COS 426 Final Project
Written Report

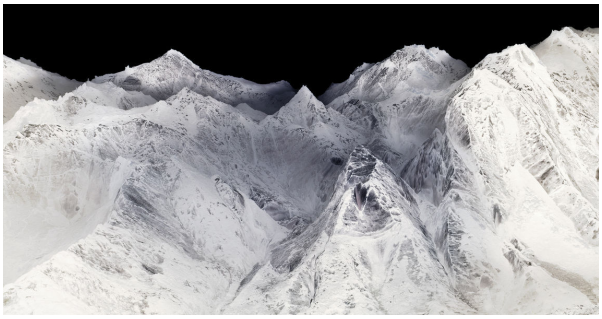
Charles An & Matthew Shih

Introduction

For our final project, we wanted to make a wallpaper or screensaver that we could use beyond the class. While creating video games, which a lot of other teams opted to do, is cool and an interesting project, we felt that we wouldn't touch the game after the class was over. However, a wallpaper or screensaver would be something that we could integrate into our daily lives and waste a lot of time staring at. We were inspired by previous course projects like Singularity and by the procedural generation examples in the slides from precept 7 because they were mesmerizing to watch and it was fascinating how natural elements like trees and mountains could be artificially generated. Our goal was to create something similar and be something that anyone could download and use.

Related previous work in procedural generation mostly involved creating still images of natural landscapes like mountains and forests or creating cool patterns that don't resemble anything. The summation of various Perlin noise functions seemed to be a consistent building block across all procedural generation projects that used fractal Brownian Motion to generate patterns. The work that we saw from others was primarily trying to find the right noise parameters and a unique function wrapping of the standardized fractal Brownian Motion function that would create a desired visual effect. By using this technique, previous projects have created very cool images ranging from mountain ranges (exhibit 1A) to random swirl patterns (exhibit 1B). While past projects involved generating still images of both recognizable terrain and unrecognizable random patterns, only the unrecognizable random patterns were animated. For example, Inigo Quiles was able to animate his project in Exhibit 1B by warping the image and was able to make the swirls change position and shape over time. However, this warping technique has not been used to animate terrain and natural landscape projects. This could be because, in order to animate recognizable objects, they would need to behave and move as they do in real life, which is a motion that is more complicated to simulate. On the other hand, any warping of an unrecognizable pattern is likely to create a cool effect since it doesn't have to obey the same "rules". Due to a lack of previous work in the area, our goal was to generate and animate recognizable terrain.

Exhibit 1: Examples of Previous Procedural Generation Projects



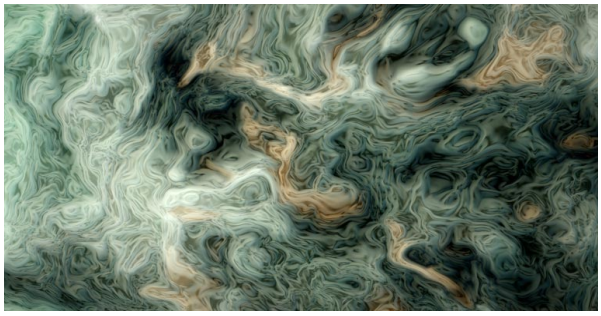


Exhibit 1A: *Blackout - Dan Holdsworth (2010)*	**Exhibit 1B:** *Inigo Quiles (2002)*

Our approach was to generate a still image using procedural generation and then change some of the noise and fractal Brownian Motion parameters by introducing the current time as a parameter to create the effect of animation. This was the approach used by Inigo Quiles to animate his unrecognizable patterns and if we want to create a screensaver that can run in the background indefinitely, then changing the image using the current time is crucial. Otherwise, we would only be able to generate the animated effect for a finite period of time and then have to loop over it, which wouldn't look realistic because real-life movements aren't looped.

In order to successfully animate recognizable terrain, we needed to find the right landscape and subject to generate. As mentioned before, recognizable terrain must move in a way that resembles how it does in real life. Due to the random-like nature of fractal Brownian Motion (because of Perlin noise) and the way we adjust parameters for animation, not every terrain will work. We need a terrain that appears to have random motion naturally so that our animation of it will look realistic. This rules out terrains like mountains and trees since mountains do not move (making the animation pointless) and trees move in a predictable back and forth motion in the wind. Also, trees are connected to themselves, so if one part of it moves, nearby parts of the tree will move with it. This is similar to the constraints in the cloth simulation assignment, however, since procedural generation randomly generates curves and shapes that just happen to look like trees, it is much harder to add constraints since there are no tree particles that we can link to each other as we did in the cloth simulation. Therefore, we need a terrain that has random-like motion and isn't connected to itself like trees are. Two such terrains are water and sand. Water and sand move in seemingly unpredictable ways and implicitly have 'particles' that have negligible constraints between them. Fortunately, water and sand often exist together in nature and thus, the recognizable terrain that we will generate and animate is a coastline.

Generating the still images for water and sand should be feasible and work well because they don't have a defined shape and can be found in a wide variety of shapes and patterns. This should make using Perlin noise and fractal Brownian Motion effective since Exhibit 1B shows that we can generate the smoothness that we associate with water and Exhibit 1A shows that we can also generate the coarser textures that we see in sand. Furthermore, when water washes up on the shore, the outline of the boundary between the sand and the water is a seemingly random curve that can be modeled well using a one dimensional fractal Brownian Motion.

Methodology

In order to setup the environment for our project, we chose to use assignment 3 as a base since, intially, we thought that using a scene and shaders would make the result more realistic. In the scene, we started off using a large plane and coloring that similar to how we colored special materials (like checkerboard) in assignment 3. However, with the plane, our zoom in and out

feature would be moving the camera relative to the plane and this is an issue since eventually we would clip through the plane and see nothing on the other side. Also, the ability to rotate the camera was undesirable since we weren't creating a three dimensional representation of a coastline. We chose to represent the coastline in two dimensions because it was simpler and would be a good starting point for future work. Therefore, we later chose to display the textures directly onto the screen instead of onto a plane object.

The first step in generating a still image of our coastline is to implement Perlin noise. The choice of noise function is not that important and we could have implemented other noise functions such as gradient or simplex noise. Perlin noise was something that we had worked with in previous assignments and so we chose it out of familiarity. The general idea behind Perlin noise is to randomly generate points and then smoothly interpolate between the points. In two dimensional coordinates, we find the grid vertices around a given point and assign each vertex a random gradient vector, then we find the distance vectors from each grid vertex and dot the distance vectors with the gradient vectors. Finally, we interpolate the results using a fade function.

Using our Perlin noise, we can build our fractal Brownian Motion function by summing over different levels of Perlin noise samples. For context, the amount of times that we iterate over Perlin noise is called the octave, the change in the noise frequency is the lacunarity, and the amplitude of the noise is the gain. By increasing the number of octaves, increasing the lacunarity, and decreasing the gain, we can generate noise functions with more detail and by summing over these different noise functions of varying detail, we create fractal Brownian Motion. Research shows that a gain of 0.5 is ideal since it creates visually appealing and accurate fractal terrains. We also experimented with different gain values and found that a gain value of 0.5 did in fact look the best, so we stuck with it.

By composing our fractal Brownian Motion function and wrapping it inside itself, we can generate different textures and patterns. This process is called Procedural Generation. By wrapping the fractal Brownian Motion function inside itself, we can customize how our result looks. We experimented a lot with the function compositions and with the function parameters and for the water, we finished with the following simplified composition: $f(p) = fbm(p + fbm(p + fbm(p)))$. The actual composition has a lot of parameters and constants, but the simplified version shown here gives a good idea of how the fractal Brownian Motion function generates textures. What we found was that the less wrapping you use (lower number of fbm() functions), the more cloudy the texture looked and the more wrapping you use (higher number of fbm() functions), the more detail and granularity the texture has. Exhibit 2 shows this effect. Therefore, by adjusting how many fractal Brownian Motion functions we use, we can make textures look like a gas, liquid, or solid. For the water, we used enough *fbm* functions so that it didn't look like a gas, but not enough to have too much detail and look solid. For the sand, because of how granular it naturally is, using a lot more *fbm* functions gave the desired effect.

Exhibit 2: Different fractal Brownian Motion Function Compositions	

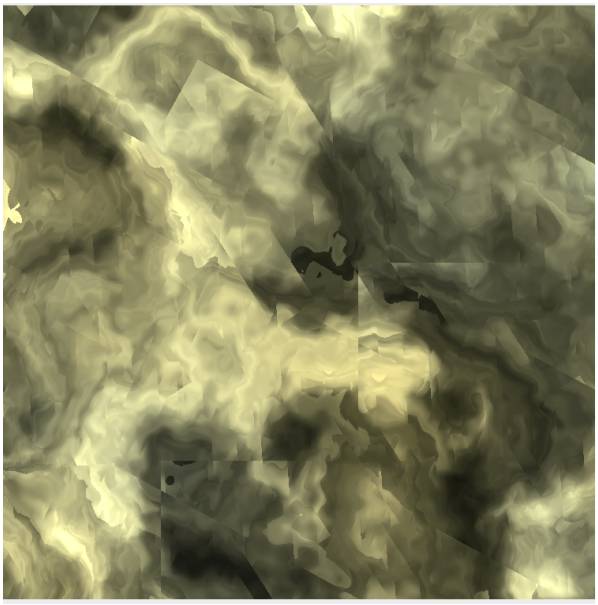
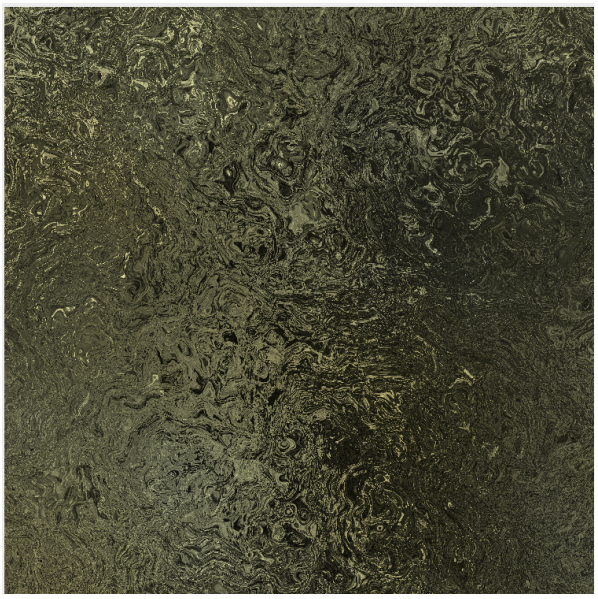

	

Exhibit 2A: Less function wrapping - *fbm(p+fbm(p))*	Exhibit 2B: More function Wrapping - *fbm(p+fbm(p+fbm(p+fbm(p+fbm(p)))))*

In order to add color, we used the intermediate *fbm* function values to augment a pre-determined color pallette. Since we are using multiple *fbm* functions, this gives the effect of layering and adds depth to the texture.

Next, we adjusted the fractional Brownian Motion function inputs to have a time component, which allowed the image to change pattern as time goes on. By adjusting how much of the time parameter we use, we can customize how the animation looks. This took a lot of tinkering, but ultimately, we were able to find the values that gave the water and sand a flowing look. The water should flow much more than the sand, so for the sand, we scaled down the time parameter so that there was much less movement.

Next, we needed to implement the transition between the water and the sand. When waves crash onto a beach and run up the sand, the boundary between the water and the sand appears to be a random curve. We saw this as an opportunity to use a one dimensional fractal Brownian Motion function as the boundary between the water and sand. We added a time parameter to the boundary function in order to give motion to the water and sand intersection, thus making it seem more realistic. Also we interpolated the sand and water textures at the border to make a msmoother transition.

We also wanted to add a degree of user interaction with our project and we chose to implement a customizable color palette and the ability to zoom in and out of the coastline. For the color pallete, we have a set of default values that we think look the most like water and sand,

but the user is free to customize what color they want the water or sand to be. In addition to changing the visuals, by adjusting the color values, the user is able to see how the fractal Brownian Motion function behaves. Because fractal Brownian Motion functions have a self-similarity property because of its fractal nature, we thought it was appropriate to allow the user to zoom in and out of the coastline in order to observe the self-similarity property. This also adds a degree of realism since coastlines are known to be fractals themselves.

Results

Our primary measure of success was how realistic our results were. In general, we are happy with how our results turned out. The water part looks like it is actually flowing and the animation behind it looks sound. The sand looks very granular at certain levels of zoom and looks accurate. Also, the sand texture changes very slowly, similar to how sand on a beach doesn't change position a lot. However, there are areas where the results fall short. Despite the flowing effect looking accruate, the underlying texture that is being warped doesn't look exactly like water and you can tell that it is not the ocean when zoomed in. The sand has a similar effect: you can tell that it is not actually sand since it has 'swirls' in it caused by the fractal Brownian Motion function composition. Also, the intersection between the water and the sand doesn't look entirely accurate either. Although the results aren't photo-realistic, you can still tell what kind of terrain we are trying to simulate. This is because the water animation and motion is very convincing.

Discussion

Overall, the approach we took does seem promising. Despite the water not being photo-realistic, we believe that with the right fractal Brownian Motion function composition, the water texture will look much better and closer to the real thing. The flowing effect on the water is very convincing and shows that our animation technique does a very good job at simulating real life random-like motions.

Our process for finding the best *fbm* function compositions was mostly guess and check. There are in theory an infinite number of patterns and textures that can be generated using various fBM function compositions and our guess and check method doesn't come close to exhausting all possibilties. A more effective way would be to mathematically work out how wrapping fBM functions behave and use that math to "solve" for an optimal composition and thus narrow the list of possible fBM function compositions to try.

More work should be done on mapping real life textures to fBM function compositions. Because of how standardized fBM functions seem to be and how they are used as building blocks in procedural generation, it seems possible to have fBM function wrapping 'templates' that can be used to create a desired texture.

Conclusion

Our goal was to create a scfreensaver or wallpaper that is nice to look at and we feel like we effectively attained this goal. Despite not being photo-realistic, the water motion (especially when zoomed in) is very mesmerizing and we found ourselves staring at it for long periods of time. Our next steps would be to continue playing around with different fBM function compositions to get a more photo-realistic visual effect and make the coastline more realistic. This also includes playing around with the sand and water intersection to look more like waves crashing onto a beach. Given procedural generation's ability to generate organic shapes, this seems possible to achieve. We also want to make the texture three dimensional which will allow us to create waves and make the scene even more realistic. We had some issues implementing user click interactions because of difficulties calling shader functions in .frag files from javascript and we would like to revisit these because it would add a cool way for users to directly interact with the environment and change the motions.

Contributions

We both worked on almost every aspect of the project with Matt doing more of the user interaction features and Charles doing more of the fBM parameter tweaking.

Works Cited

https://thebookofshaders.com/13/
https://iquilezles.org/articles/fbm/
https://iquilezles.org/articles/warp/
https://www.cs.princeton.edu/courses/archive/spring22/cos426/precepts/Precept-7.pdf
https://iquilezles.org/articles/warp/