Terrain Generation I

## Overview

The main environment we wish to generate is the depth of an ocean including the islands. Underwater caves will be included as well as underwater rock formations. Some formations will be huge enough to reach above sea level. Those can be perceived as big rocks or even islands that might also include vegetation. The distinctive feature will be the blend of underwater and above-water terrain and the different textures in each type of landscape. For instance, caves will have different geometry deformations compared to the hills and mountains above sea level. Furthermore, we will consider adding different types of vegetation based on the type of terrain. In the ocean we can randomly place algae or seaweed plants, in the mountains bushes and small trees and in the beaches grass and foliage.

A picture containing swimming

Description automatically generatedA picture containing tree, blue, reef, colorful

Description automatically generated

Figure 1: (Left) Underwater Cave, (Right) Rocky Islands

## Our Method

Our system consists of two parts, an evaluation of a density function in 3D space and the geometry generation based on this function.

The density function will be evaluated using compute shaders and sampling of different octaves of noise (e.g., Perlin noise). In essence, we will be using Fractal Brownian Motion (FBM) with the lowest frequency covering the main features of the terrains (mountain, cliffs, caves). Our system’s inputs are the different noise textures, the terrain’s size and the various methods through which we will calculate the density function. The output of the system will be a textured 3D geometry (an array of 2D textures).

We will use the Marching Cubes (Bourke, 1994) algorithm to generate the polygons by sampling the 3D texture. The algorithm takes as input the density values of the corners of a voxel. Each combination defines a specific set of polygons that will be generated from the initial voxel. We will render blocks of voxels using GPU instancing, with each of them corresponding to one voxel in the 3D geometry. The marching cubes algorithm will be implemented in a compute shader. This shader is capable of streaming vertices and producing geometry in the graphics pipeline.

We will also include a simple shading and coloring of the generated terrain based on the specific type. For instance, the mountains above sea level can contain tree models, while the beach can contain grass textures and foliage. For accurate lighting the normals will be calculated by the spatial difference of neighbors probably in the compute shader stage.

Shape, polygon

Description automatically generated

Figure 2: Examples of sets of generated polygons using the marching cubes

## Suitability

Voxel-based methods are more suited for creating caves and overhangs. Since our main concept is to create complex underwater terrain it would not be possible with basic 2D terrain generation algorithms such as vertex displacement with heightmap textures.

By changing the types of noise, the number of octaves and the frequencies we generalize our method to create different types of terrain. Since the method is voxel-based the possibilities are limitless. Also, different manipulation of the noise such as warping or other various creative functions can produce interesting results.

## Parameters

* Type of Noise – We expect to use Perlin noise to evaluate the density function but we will experiment with other types of noises as well
* Octaves
* Functions that manipulate the noise (floor, smoothstep interpolation)
* Terrain size
* Voxel resolution
* 3D models – Trees, grass, bushes, corals, etc.

The expected outputs are the 3D density texture, the generated polygons using marching cubes and the final terrain. We expect the final terrain to be closer to an underwater world than a full island, but will probably contain an archipelago consisting of small landmasses

We will build our application using Unity 3D 2020.3.9f1 and we will fully utilize the GPU by programming compute, vertex, geometry and pixel shaders (nVidia Developer: GPU Gems 3, 2007). We will consider using readily available assets under the CC or BSD License for the various 3D objects that we want to embellish the terrain with.

# References

Bourke, P. (1994, May). *http://paulbourke.net/*. Retrieved from http://paulbourke.net/: http://paulbourke.net/geometry/polygonise/

*nVidia Developer: GPU Gems 3*. (2007). Retrieved from nVidia Developer: https://developer.nvidia.com/gpugems/gpugems3/part-i-geometry/chapter-1-generating-complex-procedural-terrains-using-gpu