Terrain Generation & Water Caustics

CMPM 163 Final Project

Alan Vasilkovsky
Computer Science
University of California, Santa Cruz
Santa Cruz, CA
amvasilk@ucsc.edu

Bradley Gallardo
Computer Science: Game Design
University of California, Santa Cruz
Santa Cruz, CA
bradleygallardo@ucsc.edu

Introduction

In this project, we created an underwater scene using an advanced noise function to generate terrain and water effect which has caustics. Alan Vasilkovsky worked on the terrain generation and fog, and Bradley Gallardo worked on the caustic effect for the water.

Abstract

Our vision was to create a beautiful looking underwater scene, inspired by the gorgeous coral reefs. To do this, our goals were to make a detailed, procedurally generated terrain and add underwater caustics. Additionally, our other goal was to make underwater Crepuscular Rays and add some volume rendered fog although these parts didn't come to fruition with our given time limit.

1 Terrain Generation

While it is common for game creators and artists to create their own, highly-detailed terrains and assets by hand, a more interesting route (in our opinion) is to have the terrain be generated. This adds an element of randomness and excitement to a project. Since randomly generated terrains are commonly created with a noise function, the creator can change this function to mold the terrain into whatever shape they desire given with a little bit of knowledge in mathematics.

1.1 Shader/Implementation

To create the terrain generator/displacement shader I started with a standard Unity surface shader. First, I dropped in a basic 3D plane game object. To add tessellation to it, all you have to do is add a pragma directive to your shader which specifies a tessellation function. Now that tessellation is added, you can create a slider which controls the amount of detail in the mesh, which was set to a fairly high number to add lots of detail to

the mesh. This allows the mesh to have many vertices and thus look smooth when stretched and morphed.

The next step is to add the noise function to your mesh to give it life and make it look like real rolling hills. For my noise function, I used the simplex noise from the Noise Shader Library for Unity, created by Keijiro on GitHub. Although these built in noise functions make it easy to create interesting terrains with a simple function call, the interesting part comes when you add some variables to the shader to control how the terrain is generated. I added a few variables: Noise Speed, Noise Scale and Noise Frequency, to be able to move and shape the terrain as I please. This can be achieved by adding and multiplying these variables to parts of your noise function to create a cool effect. I also created a simple C# script which allows you to control these variables with your keyboard and get some awesome looking terrains.

The final step is to recalculate the normals of your mesh so that you achieve realistic lighting. This is a two step process. First, you have to calculate the tangent and the bitangent of every vertex in your mesh by offsetting the original vertex. Then, you have to take the cross product of those two vectors and normalize it to get your final normal.

2 Volumetric Fog

Fog is a common staple in many forms of digital media from indie games and AAA titles to movies and TV shows. There are several ways you can create fog, but the two most common methods that we researched were using image effects on the entire scene by applying it with the camera, or using a volume rendering technique to make the fog appear more three-dimensional

2.1 Implementation

For our volumetric fog, I decided to go with a volume rendering technique using a particle system. To start, I added a standard particle system to the Unity scene, and I went to work on changing up the properties to make it look like real fog. The first thing was to get rid of the standard circular texture and replace it with a more realistic looking cloud texture. I found a sprite sheet of cloud textures from Google which I used to sample from.

Next, I changed the default cone shaped particle emitter to a box emitter to make the particle system look like a blocky patch of cloud, and I increased the scale a bit to fit more of the scene. After that, I changed the starting size and starting angle of the particles themselves to be a randomly chosen number between two set values to give the fog a more realistic look. If you don’t change these values, it just looks like a perfectly square patch of fog.

Lastly, to make it look as though the fog was disappearing and reappearing, I altered the color slider so that there were three preset values: White with 0% alpha at the beginning and end of the particle’s lifetime, and White with 100% alpha in the middle of their lifetime. Unity blends

these values together to give a smooth color curve, which really makes it look like the fog draws in slowly and then dissipates, just like real fog.

Unfortunately, given the time constraints, I was not able to make the fog appear in the final build of the game, as the new Unity particle system is a bit finicky.

3 Water

Water effects are commonly used in most games. Each has their own way of producing their desired water. I was trying to simple water effect, that wasn't too overbearing so that the caustic effects can shine.

3.1 Shader

Functionally, the water is just a linear interpolation between a brighter and darker color. The lerp is dependent on the current pixel and pixel directly behind. If it latter pixel is deeper than it will be colored darker.

3.2 Implementation

To implement this, create a plane, add the shader, and adjust the shallow and deep colors.

4 Caustics

Caustics are grouping of light rays that are reflected or refracted by a curved surface or object. The caustic itself are the boundaries of the curved surface created by light. In practice people use either forward or backward ray tracing to accurately create caustics. However, I take a more simpler and less computationally taxing route.

4.1 Shader

My shader takes in an animated caustics texture map and plays it over the surface of the water. Using a caustics map generator I created this asset and downloaded an image to act as distortion. The actual shader code, just reads the two textures and lerps between them. Additionally, for the rim, If the terrain height is close to the water, the edge will be given a black gradient. This is done using a linear saturate interpolation between the low point and the edge.

4.2 Implementation

To implement this, create an empty entity, add the projector component, adjust to scene, apply shader, and tweak till you are satisfied.

5 Technical Challenges

Our biggest challenge was trying to create shader programs that are compatible together. Both us had busy schedules, especially coming into finals week. We had trouble finding time to work together. So a lot of our code was made independently.

Bradley Gallardo: A challenge that I faced was that it was difficult to create a realistic caustics shader that used ray tracing to accurately reflect light. After some time, I ended switching to a different route for producing caustics. Using a projector I played a textured animation of caustics over the water surface.

Alan Vasilkovsky: A technical challenge that I faced was getting the proper normal calculations on my randomly generated mesh. Unfortunately, it wasn't as simple as running a build in Unity function to calculate the normals. I had to do some complex mathematics which first involved calculating the tangent and the bitangent for every triangle on the mesh, and then using the result of those calculations to calculate the final normal.