Procedurally Generating a Low Polygon Terrain in Unity

Mats Andersson
matsand6@kth.se

December 2020

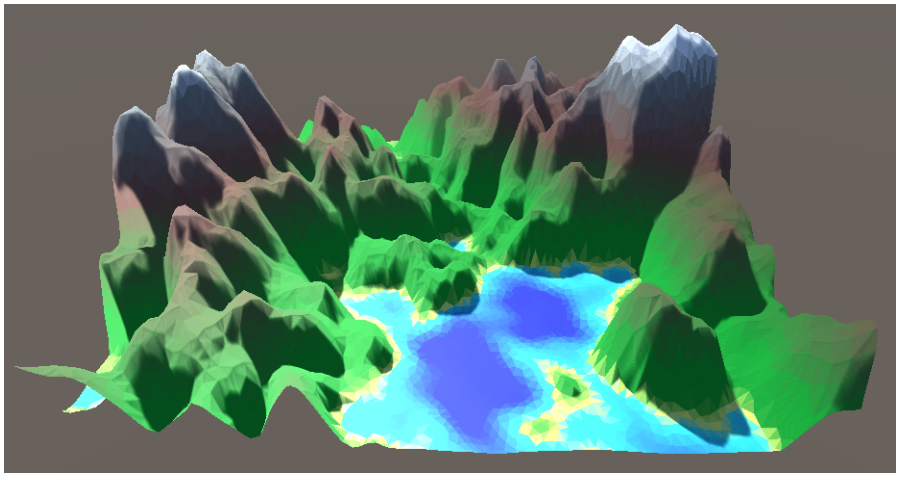

Figure 1: *Low poly terrain from the implementation in this project.*

1 Introduction

This report presents and implementation of procedural low polygon terrain generation. The project was heavily inspired by a video presentation by Kristin Stock named "Procedurally Generated Low-Poly Terrains in Unity"[1]. In the video she explains how she developed an implementation in Unity which generates a low polygon terrain by performing the following steps:

1. Generate vertices randomly, but with a minimum distance between each other using poisson disc sampling[2].
2. Triangulate the the vertices using Delauney triangulation, which results in triangles with maximized minimum angles[3][4].

3. Generate a heightmap using Perlin noise, using the heightmap data to decide the height for each vertex[5].

4. Color the triangles by taking the average height of the triangle vertices and sampling a gradient to decide the color of the triangle.

This project was performed with the goal of replicating this implementation in order to generate a low polygon terrain of its own.

2 Implementation

2.1 Mesh Generation in Unity

Unity has a mesh class which can be used to create meshes from scripts[6]. In order to create a mesh one typically submits vertex data and face data. The minimum amount of vertex data required is simply the position of the vertex. The face data is usually triplets of vertex indexes specifying which three vertexes in the vertex data should form triangles.

The following snippet is one of Unity's own examples on how to create a quad consisting of two triangles[7]:

```
Mesh mesh = new Mesh();

Vector3[] vertices = new Vector3[4]
{
    new Vector3(0, 0, 0),
    new Vector3(1, 0, 0),
    new Vector3(0, 1, 0),
    new Vector3(1, 1, 0)
};
mesh.vertices = vertices;

int[] tris = new int[6]
{
    // lower left triangle
    0, 2, 1,
    // upper right triangle
    2, 3, 1
};
mesh.triangles = tris;
```

Listing 1: *Simple example on how to create a quad from a script in Unity.*

For this project, the mesh class was used to create the terrain. By randomly placing the the vertices in the XZ-plane and then setting the Y value of the vertex by sampling a height map a terrain can be procedurally generated. The

problem with randomly placing vertices is that if they are too random the triangles in the mesh can become very small, which is solved using Poisson disc sampling.

2.1.1 Vertex Generation using Poisson Disc Sampling

When randomizing vertex positions in the XZ-plane, vertices can end up very close together or even on top of each other if the placement is completely random. Poisson disc sampling is a method that produces randomly placed vertices that have a minimum distance between them. The algorithm uses three types of points:

1. Candidate points
2. Active points
3. Inactive points

The algorithm also takes the input of a user defined radius r, which is the minimum distance desired between points. If the radius is a small number, the points will be allowed to more tightly packed, and thus more points can be generated.

When a point is first generated, it is a candidate point. If the point is not within the specified radius of any other point it is accepted and added to the list of active points. The first point generated will be a candidate point that is instantly accepted since there are no other points that it can be within the raidus of.

The algorithm then chooses a random point from the list of active points and tries to generate a new candidate point somewhere between the radius r and twice the radius $2r$. As before, if the candidate point is not within the radius of any already accepted point the candidate point is accepted. However, if none of the generated candidate points in a set amount of tries can be accepted the selected active point is set as inactive and not used for further candidate point generation. The point can be seen as completely covered by other points.

Mike Bostock has created a nice visualisation for how the process works, available online[8].

For this project, an existing implementation of Poisson Disc Sampling created by Sebastian Lague was used. The implementation was created for a video presentation on Youtube[9] and the code is available on GitHub[10].

2.1.2 Triangle Forming using Delauney Triangulation

The process of deciding which vertices that makes up triangles is called triangulation. Delauney Triangulation is a variant of triangulation that maximizes the minimum angle of the three corners of the triangle[3]. It does this by making sure that none of the vertices are inside the circumcircle of any of the triangles generated in the triangulation.

In short, Delauney triangulation can be described as a method that ensures that the triangles look nice since generation of sliver triangles are avoided. Together with Poisson disc sampling one gets a vertex and triangle generation that ensures that triangles are generated with a desirable size and shape.

Writing an entire triangulation program was a bit too ambitious for this project, so an existing Delauney Triangulation library named Delaunator[4] was used and adapted for Unity.

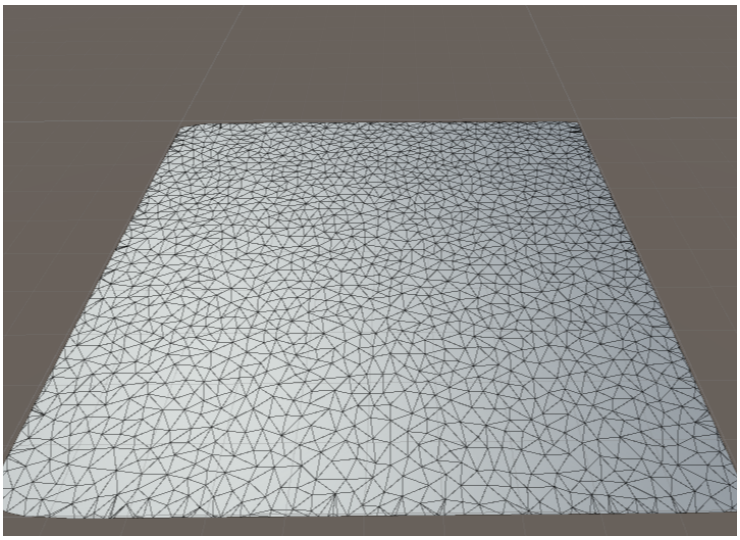

Figure 2: *The mesh after vertex generation and triangulation, shown as a shaded wireframe.*

2.2 Heightmap Generation

By generating vertices and triangles the 2d basis of the terrain is completed. In order to get the third dimension - height - we need to generate information about what height values to set for each vertex. For this project, Perlin noise was used. Perlin noise is a gradient noise where the values are not completely random, but instead consists of "waves"[5]. Perlin noise is commonly used for terrain generation since it forms a coherent pattern that can be nicely translated into height data.

An issue when using Perlin noise is that it does not fully capture how a terrain looks. By just using a regular Perlin noise the terrain will look very wavy and featureless. A common example on how to improve this is to use octaves. When using octaves one samples a Perlin noise several times, but lets each sample influence the terrain less. The first octave forms the big features such as the general shape of a mountain range, while the later samples form smaller features such as peaks in the mountain or even smaller as it goes on,

like boulders or pebbles. The following figures show how octaves add up to form a nice terrain.

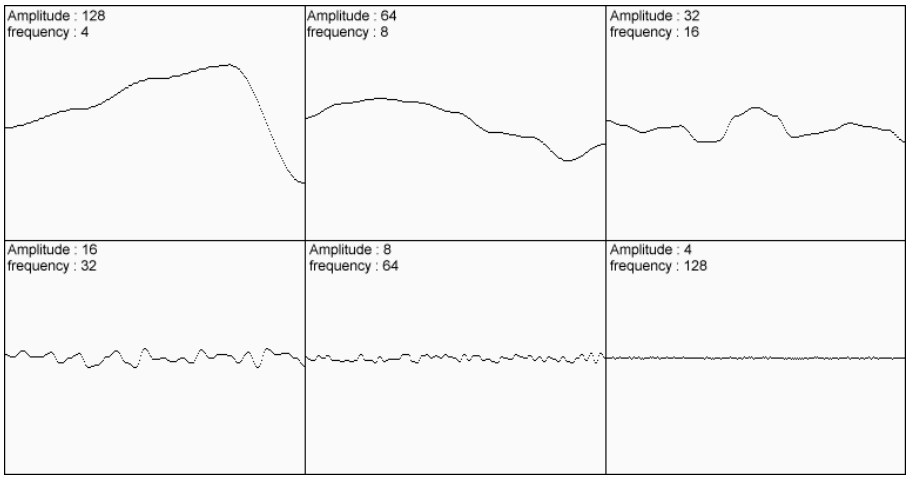

Figure 3: *Example images by Hugo Elias[11] showing six different Perlin noises with decreasing impact on the final product.*

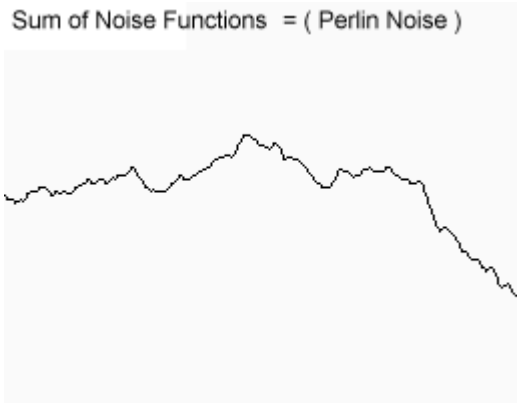

Figure 4: *The final sum of the noises looks better as terrain data.*

Sebastian Lague[12] has a good video presentation on the basics of octaves in Perlin noise and Adrian Biagioli[5] has a nice introductory article on the subject with references to an article written by Hugo Elias[11]. All of these inspired the methods used for Perlin noise in this project.

Unity has its own two dimensional Perlin function available in a library[13], which was used in this project. An inspector property where the amount of octaves could easily be controlled was added, and the Perlin noise was implemented within a function that samples the set number of octaves and adds them

together to form the height data for the terrain.

The values retrieved from Unitys Perlin noise function are between 0 and 1 by default, so the height values from the Perlin function after going through all of the octaves were translated into the range -0.7 to +1.3 so that 0 could be used as the sea level. The triangles were then colored depending on their height according to values set in a gradient, which will be explained in the next chapter.

Finally, positive values were then scaled upwards by a height value that could be set in the Unity inspector while all values below 0 were clamped towards 0 in order to shaped the sea level at approximately the height 0.

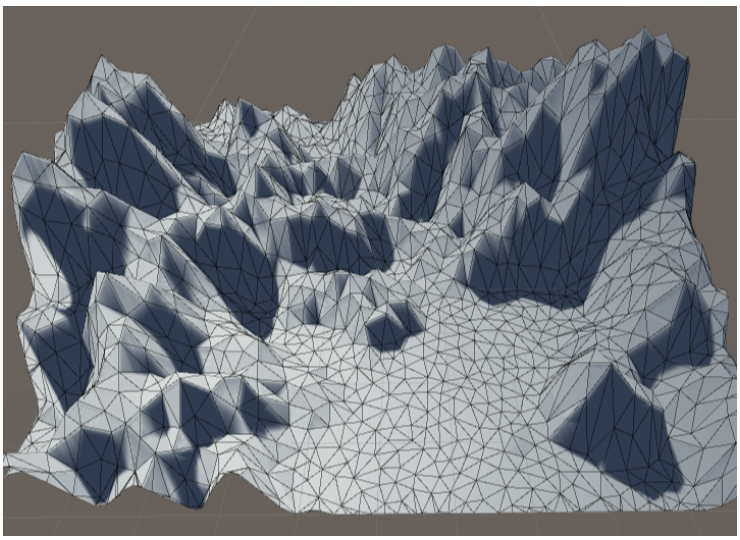

Figure 5: *The mesh after setting the height values using Perlin noise, shown as a shaded wireframe.*

2.3 Mesh Coloring

In order to color the terrain, a gradient was used. Gradients exist in Unity as a property that can be added to the inspector. Since the heightmap values were translated to be in the range -0.7 to +1.3 that had to be accounted for when sampling the gradient. The coloring of the triangles in the mesh was performed in the following steps:

1. Get the height of the triangle by averaging the height of its three vertices.
2. Sample the gradient using the retrieved height value in order to get a color.
3. Set the color of the three vertices to the color sampled from the gradient.

Figure 6: *The gradient used in the final version of the implementation.*

For the final implementation the following gradient was used.
The colors chosen were selected using the following idea:

1. Dark blue = deep sea.
2. Light blue = shallow sea.
3. Yellow = beaches.
4. Green = forest area.
5. Brown = lower part of mountains.
6. Grey = upper part of mountains.
7. White = snowy top of the mountains.

The result was a nicely looking terrain, in line with the objective of the project.

3 Results and Conclusion

After sampling colors from the gradient and coloring the triangles in the mesh the result is a nicely looking terrain. The exact appearance of course depends on a lot of variables used throughout the process. Some examples are:

1. The minimum radius between points when performing Poisson disc sampling.
2. The number of allowed sample attempts per vertex in Poisson disc sampling.
3. The number of octaves used when sampling the Perlin noise.
4. The chosen colors in the gradient.

Changing any number of these variables would change the appearance of the terrain. If desired, one could as an example add more polygons to the terrain by allowing more closely positioned vertices.

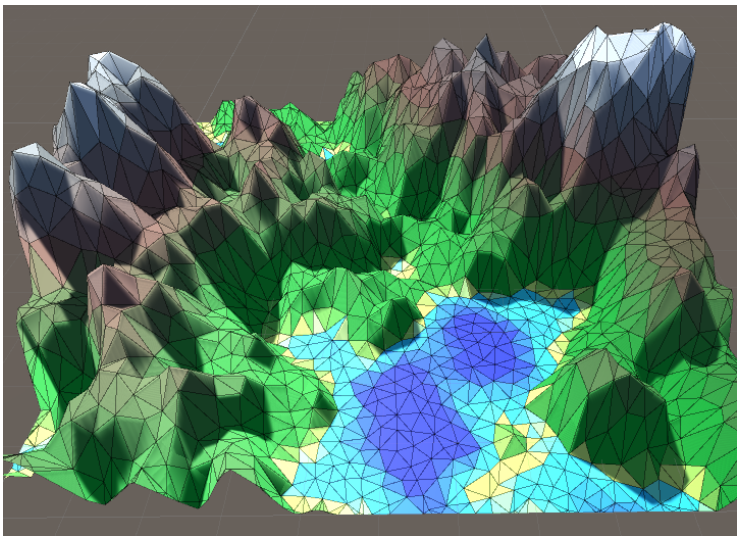

Figure 7: *The mesh after all steps are completed and the coloring has been added, shown as a shaded wireframe. Note that the number of polygons is lower in this example compared to the introductory image in the beginning of the report in order to better visualise the terrain generations steps.*

Overall, the used concepts work very well when procedurally generating a low polygon terrain. Implementing all of them would be tedious work however, and using existing libraries was definitely a key in making the work load in the project reasonable. Implementing a Delaunay triangulation in particular would be very heavy work. The triangulation library used had some issues however. When increasing the area of the terrain there occured situations where the terrain wouldn't get fully generated even though the Unity mesh should be able to fit all of the vertices and triangles. This issue most likely originated from the triangulation library but trouble shooting it was deemed out of the scope for the purpose of this assignment.

When using reasonable settings the terrain generation worked very well and in summary the goal of procedurally generating a low polygon terrain was definitely achieved.

References

[1] Kristin Stock. *Procedurally Generated Low-Poly Terrains in Unity*. Youtube. 2020. URL: https://www.youtube.com/watch?v=sRn8TL3EKDU.

[2] Robert Bridson. *Fast Poisson Disk Sampling in Arbitrary Dimensions*. 2007. URL: https://www.cs.ubc.ca/~rbridson/docs/bridson-siggraph07-poissondisk.pdf.

[3] *Delaunay triangulation*. Wikipedia. 2020. URL: https://en.wikipedia.org/wiki/Delaunay_triangulation.

[4] *Delaunator*. 2019. URL: https://github.com/mapbox/delaunator.

[5] Adrian Biagioli. *Understanding Perlin Noise*. 2014. URL: https://adrianb.io/2014/08/09/perlinnoise.html.

[6] *Mesh*. Unity. 2020. URL: https://docs.unity3d.com/ScriptReference/Mesh.html.

[7] *Example - Creating a quad*. Unity. 2020. URL: https://docs.unity3d.com/Manual/Example-CreatingaBillboardPlane.html.

[8] Mike Bostock. *Poisson-Disc II*. 2019. URL: https://bl.ocks.org/mbostock/dbb02448b0f93e4c82c3.

[9] Sebastian Lague. *[Unity] Procedural Object Placement (E01: poisson disc sampling)*. Youtube. 2018. URL: https://www.youtube.com/watch?v=7WcmyxyFO7o.

[10] Sebastian Lague. *Poisson-Disc-Sampling*. 2018. URL: https://github.com/SebLague/Poisson-Disc-Sampling.

[11] Hugo Elias. *Perlin Noise*. 2016. URL: http://www.arendpeter.com/Perlin_Noise.html.

[12] Sebastian Lague. *Procedural Landmass Generation (E03: Octaves)*. Youtube. 2016. URL: https://www.youtube.com/watch?v=MRNFcywkUSA.

[13] *Mathf.PerlinNoise*. Unity. 2020. URL: https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html.