Tommy Bacher and Colby Jeffries

Computer Science 300 Project 4

Fractal Terrain Generation

Tommy’s Section:

In terms of division of labor, Colby and I have found that we work best if a majority of the work is accomplished first together, and then we split off and focus on tuning up the parts of projects that are not fully finished. As such we met in Taylor and worked together on a template for projects that Colby and I have created. We used a git repository to store and transfer the project thus allowing us to better keep track of the changes each person made while also allowing us to modify the project from multiple different computers. Each of us had our primary focuses which I will discuss now.

For Project Four, I ended up focusing most of my work on setting up the classes and storage structures necessary to make the project run. I focused on building the fractal terrain class which was mostly ported from the java code that was made available on the website. My first job was to translate all the Java based functions into C++ based functions which was mostly a matter of removing the public and private designations while also creating prototypes in the class that correspond to the functions in the .cpp file. Another point of concern was the random number generator which needed to be translated to use the rand() function in the Math library. This was achieved by simply change the rng() function to rely on the rand() function instead of the number generator that java supports. The most difficult thing was how to manage the array because, unlike java, in C++ true arrays cannot have dynamic size. Originally a double pointer was considered, but issues with memory allocation arose and I decided to use a vector of vectors to represent the terrain. This change was easy to implement in the Java code as vectors can be modified in the same way as arrays and thus required no additional work.

I also worked on integrating the separate classes together to create the terrain generation. Each class was also pulled from the website as java code and translated into C++ code to be used in the project. The RGB class was able to support the colorization of the terrain through representing an OpenGL color using three integers to represent the red, green, and blue components respectively. The Triangle and Triple classes were converted from Java to C++ by Colby with my help during our work in Taylor.

Colby and I collectively solved problems during the course of this project. A consistent issue that we ran into was related to the way arrays function in C++ versus java. Since most code provided was Java code, vectors were the primary C++ means of generating dynamically sized arrays for use in this project. Consistently, Colby’s compiler would not flag errors related to a C++ array being initialized with a non-constant variable, but my compiler would and thus we’d have to pass the code back and forth working on an implementation that relied on vectors rather than arrays.

As for OpenGL, much of the work was done collectively with Colby and I drawing from our other projects and labs to generate the code necessary to display the terrain based on the generation in the generateTerrain function. Our initial focus was on getting the terrain to appear before doing anything else. There were some problems related to the conversion of classes from Java to C++ specifically with the Triple and RGB class that only became apparent when working with OpenGL, but once solved drawing the terrain was a fairly simple task.

Colby was the one to do most of the work implementing the lighting into the scene, and he did an excellent job of creating proper shadows that brought the terrain to life. He likely discusses it in his section that his implementation draws heavily from previous code provided on lighting and shadows. Colby also implemented the terrain rotation which is a simply port from our previous projects that simply requires wrapping the entire OpenGL drawing in a glRotate with an increasing angle parameter.

Once the terrain was complete we added a few extra features to further explore the way the diamond-square algorithm generates terrain. The level of detail, LOD, can be lowered or increased to change the detail that is put into generating the terrain. At lower levels the terrain has a more obvious downward slope and a more pixelated look whereas at higher levels of detail the terrain looks almost life-like. Colby added an adjustment to the roughness of the terrain which controls how much potential range there is between the lowest valleys and the highest peaks. Particularly of note is that when the roughness is increased to high levels at high levels of detail the terrain begins to look more like a sound wave captured in a moment rather than actual terrain, with huge peaks leading into small valleys. Water implementation was simply a matter of finding the point where the green switches to white and then blue, and placing a series of OpenGL vertices to act as a water plane. In particular, with water on at high levels of detail and medium roughness, occasionally the terrain looks like an archipelago which is exquisite.

Colby’s Section:

In terms of work division, this project was done almost entirely together. Tommy and I met in Taylor and knocked out a good majority of the project in a single day. Using the provided Java code as basis and inspiration, we created most of the needed classes and the code to calculate the heights, colors, and normals of the tessellated triangles. Tommy and I discussed how to display the land, and then I implemented it. I also added the water, and “pillar of ground” underneath the terrain. As a final touch to my work I added the menu options for water, LOD, reset, roughness, water level, and exaggeration (scaling factor). As always, we did equal shares of work and used Git to keep track of everything and allow us to work at the same time.

As for what we did in the lab, we heavily used the provided Java code in order to create a polished final product with the little time we had. We used square-diamond algorithm initially because we had it in front of us. Ultimately, we found that it looked really nice. It created realistic looking land forms, and we decided to keep it. We selected our colors in hopes of making realistic looking lands. The top color is a bluish grey, which looks a lot like a mountain top. The center green is a darker green to represent forests, which fades to blue as it gets closer to the ocean, which is a deep blue-green.

The default settings are an LOD of 5 and a roughness of 0.5. These settings yield a video-gamey look, and with water enable, it looks like a tropical coast. Increasing the roughness makes the terrain look more mountainous to an extent, and then the land starts to look more like grass. Increasing the LOD generates more detailed terrain, and when it is set to 7, it really looks like a realistic coastline. Lower LOD generate more and more basic land forms, with less than 4 looking almost like building blocks.

The ground below the terrain was generating by creating a polygon for every two points going along the edge to down below where the camera can see. This gives the effect that the terrain is cut out of the ground. Interestingly, water was done in much the same way. First a flat plan at 0.2 across the whole terrain. This gets most of the look of the water, be doesn’t look good around the edges. To remedy this, for every two points around the edge that are below 0.2, a polygon is created that goes up to 0.2. Later on, I changed this 0.2 value to be adjustable in the water level menu. This again generates the illusion that the terrain was cut out of a larger piece of ground.

Couple of interesting technical things to mention: regular arrays were not usable for our implementation. Because 2D arrays cannot be declared with non-constant variables, we had to use vectors. Ultimately, this is a good thing in terms of programming practice, but we also run into issues with the maximum size, which is why we are limited to an LOD of 7.

In terms of using what I have learned this semester, I can safely say I used it all in the project. This project touches on just about every topic that we have covered in class. We didn’t use GLSL ourselves, but that was definitely room for it in doing lighting. I really liked the final product in this project, it really looks great.

Final thought: I’d prefer to have switched this project with project three. I feel that that project was more open, and allowed more freedom in terms of what we do, and how complicated it could get.