Jack Neus (@jneus)
Lauren von Berg (@lmberg)

COS 426 Final Project Report

"The mountains are calling and I must go." - John Muir

Introduction

The mountains are a truly special place. Whether in the Swiss Alps, the windswept High Sierras of California, or the sprawling Canadian Rockies, mountains remind us of our insignificance as humans. It is only in these alpine environments that we mere humans are able to experience the forces of the Earth in their purest forms.

Our goal for this project was to create a reminder of some of the places that are most dear to us. While our application is no substitute for the real deal, we hope to enable users to study mountains and their environments in order to gain a better appreciation for the alpine and for the ways in which these peaks interact with the Earth and its weather systems.

Presently, the simulation and generation of mountainous terrain and weather systems can be fit into two categories: entertainment-oriented technologies and geoscience-oriented technologies. There is massive demand in the television, movie, and video game industries to algorithmically generate terrain and animate things like weather. In striving to create visually-appealing terrain, however, the entertainment industry sometimes compromises on physical accuracy, creating and animating scenes that have little in common with the real world. On the other hand, the scholarly world has a real need to be able to accurately model weather systems and geological phenomena. While scientific programmers have been largely successful in implementing highly-accurate models of the physical world, these models often lack the cosmetic appeal that is so present in entertainment-focused applications.

The divide between these two classes of products is sizeable: we were unable to find anything that both looked great and accurately reflected the geological and meteorological laws that govern the physical world. As such, the goal of this project was to create what we were unable to find: an interactive alpine experience that is both aesthetically-pleasing and based in reality.

Methodology

Our project has three main components: mountain generation, sky simulation, and weather simulation. Together, these components require a platform that both supports the rendering of meshes and has a particle system. Assignment 5 is one such platform, so we decided to use the generic portions of the assignment code to start our project.

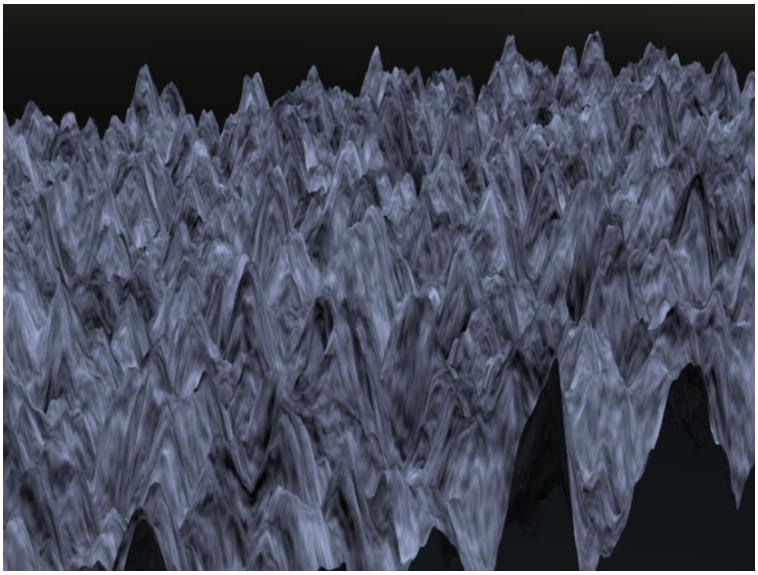
Mountain generation:

We had a few options in regard to our implementation of mountain generation. The first was to manipulate the vertex positions and colors of a mesh with GLSL shaders. This would have been a good approach in order to achieve accurate coloring of the mountains using Perlin noise. However, we decided not to use shaders because they are harder to debug and because

GLSL is much less flexible than JavaScript (something we had painfully learned through previous assignments).

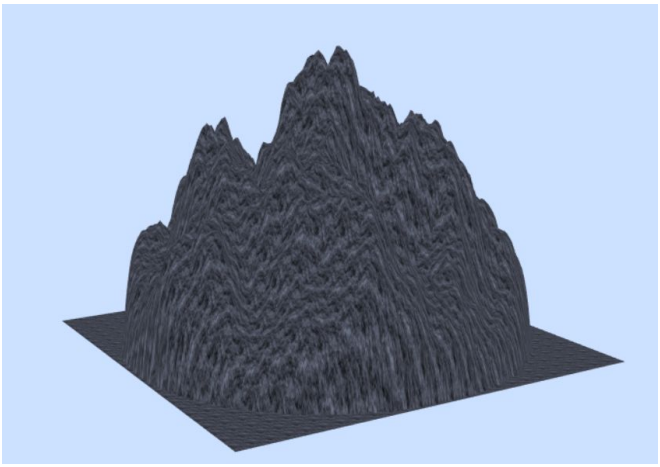
Instead, we decided to create our terrain directly in application code using THREE.js primitives. We used a PlaneBufferGeometry as the overall mesh for the mountainous region, modifying the displacement (height) of the mesh at different locations using a displacement map. Generating a displacement map that accurately reflected the topology of mountains was an involved process, and will be discussed in detail below.

As with many of the existing technologies that generate terrain, we rely on procedural generation to create the basic structure of our mountains. First, we used the Fractal Brownian Motion algorithm to generate a fairly-random noisy displacement map. This algorithm alone created a height map that certainly reflected the high variance of elevation in mountainous regions, but not in a way that truly captured the essence of the peaks we know and love -- instead, it looked more like an endless field of nondescript mountains. We wanted to generate fewer, more impressive peaks.

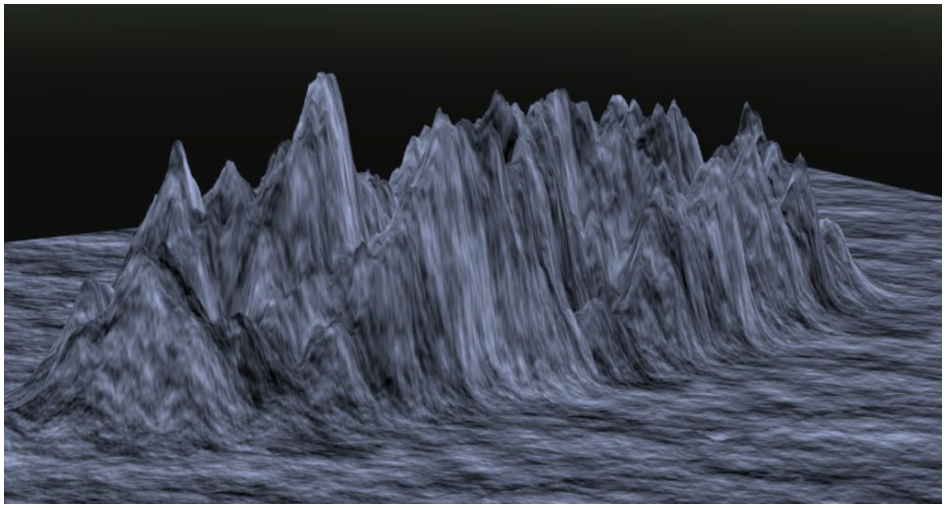


Basic height map with FBM

To accomplish this, we decided to position the mountains at the center of the mesh and decay towards the edges, creating a few prominent central peaks. Mountains often feature increasingly steep dropoffs of bare rock (eventually leading to cliffs) towards their edges. We felt that this property was well captured by the shape of the logarithm curve, so we started experimenting with logarithmic decay. We calculated the bounding ellipse of the mountainous region and applied logarithmic decay at each point in the height map using the point's normalized radial distance from the center of the bounding ellipse. This yielded the following terrain:

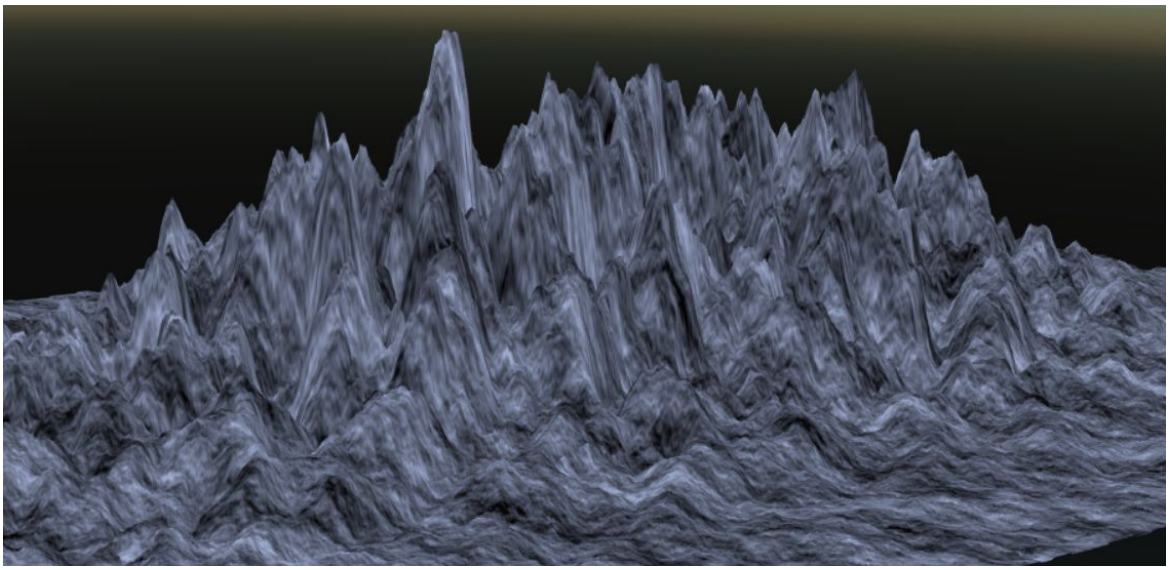


This doesn't look quite right -- while mountains do often "cliff out", they eventually smooth out at the bottom. This smoothing manifests itself physically in talus fields. Talus fields are low-angle slopes of crushed rock that sit at the base of a mountain. We felt that the steepening of a mountain's rock layer combined with presence of the talus slopes was well captured by the sigmoid function (https://en.wikipedia.org/wiki/Sigmoid_function), so we experimented with sigmoidic decay:



Sigmoidic decay

This was much better, although the mountains were still pretty steep and didn't reflect the shape of talus slopes much. To fix this, we explicitly defined a smaller ellipse that was concentric with the bounding ellipse. We apply sigmoidic decay within this ellipse, and decay the terrain outside that ellipse (but inside the bounding ellipse) with the inverse function (1/x), which decreases more gradually than the sigmoid. This yielded the following terrain, which is a pretty good representation of mountain ranges like the Tetons (which is the youngest spur of the American Rockies and therefore the most jagged -- there's been less time for erosion to occur).



Sigmoidic decay until the talus, then inverse decay.

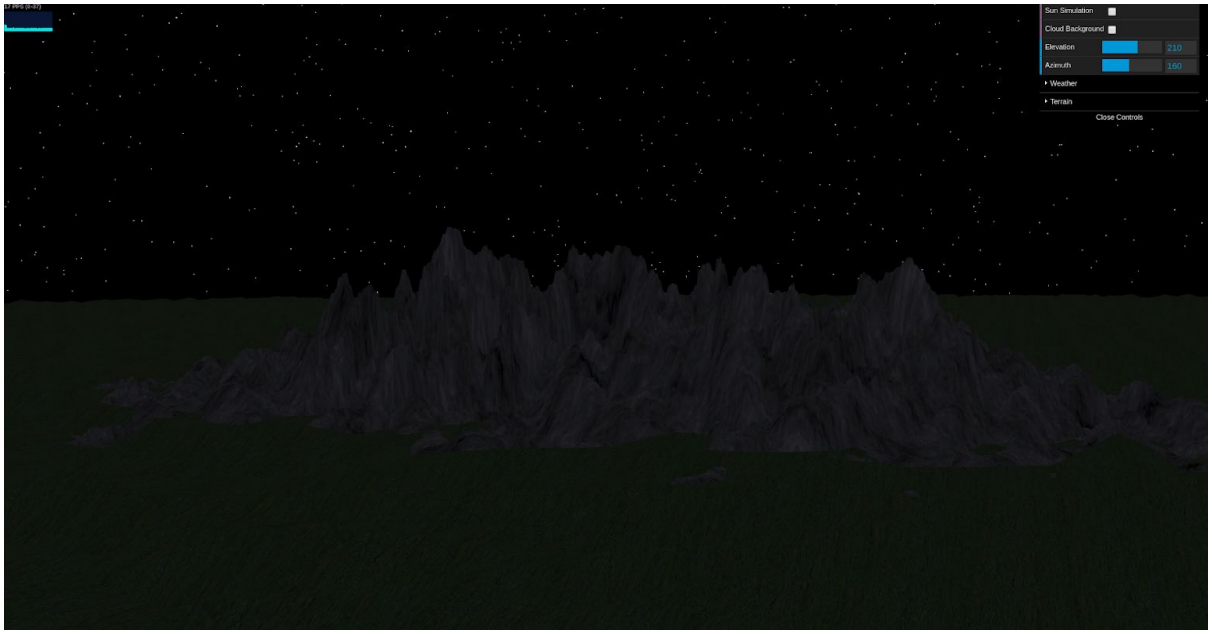
Sky simulation:

We wanted to have a semi-accurate representation of the sky in our world. Specifically, we wanted to account for the atmospheric processes which affect the perceived color of the sky as the sun moves across it. It was also important for us to have a good simulation of the night sky, with stars coming up as the sun went down. At first, we wanted to model the color of the sky during different times of the day ourselves by accounting for Rayleigh and Mie atmospheric scattering. However, these calculations were too complicated for us to do ourselves without in-depth knowledge of the relevant atmospheric science. Therefore, we found a shader which implemented the scientific calculations for us (https://github.com/wwwtyro/glsl-atmosphere). When given the position of the sun and the position of a pixel, it uses real atmospheric and planetary constants to compute the perceived color of the sky at that pixel location. It still took quite a bit of work to apply the shader to a material, but once we figured out how to use ShaderMaterials we were able display our sky on a sphere around our scene and mimic the spherical nature of Earth.

To further augment the sky simulation, we allowed our “sun,” a directional light in the scene, to orbit around the scene. When the sun simulation is running, for every timestep in the renderer we rotate the sun by 0.001 radians and update the sky material uniform for sun position as well as for the light position. This allows us to have a visually-appealing sunset behind the mountains.

Moreover, whenever the sun dips below a certain distance above the horizon, we have stars come out. We could have implemented our stars as a texture or new fragment shader on the sky sphere, however both of these methods made it difficult to get randomness and smooth movement of the stars. Therefore, we decided to implement our stars as a point object in a cloud outside the sky sphere. We generated points uniformly at random in the volume between the sky sphere and a large distance beyond it. This was done to give the appearance of some

stars being smaller than others (because they're farther away). We also rotate our stars by 0.001 radians in every timestep to simulate the rotation of Earth.

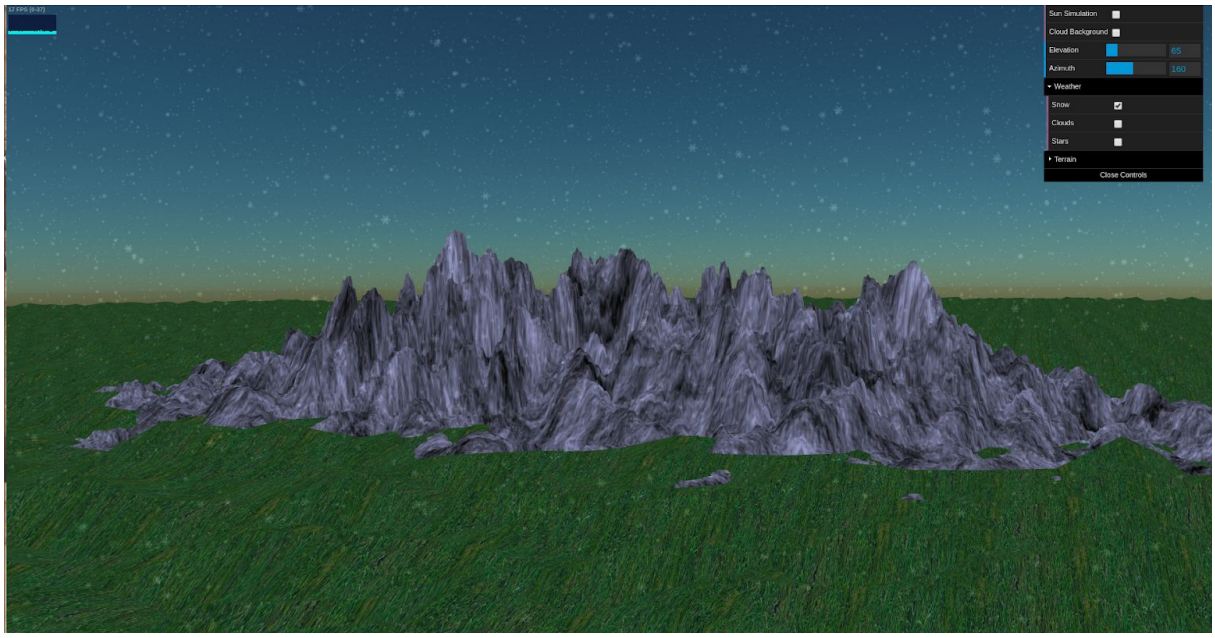


Weather Simulation:

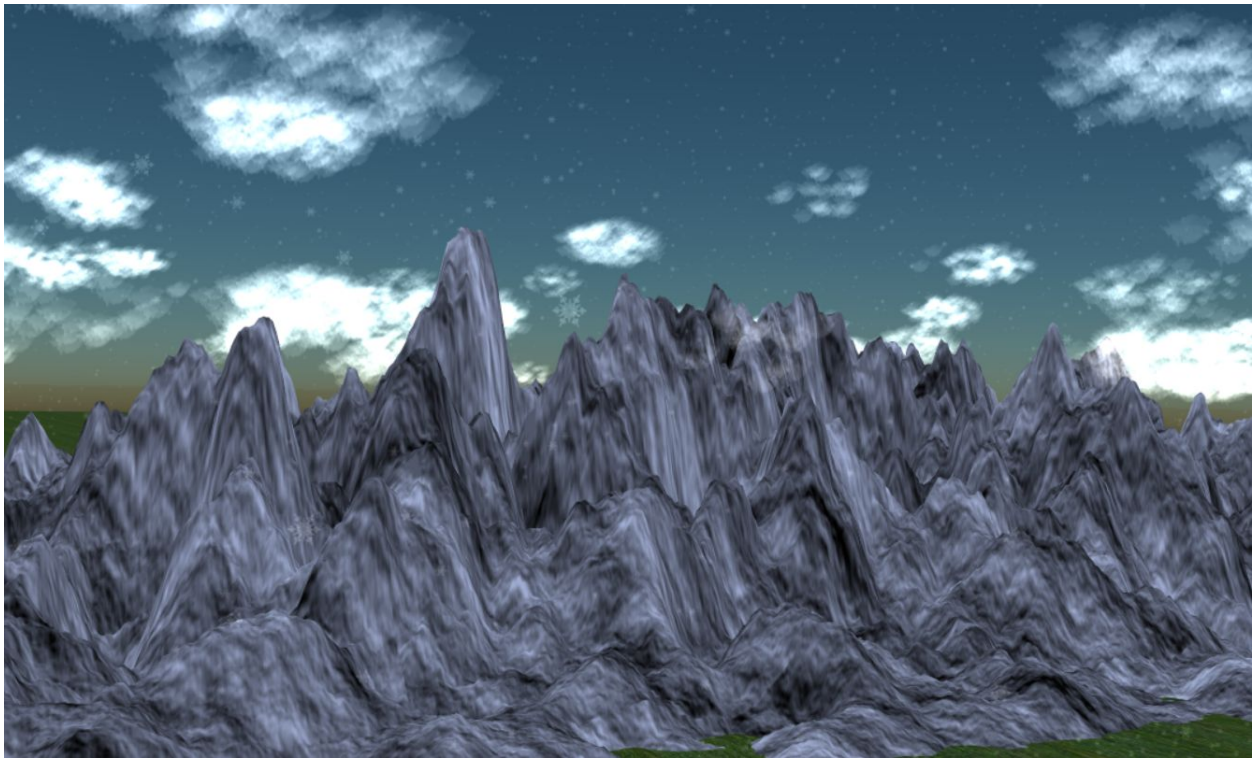
We implemented both clouds and snow for our weather simulations.

For the snow simulation, we needed to decide whether our snow would be represented by particles in a particle engine or points in a point object. We decided to use a point object since it made manipulation of the snow particles easier. We create a snow object by generating

20000 points uniformly at random in a sphere centered at the origin, and making sure that the y position of each point is above the ground plane. To get the effect of falling snow, for each render step we iterate through all of the vertices, adding a “gravity force” to each vertex. If the snowflake goes below the ground plane, we add it to a snow accumulator object and generate a new snowflake at a random location above the ground. Unfortunately, adding new vertices at every step becomes very slow as the simulation goes on, so we decided to only accumulate snow for the first render step. To make the snowflakes look good, we represent them with a image texture which additive blending, an opacity of 0.1, and without writing their depth to the z-buffer. This way, a snowflake cannot “block” other snowflakes and their colors blend seamlessly.



To generate clouds, we set a total number of cloud particles and cluster those particles into groups of 1-200, creating individual clouds. For each group, we randomly choose a point in our world and create an ellipsoid of semi-random proportions centered at that point. We randomly sample the location for each particle in a group from the group's bounding ellipsoid.



Results

The main goal of the project was to create a realistic landscape with mountains, sky, and weather, and so we measured our success by how evaluating both how good these different components of the platform looked and how accurately they model the physical world. The majority of our experimentation on our project came from trying many different parameters for our scene to achieve these metrics. For instance, we experimented with the radius of the sky, the radius of the stars, and the radius of the snow cloud in order to obtain the most aesthetically pleasing results. We found that the stars looked better when there were more of them far away, rather than fewer of them at a closer radius. Moreover, the snow simulation was better with a smaller point cloud radius so that fewer vertices needed to be manipulated to get the same snow density. Furthermore, the sky radius was best far away so that you could move around the scene more freely without “exiting” the earth.

As discussed, we did a lot of experimenting in order to make the mountains look as accurate as possible. One constraint we faced was that when the size of the mountains was too large, it took a lot of time to render the scene. This is why our mountain patch is relatively small.

Discussion

We are satisfied with the accuracy of our terrain generation. We leveraged traditional terrain generation techniques (i.e. procedural generation) to create the basic height map, but then modified that height map using specific knowledge about the structure and layout of mountainous regions. Viewing the result of our final generation algorithm for the first time was a

neat experience because we were surprised by the resemblance that our terrain bears to the Teton range of Wyoming.

The Tetons

One specific characteristic of our terrain that interested us was the presence of false summits -- smaller prominences (local maxima) part of the way up larger peaks. False summits are a real thing on real mountains, and it was cool to see them pop up in our simulation.

Disappointment Peak is a famous (and aptly-named) false summit. It was first summited in the 19th century by a group of mountaineers who thought they were climbing the Grand Teton. You can imagine their dismay when they reached the top only to find a massive void between them and the Grand!

We are also very satisfied with how our sky simulation turned out. Using a fragment shader to calculate sky color is very fast. However, we think that the colors of the sky could be slightly improved. For instance, the transition from day to night right as the sun passes the horizon is not very accurate. The part of the sky close to the horizon becomes bright red with

the sky above it immediately turning to black. In the future, it would be nice to alter the shader to have the sky have more of a gradient from light blue to dark blue to black.

Moreover, we are pleased with our star simulation. The point object for the stars renders very quickly and allows us to manipulate star positions very easily. There were a few alterations we would have liked to make to the stars if we had time. We would have liked to simulate the Milky Way by increasing the probability density for generated points near a certain cross section of the sky as well as having different colors based on their position in the sky. For instance, points near the Milky Way disk could be more colorful than those farther away. We also would have liked to add a moon simulation during the night where the moon would cast a small amount of light onto our mountains.

In terms of our weather simulation: While having the snow particles in a point cloud was perhaps the easiest way to work with them, it would have been nice to have more particle simulation effects on the snowflakes, such as wind. For this, we would have needed to make our snowflakes into actually geometries with faces so that they could be blown around. This would likely take up a lot more computational power, but would create a nice visual effect.

Conclusion

While there is still much to be done, our platform successfully bridges the gap between entertainment applications and scientific applications of terrain generation and weather modeling. Users are able to generate fairly realistic mountains, model the path of the sun through the sky, and simulate clouds and snow showers. In completing this project, we learned more about the physical laws of our world, procedural generation as it pertains to graphics, and GLSL shaders (among other things). And though we're looking forward to the semester being over and being able to go out and play in actual mountains, this semester of class equipped us well to make do in the meantime.