Biased Terrain Generation for Floating Islands

Oier Perez and Felix Neila

1 Introduction

Many games employ terrain generation as a form of content generation, and use it to procedurally create entire worlds. However, sometimes the objective is not to create large, generic terrain but rather to generate specific shapes or terrain features, such as islands, mountains or valleys.

Under these circumstances, the objective is to modify or bias terrain generation methods into creating the desired features. This paper will discuss this idea applied specifically to the generation of floating islands, a fantasy terrain feature that many fictional settings employ and that is usually not covered by standard terrain generation methods.

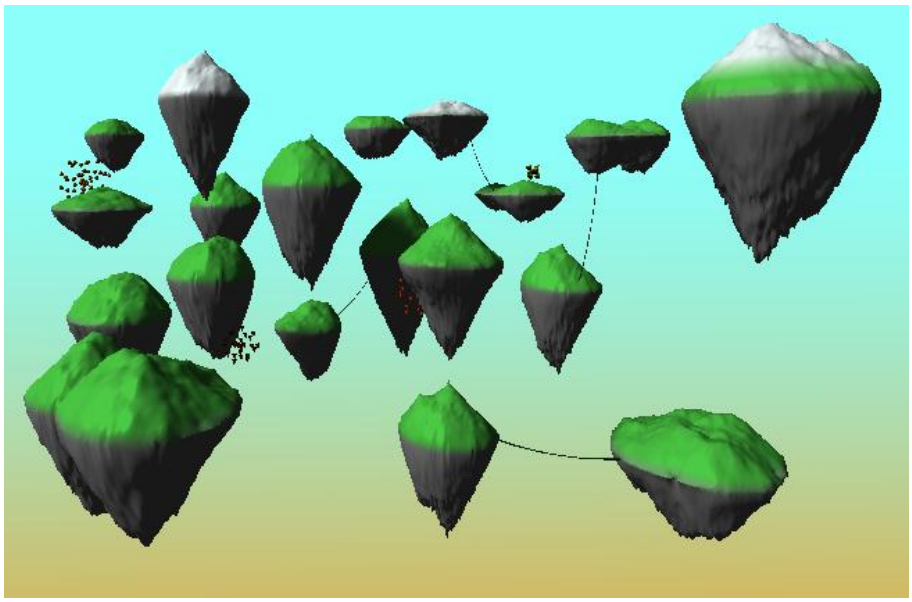

Figure 1 Floating Island videogame environment

2 Overview

In order to generate floating islands, it is first necessary to define how they should look. The desired floating islands will be like normal islands but with a rocky underside. Several of them will be generated and placed in an area, to then create a complete terrain composed of a group of floating rocks.

Therefore, the island generation process can be subdivided into several steps: first calculate the points at which to place the islands, then for each one generate island-like terrain and the rocky bottom half.

3 Island Placement

For the islands to look naturally placed, it is important that they not be placed using clear patterns. However, it is also necessary for them to not overlap, so they cannot be simply randomly placed. Thus, the method used for placement shall be Poisson Disc Sampling[i], which creates a blue noise sample pattern of points. Since the objects being placed are floating, the sampling used will have to be done in three dimensions.

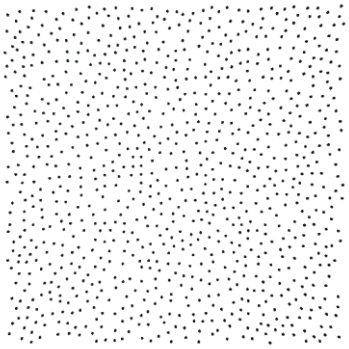

Figure 2 2D Poisson Disc Sampling

This algorithm finds points with a minimum distance r between them in the domain $\mathbb{R}^n$ given a limit of samples to choose before giving up of k.

Poisson disc sampling works by placing a random point in the space and adding it to an active list. Then, while the active list is not empty, a random origin point is chosen from it, and the addition of a new point at a distance between r and 2r is attempted. If the new point is at a distance of less than r from another point, then it is not added. If it is not, then the point is added, and it is also put in the active list. k additions are attempted for a point, until one succeeds. If all fail, then the origin point is removed from the active list.

In order to make the check against other points more efficient, the points can be added to a uniform grid of cell size $r / \sqrt{n}$, which can be expressed as an array of integers, since only one point will be in each cell. Then, to check whether a point has a collision with other points, it is only necessary to check the surrounding cells.

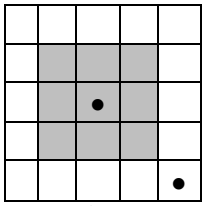

Figure 3 Uniform Grid Usage

When applied to 3D, and using the maximum size of the islands as r, this produces a set of points that can be used as island positions such that the islands will not overlap but will still look randomly placed.

4 Terrain Generation

4.1 Noise generation

First, in order to generate the base terrain, a height map-based terrain generation algorithm will be used. Theoretically, any method of height map generation, but this paper will focus on midpoint displacement, and more specifically the Diamond-Square algorithm[ii], due to ease of use and simplicity of implementation.

The Diamond-Square algorithm is a method to recursively fill a grid of $2^n + 1$ values so that it smoothly transitions between its four edges. It consists of two steps: the square step and the diamond step.

First, the central point of the grid is chosen, and fulfilled with the average of the four corners (square step). Then, the centres of the four sides are filled with the averages of the values to their cardinal directions (diamond step). Then, this is recursively repeated for the four sub-grids generated.

```
void DiamondSquare(float array[ARRAY_SIZE],
                   unsigned size, int iteration)
{
    int half = size / 2;
    if(half < 1)
        return;

    for(int i = half; i < ARRAY_SIZE; i += size)
        for(int j = half; j < ARRAY_SIZE; j += size)
           SquareStep(array, size, i, j, iteration);

    int col = 0;
    for(int i = 0; i < ARRAY_SIZE; i += half)
    {
        col++;
        if(IsOdd(col))
            for(int j = half; j < ARRAY_SIZE; j += size)
                DiamondStep(array, size, i, j, iteration);
        else
            for(int j = 0; j < ARRAY_SIZE; j += size)
                DiamondStep(array, size, i, j, iteration);
    }

    DiamondSquare(array, size / 2, iteration);
}
```

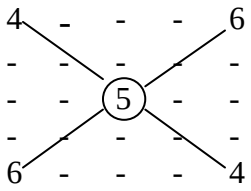

Figure 4 Square Step

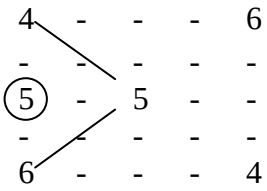

Figure 5 Diamond Step

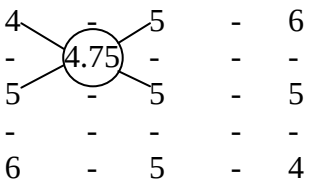

Figure 6 Square Step Second Iteration

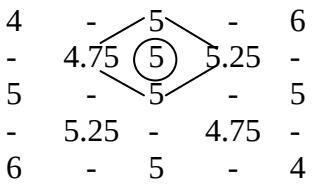

Figure 7 Diamond Step Second Iteration

In order for this to generate noise, a random value is added at each step. This value can be halved at each iteration, thus the variation becomes smaller and the final result looks smooth. This is known as smoothed midpoint displacement[iii].

However, this method generates generic terrain. In order to create floating islands, the randomness of the additions must be modified to create the desired shapes.

4.2 Shape equations

In order to give the generated terrain an island-like shape, the height of the sides of the island must tend to 0, and the island must be smaller than the height map so that it is not cut off by its borders.

So that this occurs, the random height added at each step of the smoothed midpoint displacement is multiplied by a factor that depends on the distance to the centre. In this paper, the islands desired are approximately circular, and so a simple distance equation will be used, but different shapes can be obtained by switching the multiplication factor.

Thus, the factor used shall be:

$$factor = 1 - \frac{distance^2}{radius^2}$$

where distance is the distance to the centre of the island, and radius is the distance from a side of the height map to the centre. The radius can be reduced slightly to avoid artefacts at the edges.

To avoid issues to the sides, any values farther away than radius from the centre will be set to 0, and the initial values of the sides of the height map will also start from 0. Moreover, to ensure diverse islands, the first iteration of the square step of the Diamond-Square algorithm will be skipped, and the centre of the height map will be set to a random value between 0 and 1, so that the sides starting at 0 does not affect the overall height of the island.

4.3 Bottom half modifications

To create the underside of the island, it is first important to realise that the coast of the island must connect to the generated lower half. Since the coast is randomly generated by the previous step, this half must, by necessity, use the height map as a starting point to be aware of the coastline.

The bottom half will be created by using a modified version of the height map used for the upper half. Therefore, the initial bottom half shall be a mirrored version of the top half.

The intended effect of this half is for it to look rocky and full of stalactites. If another effect is desired, different modification could be applied, and in fact the terrain could be regenerated by first flattening the height map above the coastline and recalculating those values. However, for the effect this paper desires, this is not necessary and the full range of values of the height map will be used as a starting point.

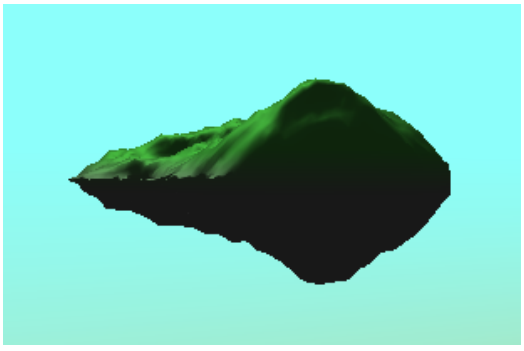

Figure 8 Mirrored Terrain

So as to create the stalactites, noise will be added or removed to any terrain that is above the coastline. In order to avoid the coastline, any values that are under the desired coastline threshold will be ignored, and any values neighbouring a value under the threshold will also be skipped since they might affect the borders if modified.

For the inner values, the noise added will once again depend on distance to the centre, and will also depend on the existing value in the height map in order to accentuate peaks and valleys. Thus, the noise factor added will be:

$$factor = \frac{value}{2} + \left(1 - \frac{distance}{radius}\right) \cdot postive\ random + negative\ random$$

Here, value is the existing value in the height map, and distance and radius are once again distance to the centre and radius of the island respectively. The positive random will be a random value in a given input range (for example, between 1.9 and 2.2) that will be added to create peaks. The negative random will be the factor that creates valleys, and will be between the negative value of the height map divided by an input factor (for example, 8) and 0.

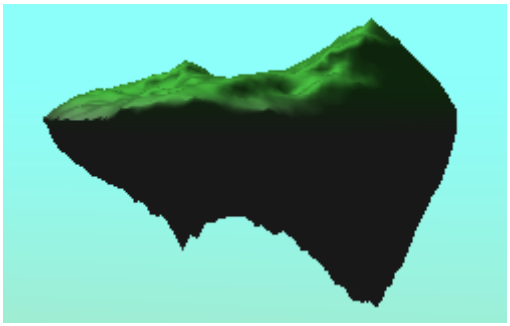

Figure 9 Island with stalactites

Adding this factor to the height map values above the threshold will then create peaks that, when inverted, will result in the desired stalactites.

4.4 Model generation

In order to create the model from the given height maps, a few details must be taken into consideration.

For the actual triangle creation, it is sufficient to employ standard height map triangulation methods (such as using every four values to create a quad). However, in order for the island to be limited to the coastline and for the two halves to fit evenly, some modifications must be applied.

To remove the unnecessary values from the model, its triangles should be clipped against a plane at the desired coastline threshold height (for example, 0.05), and offsetted by this value such that the coastline occurs at height 0. This will make the coastlines of both halves be at the same height and touching each other.

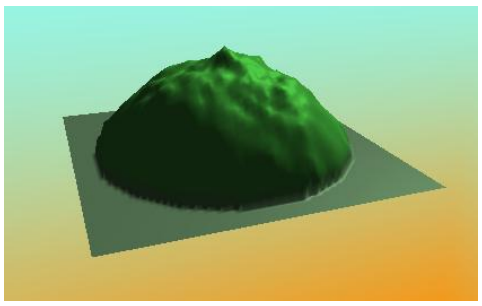

Figure 10 Terrain before clipping

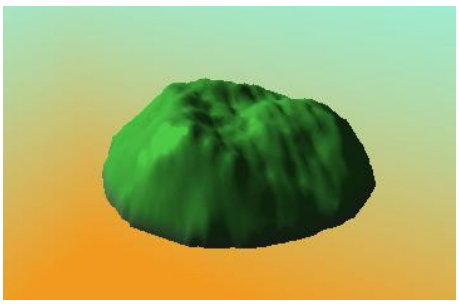

Figure 11 Terrain after clipping

However, this can still create a jagged border, and so to avoid this effect, the border values may be smoothed out. To do this, points that are close to the threshold, under a given epsilon, may be made closer to the centre of the island by a slight amount, which can be done by multiplying them by a factor close to 1 if the centre of the island is approximately at (0,0), which is true for this case. This will then make the sides of the island be slightly more rounded, thus creating a more natural transition between the two halves.

5 Conclusion

In summary, floating island generation can be executed by biasing standard terrain generation algorithms into creating island-like shapes, and then mirroring and modifying the created terrain in order to form the bottom half.

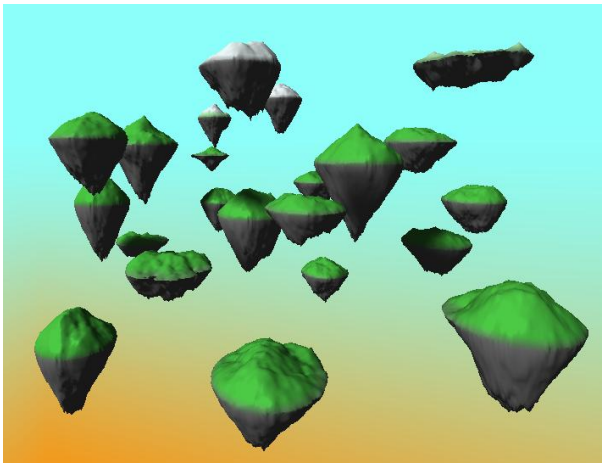

Figure 12 Final Island Generation Result

These floating islands can also be made more complex, perhaps by combining several of them into a single island or employing more advanced forms of terrain generation as a first step before biasing. They could also be made more interesting by populating them with other props such as rocks or trees, or adding further elements such as waterfalls, lakes or caves. Furthermore, other aspects of terrain generation could also be simulated and applied to the islands, such as erosion, climates and biomes, etc.

These ideas can also be applied to other forms of terrain creation, such as creating valleys, normal islands, cliffs, craters, etc. This can be done by simply changing the biasing and smoothing equations used so that they depend on their relevant parameters.

6 References

[i] Robert Bridson: Fast Poisson Disk Sampling in Arbitrary Dimensions, University of British Columbia, https://www.cct.lsu.edu/~fharhad/ganbatte/siggraph2007/CD2/content/sketches/0250.pdf

[ii] Nick O'Brien: Diamond-Square Algorithm Explanation and C++ Implementation, https://medium.com/@nickobrien/diamond-square-algorithm-explanation-and-c-implementation-5efa891e486f

[iii] Jacob Olsen: Realtime Procedural Terrain Generation, University of Southern Denmark, https://web.mit.edu/cesium/Public/terrain.pdf