# AI For Game Technology – Assignment 7

## Terrain Generation

## The team

Dimitris Christodoulou - 5141761

George Psomathianos - #TODO ADD SID

## 1.1 Abstract

In this assignment, we explore the Marching Cubes algorithm and its usefulness when creating voxel-based terrain via a density function. We use this algorithm to generate interesting underwater terrain that includes sporadic islands. Such islands will be created by huge underwater rock formation and might span in size, from big rocks to huge islands that will include vegetation.

## 1.2 The System

The application is built using Unity 3D 2020.3.9f1. We utilize the GPU by generating the terrain with geometry, compute and fragment shaders and Unity’s standard surface shaders. Since the program runs almost exclusively on the GPU, each terrain generation takes less than a minute, depending on the scale of the terrain and the amount of vegetation that is instantiated.

### 1.2.1 High level overview

Our terrain generation system consists, at a high level, of two parts; The evaluation of a density function in the 3D space and the cube marching based on this density function. The first part will determine how interesting and varied the terrain will be and the second part will actually generate the terrain. For the density function part we use noise to initialize the terrain, while we afterwards iterate over this procedure via Fractal Brownian Motion to finetune the terrain details. Once the terrain is generated, we use Unity’s surface shaders to color the terrain via a height-based layering approach and finally, we generate vegetation based on random sampling of vertices in a specific layer (the grass layer).

### 1.2.2 The system in detail

We evaluate the density function through sampling of Perlin noise. We use Compute Shaders for this part. For the terrain details we use Fractal Brownian Motion, through which we loop over a number of octaves and take low frequencies to cover the terrain’s main features such as underwater caves and mountains and high frequencies for the details of the terrain.

After we evaluate the density function we use the Marching Cubes algorithm to generate the polygons through this density function by sampling a 3D texture. This algorithm applies the density function for each vertex of each voxel of the 3D texture and associates each vertex with a specific density value. With each combination of density values in a voxel the Marching Cubes algorithm cuts a set of polygons from that voxel, with a total of 15 possible polygon configurations for each voxel. We use GPU instancing for this rendering procedure and utilize geometry shaders for the geometry generation, in order to have faster streaming of the generated vertices and polygons.

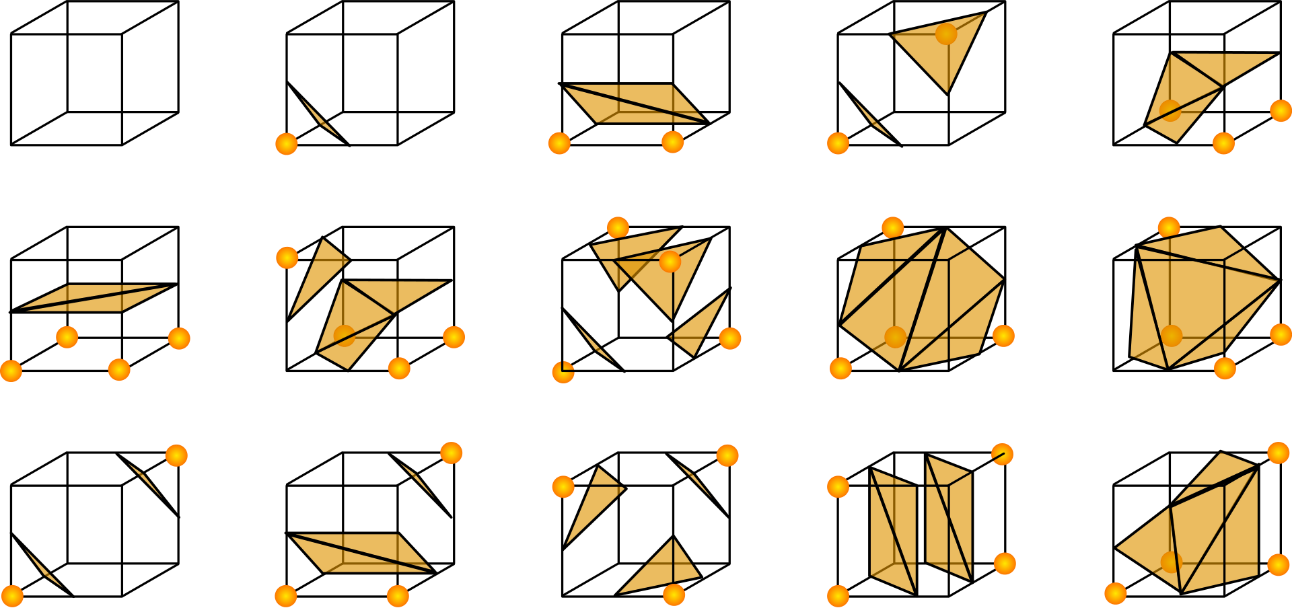


Figure : The 15 possible polygon configurations for the Marching Cubes algorithms

We expose different parameters for the terrain generation in order to introduce a large amount of variation in our terrain. Since our scope is to have mainly a large and detailed underwater habitat, we have made it so that the underwater caves scale and smoothness is adjustable, as well as the terracing between each cave layer and the total area of the cave. Since a nice aftereffect of our terrain is that we can also generate large rock formations or islands or coastlines, we also expose parameters that control how large these formations are and if they contain mountain formations or if they are flat. Therefore, we can control how mountainous, steep and tall the area above water is and also how much smooth it is. We can also control if we want multiple mountains above ground or if we want the overall height spread of the terrain to be concentrated in one landmass (i.e., one large mountain).

In order to have more interesting noise as our terrain initialization we used domain warping (Quilez, n.d.). Domain warping or noise distortion is a technique through which we can manipulate basic Perlin noise by adding other layers of Perlin noise over it and stretching our initial layer’s points based on the new layers. For this purpose, we also exposed a few parameters to manipulate the warping frequency and amplitude.

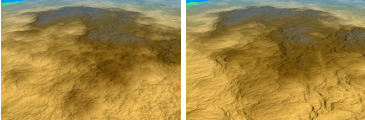


Figure : Left - Terrain without domain warping. We can almost notice how noise was applied to create it. Right - The same terrain, now with domain warping, looks more natural.

After we generate our terrain, we apply some basic shading to it. This shading is based on a simple height layering of our terrain and is done via Unity’s surface shader. We use 4 layers with 4 different shades;

* A cave/underwater layer with a gray shade
* A beach/ground layer with a desaturated yellow shade
* A grass layer with a green shade
* A mountain layer with a brown shade

After we shade the terrain mesh, we generate some simple low-poly vegetation. We generate this vegetation on the grass layer for simplicity. We do this by sampling random vertices of the mesh in the grass layer and instantiating random vegetation prefabs from a list with a random rotation on the XY plane at the sampled vertices positions. For this part we expose as parameters the starting and ending height of the vegetation (from which and until which height vegetation will start spawning) and the probability that a vertex will contain vegetation. The vegetation included in the project was obtained from the Unity asset store and is distributed through the Standard Unity Asset Store EULA. We also included a simple Post Processing volume to simulate the underwater effect.

## 1.3 Using the terrain generator application

When launching the application a terrain with some preset values is generated. The user can move around with a flycam by using the WASD buttons (W for forward, A for left strafe, D for right strafe and S for backwards). By holding down the Right Mouse Button and moving the mouse the user can rotate the camera freely. By clicking on the “Terrain Settings” button, a UI pops up with a scroll view that contains slider bars that control the terrain parameters. The user can navigate to each section of parameters by clicking the arrow buttons on each section. #TODO INCLUDE SCREENSHOTS OF UI

## 1.4 Results

From our observations we achieved what we initially had in mind without lots of deviations based on our first design, when we were thinking of underwater terrain. The terrain contains some complex underwater hallways, while in other cases it also includes tall mountains or hills or small valleys between two mountains etc. The voxel geometry is also ideal when creating this kind of terrain, as we can create interconnected layers of corridors and passages. #TODO INCLUDE SCREENSHOTS OF RESULTS PLUS ALSO THE SETTINGS USED

# References

Bruno Torres do Nascimento, F. P. (2018). GPU-Based Real-Time Procedural Distribution of Vegetation on Large-Scale Virtual Terrains. *SBC – Proceedings of SBGames 2018.* Retrieved from https://www.sbgames.org/sbgames2018/files/papers/ComputacaoFull/188348.pdf

*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

Quilez, I. (n.d.). *domain warping - 2002*. Retrieved from https://www.iquilezles.org/www/articles/warp/warp.htm