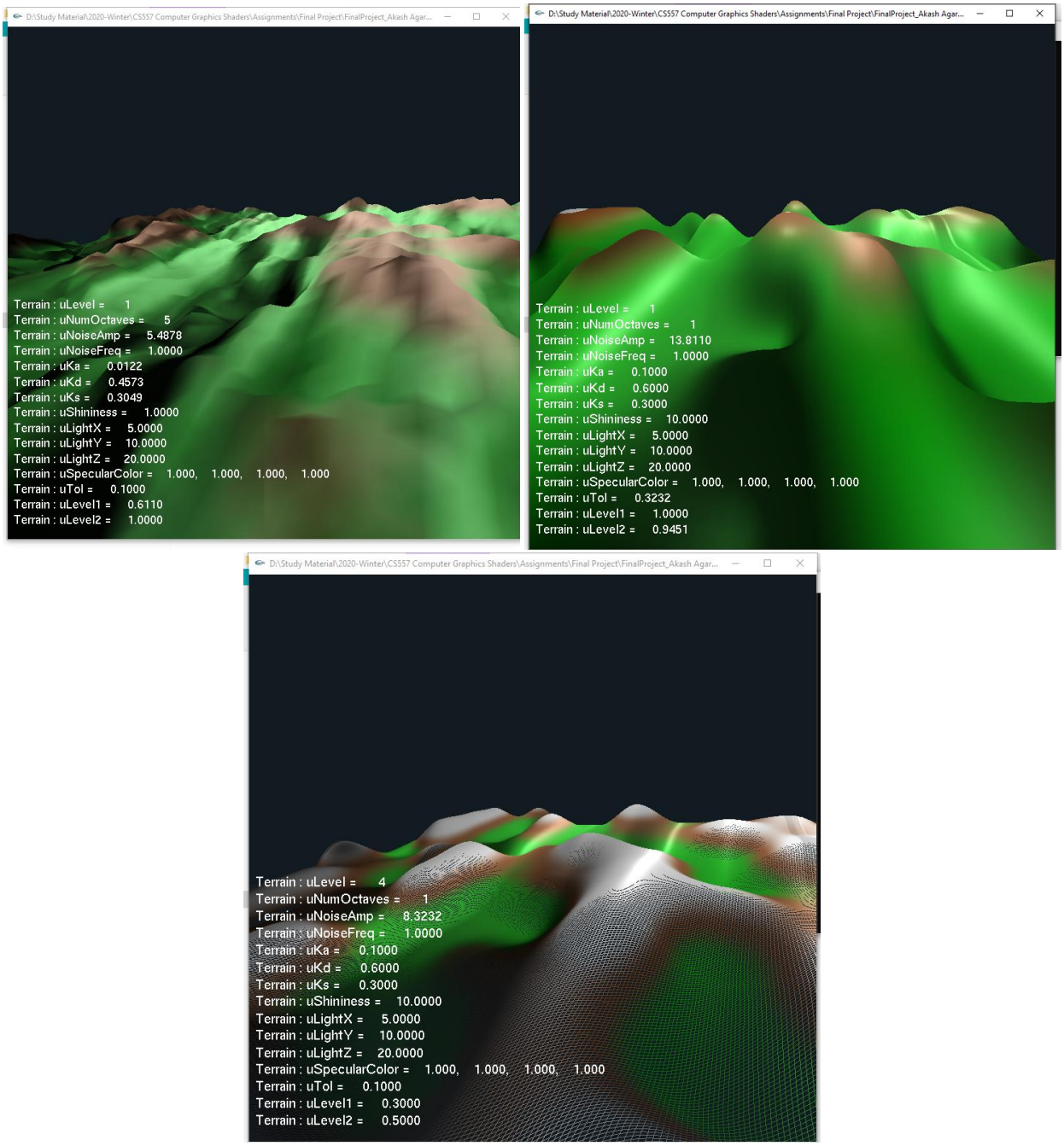
6. Terrain Images











7. Video Link

This is the link to the video: https://media.oregonstate.edu/media/t/0_4ju61ckf