Using Voronoi Diagrams to Create Realistic 3D Environments

Farina Davide, 207492
davide.farina-1@studenti.unitn.it

Abstract

For my project I decided to create a tool which lets you create realistic 3D environments by using Voronoi diagrams as a starting point. In particular, the user can set the size of the map to be created (defined in pixels) and the number of points to be used to create two voronoi diagrams, which will then be used to create the 3D map. This tool is made in Unity3D, but the resulting map is exported in PNG format, both as a height map and as a color texture, so that it can be used with other softwares as well.

Implementation

Note: in this report I will insert images of one world created by the tool. This is obviously just an example, but a representative one. Some generated environments may have different characteristics, but this example includes most features the terrains generated by this tool can have. This map has size 250x250px, 250 detail points, 10 general height points, noise of 0.2 and max height of 40.0.

As previously stated, I implemented this tool in Unity3D. The code can be found in the GitHub repository at the following link: https://github.com/DavideFarina96/3DWorldGenerator
To generate the topology of the world, I use a height map. The values of this height map range from 0.0 to 1.0, where 0 means low altitude and 1 means high altitude. In the output PNG files, these values are represented as black for 0, and white for 1, with all of the gray shades in between them.

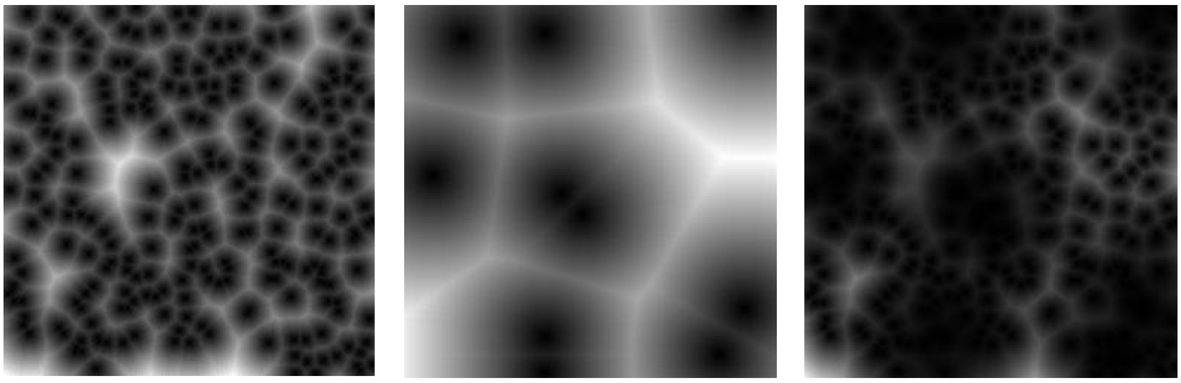
To create a Voronoi diagram, the tool randomly generates n points, where n is set by the user, and calculates the height map as a distance from the closest point. For each pixel, the distance to the closest point is calculated, and is used to determine the height of that point. The values in the range `[0, maxDistanceFromClosestPoint]` are then normalized to lay in the interval `[0,1]`. If we take a look at the second image found below, for example, we can clearly see where the points used to generate the Voronoi diagram are (black areas), and where the lines separating the various polygons of the Voronoi diagram are (white ridges). The same thing can be seen in the first image, but as the number of points used to generate the diagram was higher (`n=250` vs `n=10` for the other image), this distinction is harder to see. For each diagram, a little bit of noise is added to the value of each point, in order to make the final result appear less smooth and more randomized, more similar to real life. This value can be set by the user by changing parameter “noise”.

The final height map is generated by using two different Voronoi diagrams:

- The first diagram is used to define fine detail. This includes most mountain ridges, subtle changes in terrain shape and so on.
- The second diagram is used to define large areas in the map. This is used to characterize the terrain, and generally indicates whether an area is going to be mostly plains or mostly mountains.

To generate the fine detail diagram, the user should set the number of points n to be bigger than that of the macro-areas diagram.

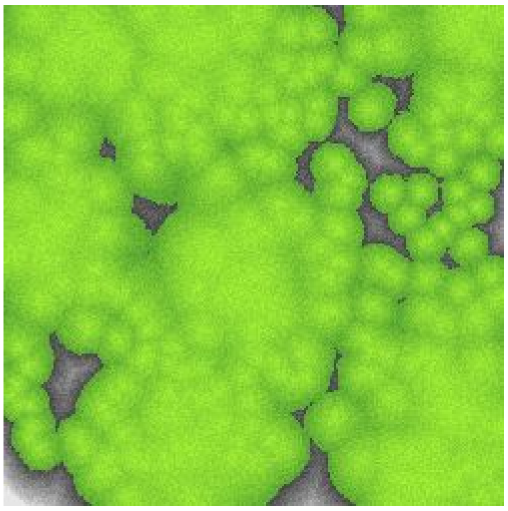
Once computed, these two diagrams are multiplied together, pixel by pixel. If the height of a point in diagram 1 was 0.2 and in the second one was 0.8, the final height of that point will be `0.2 * 0.8 = 0.16` in the final map. An example of the three maps can be found below.



Left: diagram indicating fine details of the map (ridges, ondulations)
Middle: diagram defining height macro-areas of the map (plains vs mountains)
Right:final diagram, created by multiplication of the first two

At this point, the generation of the terrain is complete. The 3D mesh seen in Unity3D is created by creating a plain with `pixelSize * pixelSize` vertices, one for each pixel, and setting its height based on the value present in the height map. The actual height of the point is calculated as `height[x,y] * maxHeight` , where `maxHeight` is set by the user.

A fourth image (found below) is also generated. This includes the colors of the environment, and are not necessary to generate the world. Still, this image is applied as a texture to the generated mesh, and is therefore integral to the generated environment itself. Its size is the same as the other ones, and can therefore also be applied in other softwares. This image is also exported at runtime, just like the other three.



Texture of the environment. Green areas are valleys, while grey areas are mountains

To generate this image, I assign a color to each pixel based on its height. If the height of the point is between [0, 0.25], the point is considered as being part of a valley, and its color will be some shade of green. The exact shade depends on the exact height, as well as a random value used to make the world appear more randomized and realistic. The code for generating the color of a point in a valley is the following:

```
float correctedHeight = height * 100.0f;
float factor = correctedHeight / 50.0f + Random.Range(-0.1f, 0.1f);
color.r = 0.6f - 0.5f * factor;
color.g = 0.9f - 0.5f * factor;
color.b = 0.2f;
```

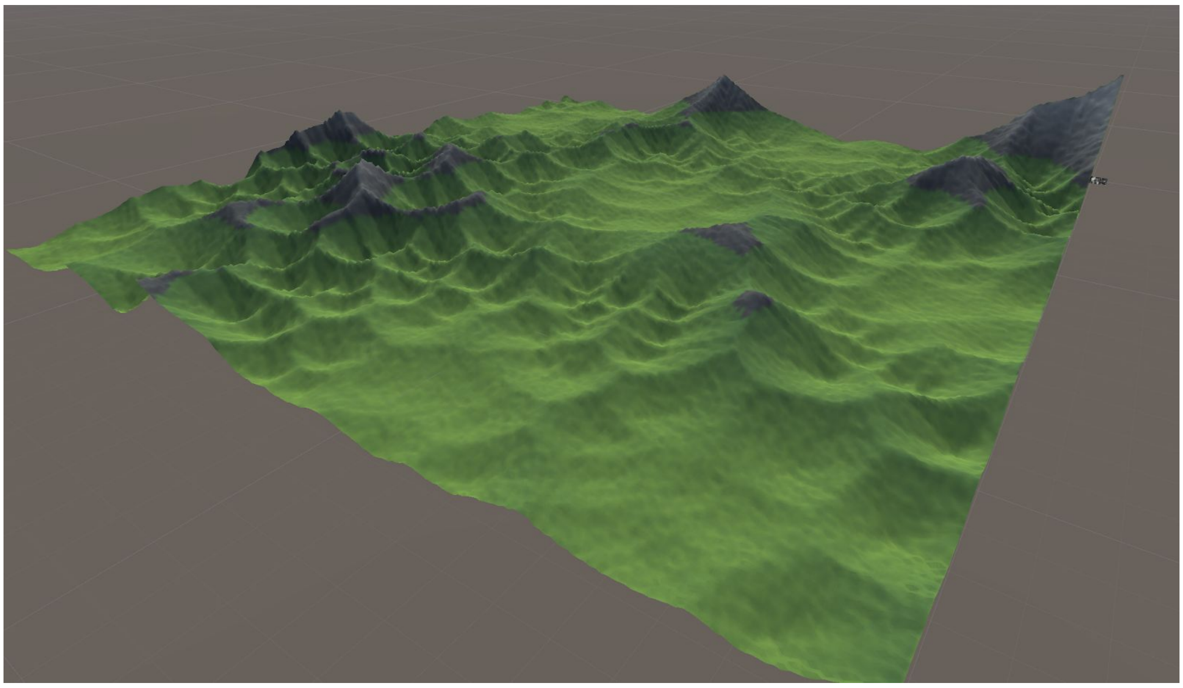
in the second line we can see the randomized value being added to the height of the point, while on the last three lines we can see how the three components (Red, Green and Blue) are calculated. As this point is part of a valley, it will be some shade of green.
The areas a point can be assigned to are:

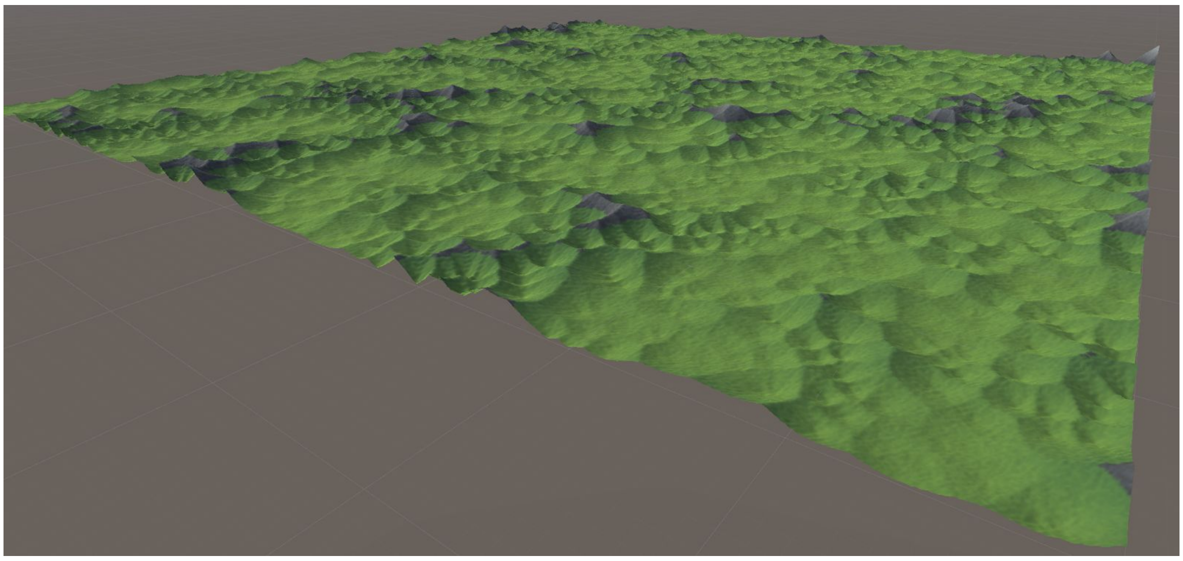
- Valley if the height is in `[0, 0.25)`
- Mountain if the height is in `[0.25, 0.70)`
- Snow if the height is in `[0.70, 1.00]`

Commented you can also find the code for generating a fourth area: beach. This was removed, as well as the sea plane, as the results were not realistic enough for my liking. The result, infact, included several lakes instead of a single sea or lake, and looked very unrealistic. This is because water (rivers, lakes and so on) carve the terrain, while the system I created simulates the increase of terren height from below. To include rivers, another approach should be developed, which carves the terrein after it is generated.

Conclusions

The image of the resulting 3D environment can be found below. To generate this terrain, the software took at most a few seconds on a 7 years old CPU (intel core i7-2600K). An example of a bigger terrain can be found below. This took about two minutes and had the following parameters: 1000x1000px, 3000 detail points, 150 general height points, noise of 0.2 and max height of 40.0.





Above: Final environment generated by the software, with the color texture applied to it
Below: Example of a bigger environment that can be generated