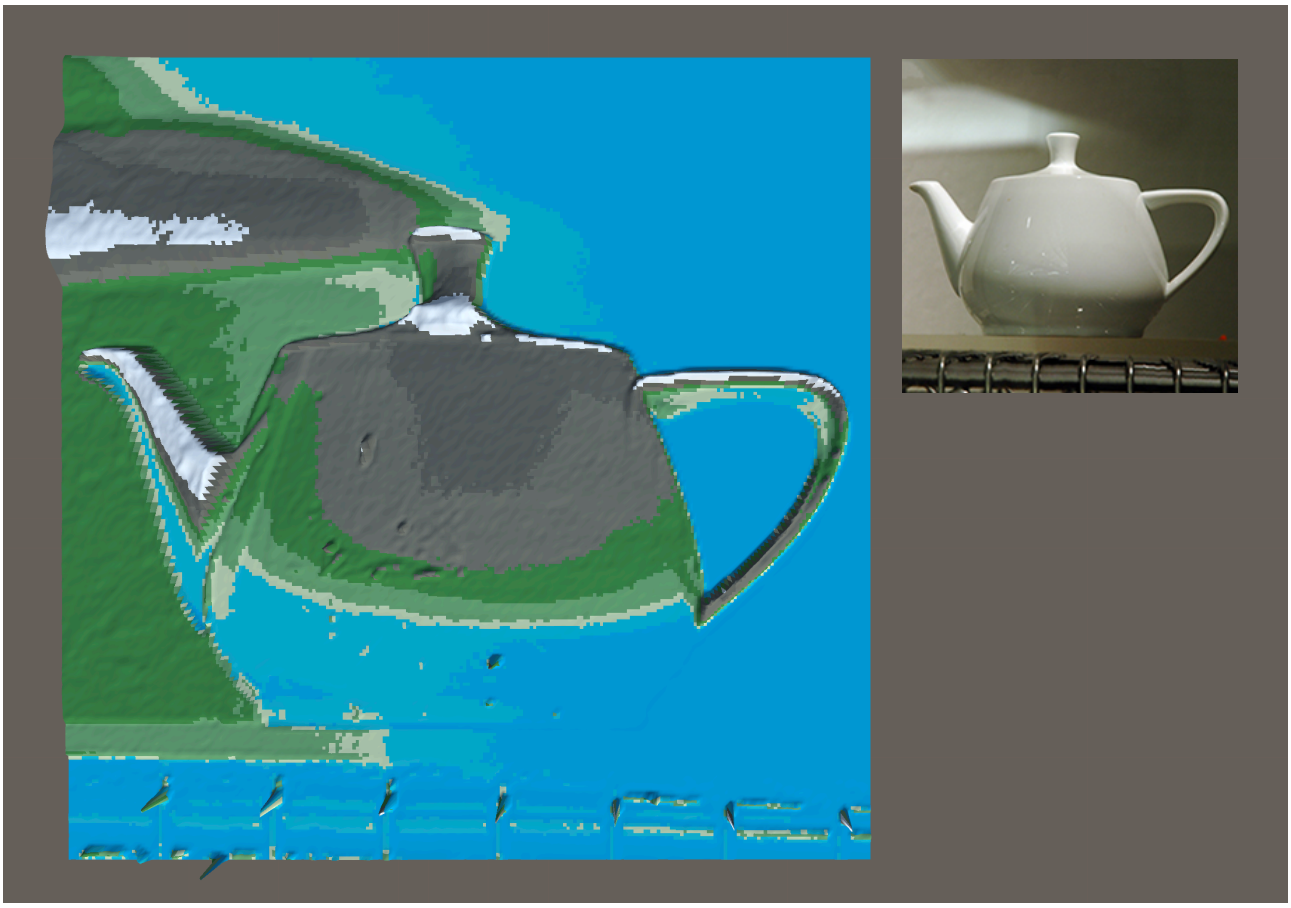
Mathematics and Graphics for Computer Games 2

Assignment 3
27 March 2017

Mesh Manipulation
Terrain Generation



Contents

Project Overview

Code Repository and Video Links

github.com/MatthewDuddington/MG2_01_MeshManipulation

youtube.com/watch?v=ORAHOnaKFkY

Aims

- Develop a program to manipulate / generate mesh vertices to form a terrain surface.
  - Provide an option to utilise pseudo-random noise to define the surface.
    - Implement the noise function based on the improved Perlin function.
  - Also provide an option to define the surface from a user provided image / photo.
- Enable mesh simplification of the terrain's structure for a different Levels of Detail at set distances.
- Enable area specific amendments to the generated content via masks.
  - Create borderless island maps with an edge mask.
- Apply a coloured texture that references the surface positions of the vertices.
  - Utilise barycentric interpolation to create smooth surface transitions between colours.
- Provide the option to apply the terrain generation to both a flat surface and also a sphere.
  - Create smooth transitions where surface edges meet.

High Level Structure

For this project, I was able to identify a developer tutorial series (Lague, S. 2016) that introduced solutions for many of the aims that I had defined. It should be noted, therefore, that I have substantially utilised this resource in the production of this project when writing my code. However, I have also ensured to supplement this by including comprehensive explanatory comments within the code to clarify my interpretation and understanding of its functionality. Classes that were written with guidance from Lague's tutorial are prefixed with "SL_".

Class: ***ImageMapGenerator***

Generates a normalised greyscale map of the provided image in a format suitable for interfacing with the SL_MapGenerator class.

Example image sources:
Utar Teapot: commons.wikimedia.org/wiki/File:Original_Utah_Teapot.jpg
author: Marshall Astor marshallastor.com
Crystal: pixabay.com/en/rock-crystal-crystal-crystal-quartz-238075/
author: Antranias

Class: **SL_Noise**

Creates a 2D map of noise data using a Perlin function. Provides mutable variables for:
- *Octaves* (the number of noise iterations which will represent different levels of detail)
- *Persistence* (how quickly the 'strength' of the amplitude influence from each subsequent octave decreases, i.e. the relative scale of details described by that octave; e.g. mountains vs boulders vs pebbles)
- *Lacunarity* (how quickly the 'sharpness' of the frequency of each subsequent octave increases which defines how complex the details of each feature become; e.g. smooth hills vs jagged mountains or water smoothed rock vs weather beaten, highly textured rock)

Also enables the height and width of the map to be defined, the scale of the resulting noise, an offset for the noise's resulting map, the seed which feeds the pseudo random number generator which provides the input for the Perlin function.

By default, the range of values between max and min possible heights of the noise map are known precisely for a discrete terrain chunk. However, these values will vary for adjacent chunks, and thus even with the same function and seed, after normalisation, some variation will be apparent in adjacent edge vertices. Therefore, an optional 'Global' normalisation option is available, which uses an estimated max height based on the combined amplitude across all the octaves, and an assumed min clamped height of 0.

Class: ***SL_MapGenerator***

This is the main interface class for the terrain generation. It passes user values to the SL_Noise class and uses the generated noise map to produce a corresponding set of height and colour map data. User defined Terrain Types are stored here, which describe the properties and qualifications of each terrain area.

Rather than processing all calculations as a single sequential sequence, for better performance, the Map Generator places the tasks of determining the map and mesh data into parallel threads that can be called upon by the SL_EndlessTerrain class. Data is returned from this class rather than complete textured meshes, as Unity can only process the updating of meshes etc. within its main thread.

Class: ***SL_MapGeneratorEditor***

Responsible for ensuring the latest variable changes in the MapGenerator instance are automatically applied to the values being passed on to generate the mesh / plane within the editor view, outside of game time.

Class: ***SL_MapDisplay***

Responsible for refreshing the drawing of the texture colour to the noise map preview plane in the editor and, likewise, the editor's preview mesh.

Class: ***SL_TextureGenerator***

Creates a texture from a given colour map or height map for applying to a mesh or displaying on the editor preview plane.

Class: ***SL_MeshGenerator***

Generates a mesh data set based on inputted map data. Like the SL_MapGenerator class, this returns data rather than building the described mesh so that simulations data productions can be handled on separate threads.

Determines how many vertices a mesh should include by taking into account the Level of Detail variable. Within this setup, 241 * 241 is used because, after discounting the first vertex, 240 contains several factors that can be used to simplify with - each time skipping 0, 2, 4, 6, 8, 10 or 12 points. 65,025 is Unity's inbuilt cap on vertices per mesh (255 * 255).

The generation function establishes the required number of vertices, defines a 3D vector position, a corresponding 2D UV position and builds a triangle for each vertex (other than those at the far right or bottom edges).

Class: ***SL_MeshData***

A data holding class that can be called on by both threaded processes (for the constructor and triangle definition stages) and also the main Unity thread (for the mesh creation stage). Contains data defining the vertex positions, set of triangles and the normalised UV coordinates.

Class: ***SL_EndlessTerrain***

Controls the activation and updating of the set of Terrain Chunks that form the overall map. Determines how far away the viewer's position is from each chunk object and calls for an update when the viewer has moved sufficiently far from their position, at the time of the previous update. Instantiates new chunks where the chunk has never been seen before, or reloads old chunks that are being returned to.

The user also defines the distances for each level of detail degree to apply vis the LoDInfo struct array 'detail levels'.

Class: ***SL_TerrainChunk***

The physical game objects within the game space that represent chunks of the terrain map. Contains the current mesh being displayed, but also the set of lower / higher level of detail meshes that have been calculated. Terrain Chunks check their distance from the viewer whenever they are updated and turn themselves visible, invisible or swap to a different level of detail as appropriate.

Class: ***SL_LoDMesh***

Holds the mesh for a specific terrain chunk at a specific level of detail. These begin in an empty state and are only filled with mesh data when that particular mesh is required by the players position for the first time. After that, the stored mesh can be reloaded to save recalculating it again upon the player's return.

Class: ***SL_FalloffGenerator***

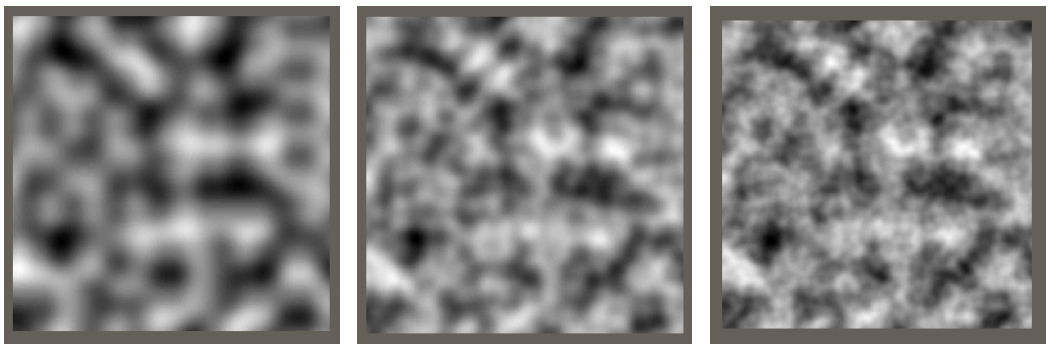
Creates an edge mask that can be optionally combined with the map data to force the terrain to resolve to zero values at the edges. Essentially has the effect of always creating a group of islands that are surrounded by ocean.

Process and Learning

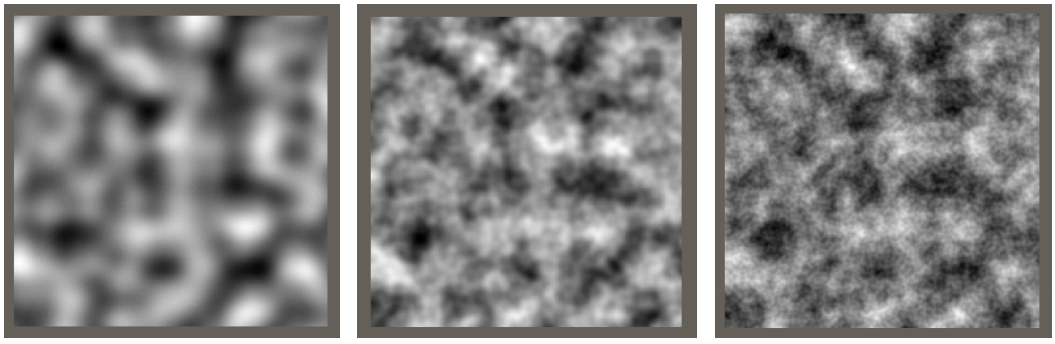
Process Record

The initial process of developing the terrain system involved setting up the greyscale data map that serves as the input for both vertex heights and the coloured terrain regions. Unity has an inbuilt 2D implementation of Perlin noise, into which can be passed coordinate points from a conceptual grid. When normalised to fully occupy the spread from 0 to 1, this floating point data can be resolved onto a greyscale texture.

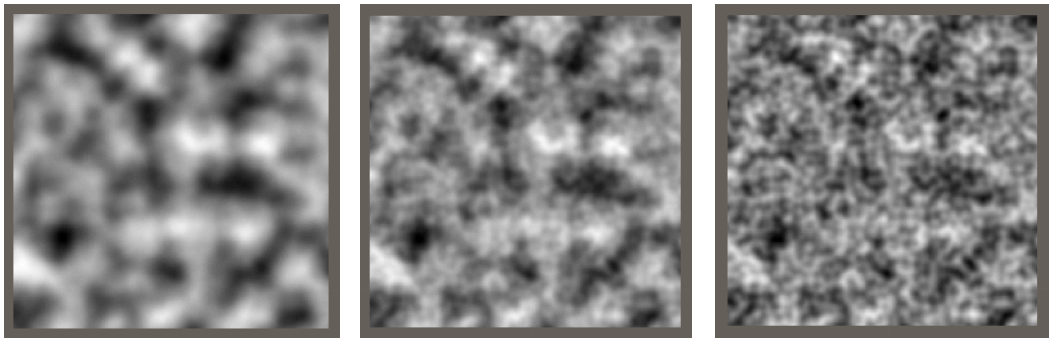
Several concepts were introduced to me:
Octaves
Persistence
Lacunarity
(My explanatory interpretation of these is provided above in the SL_Noise class description)



1, 2 and 3 Octaves using the same data

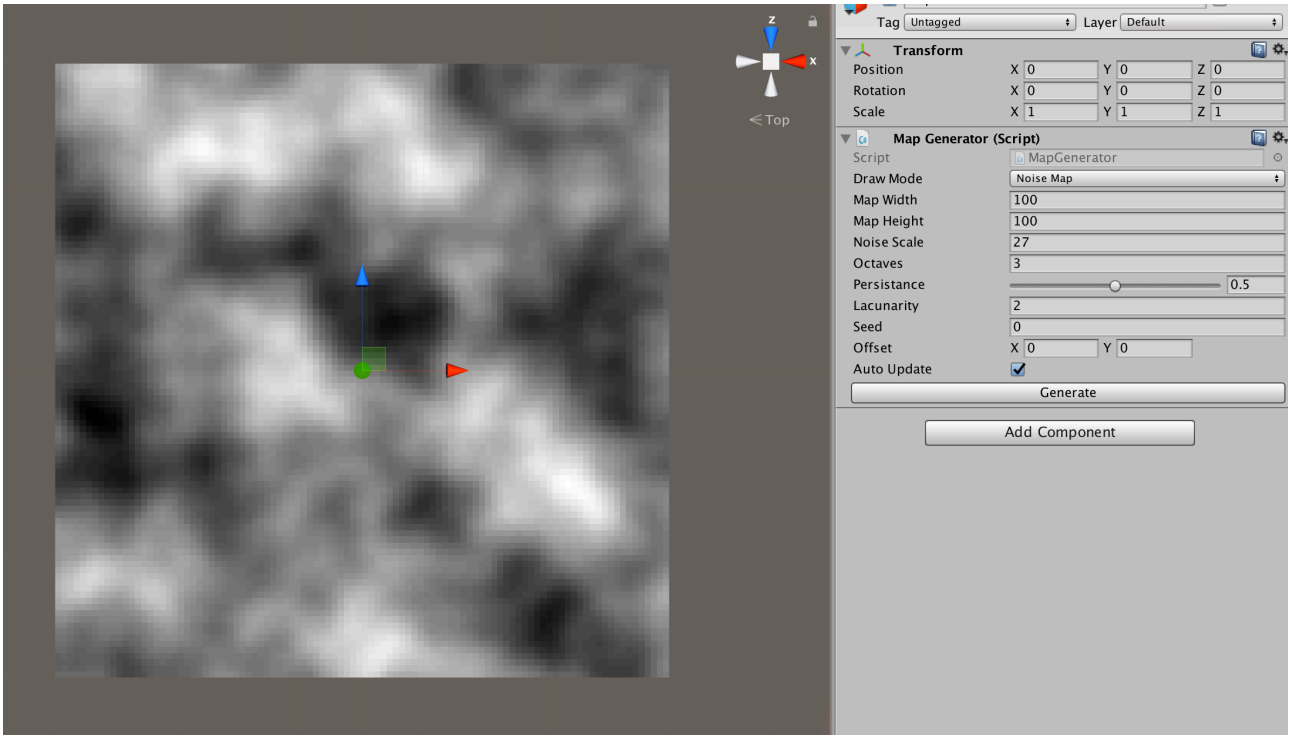


Lacunarity of 1, 2 and 3 using the same data

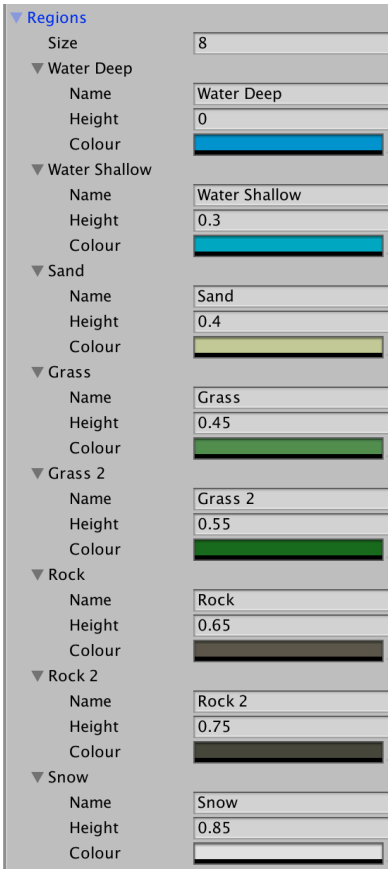


Persistence of 0.2, 0.5 and 0.9 using the same data

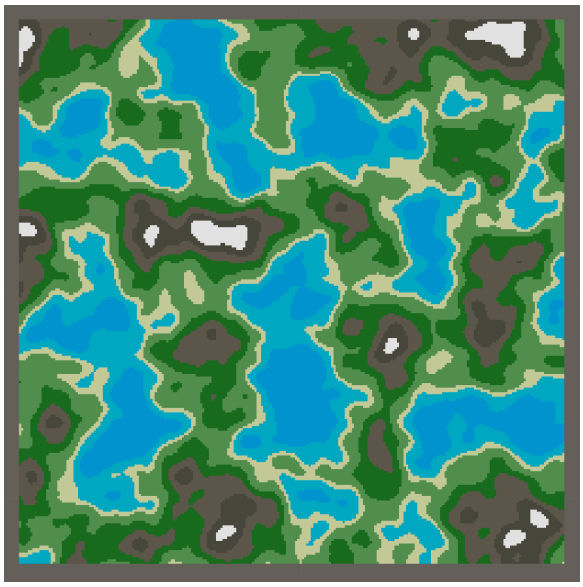
By changing these values the resulting noise can be tailored to a wide range of organic, cloudy structures. Perlin noise creates smoother noise than purely random values because it is influenced through combining both a random grid coordinate distance vector with a common set of random gradient vectors through a dot product. The space between these values is then interpolated, creating a smooth form of noise where values close to each other are similar.



Unity inspector interface with the noise variables exposed to the user

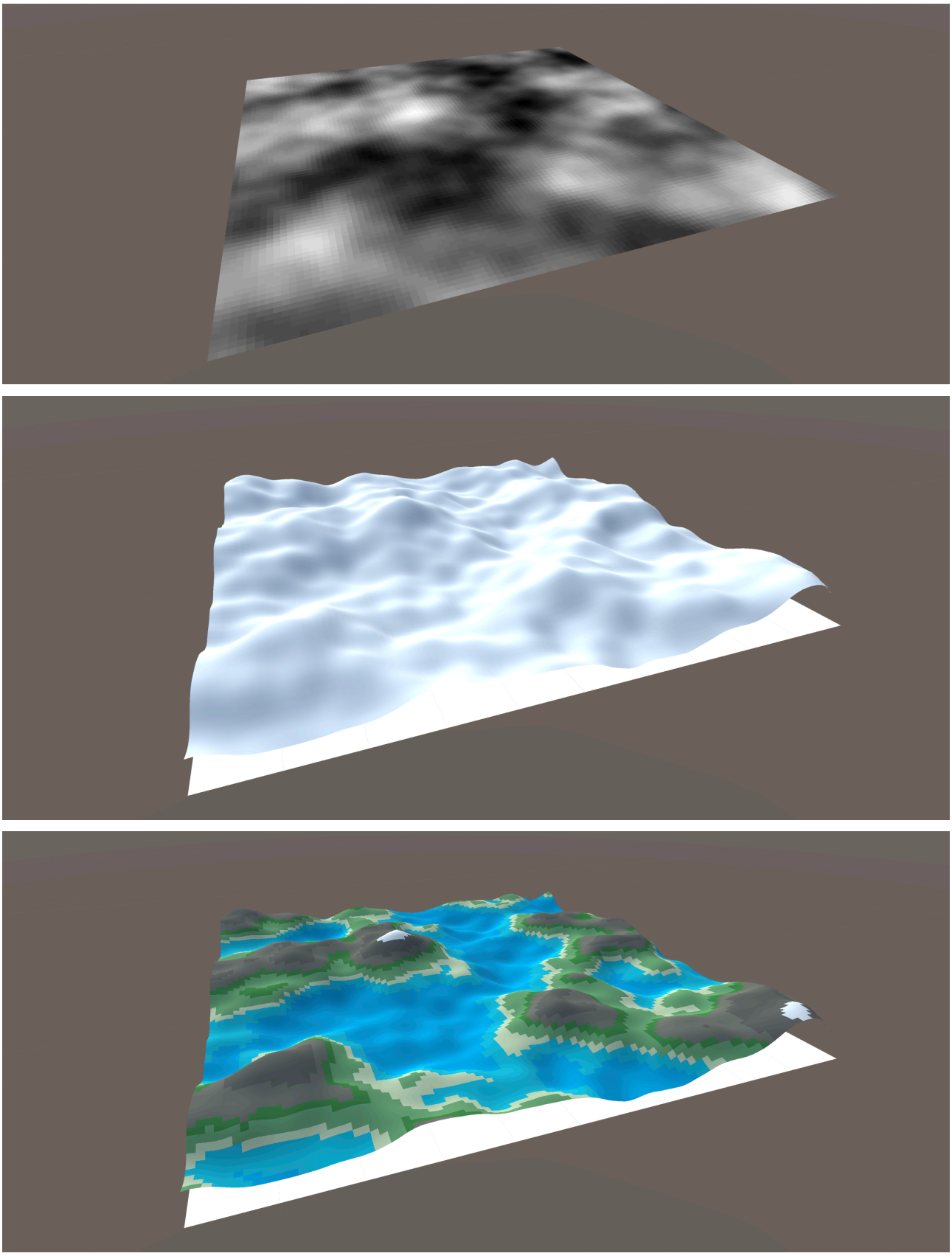


This greyscale map of values can then be interpreted as a series of regions and colours assigned to specific pixels within a texture. These regions are simply defined as subsections of the 0 - 1 range and assigned properties, such as colour.



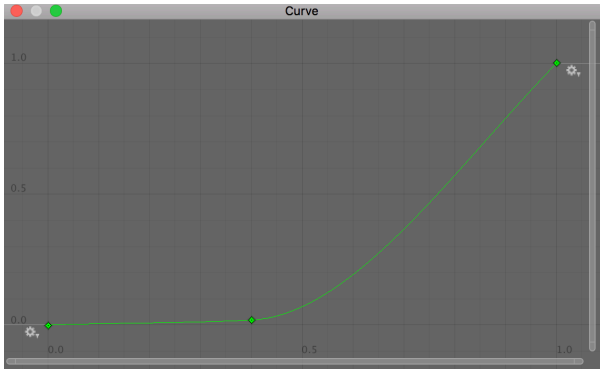
Terrain regions and resulting colour map

By assigning those same values to a height map and multiplying by a world scaler, these data points can offset the vertical position of the vertices of a mesh grid. Applying the colour map to the mesh creates the impression of a complex terrain.

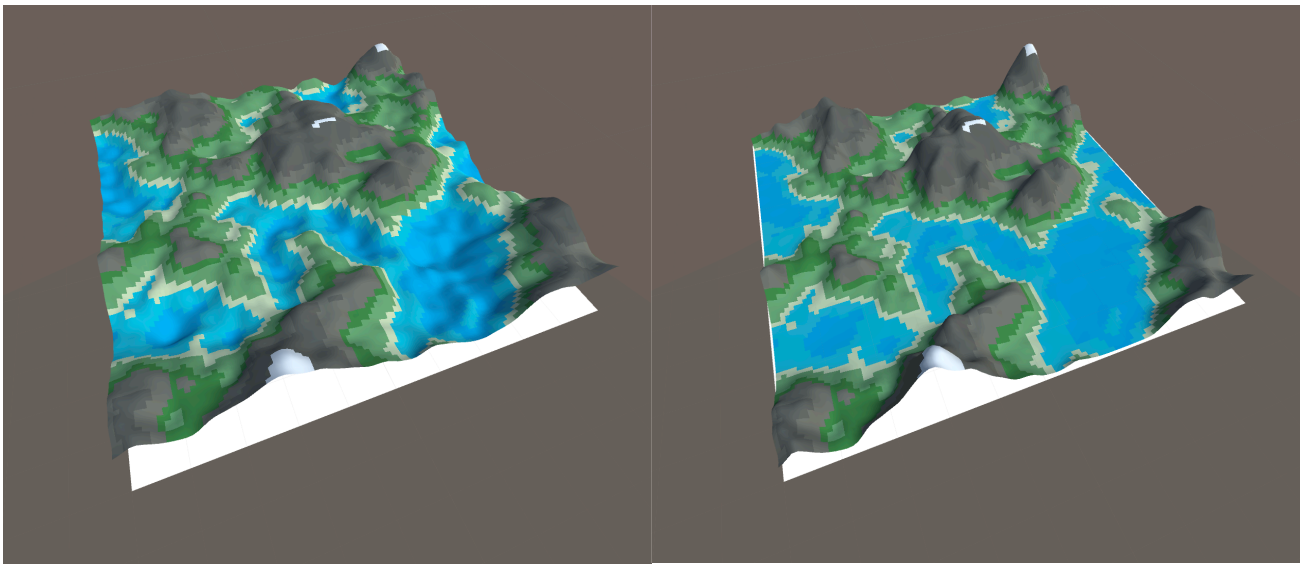


Noise map, untextured mesh and colour map textured mesh. Same base noise source.

The basic noise includes linear height detail across all parts of the mesh; this creates a strange hilly structure within sections that are implied to be water. To solve this a smoothing curve can be applied to the data that surpasses the effect of the multiplication until after the desired region is passed - in this case everything up to the 0.4 mark, where the sand region begins.

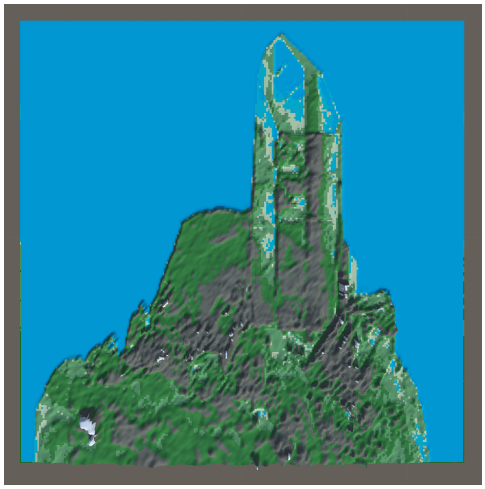


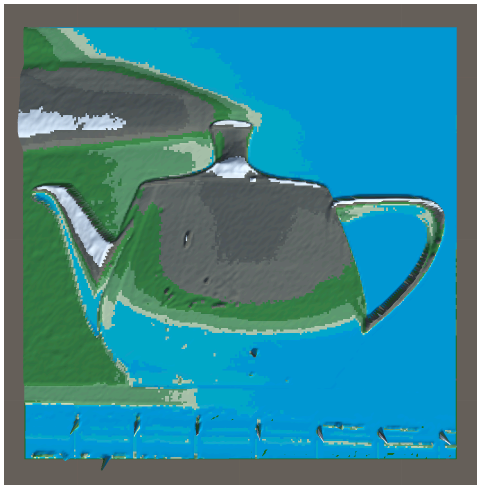
Smoothing curve

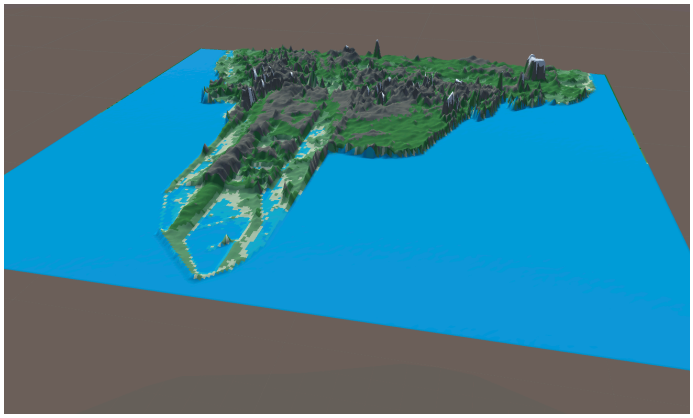


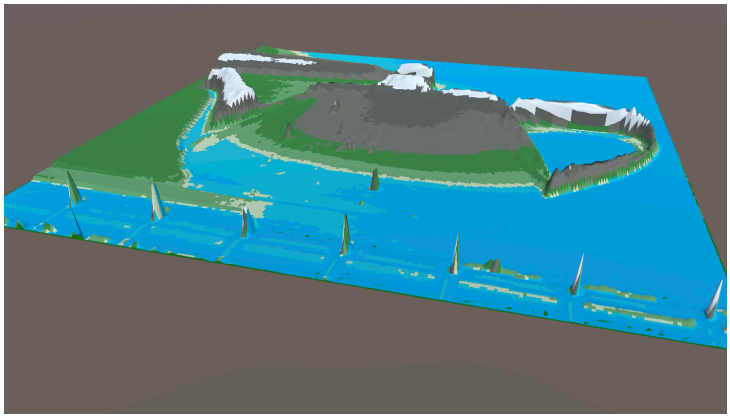
Example mesh before and after smoothing curve applied

As well as pseudo random noise, I implemented the ability to pass in an image as the greyscale map source. This opens the possibility for creating stylised levels from hand drawn maps or photographs; though some care needs to be taken to avoid extreme changes in height where the differences in contrast are significant. In a future development I would wish to implement a smoothing function to refine this.

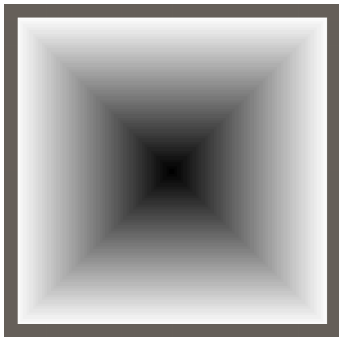




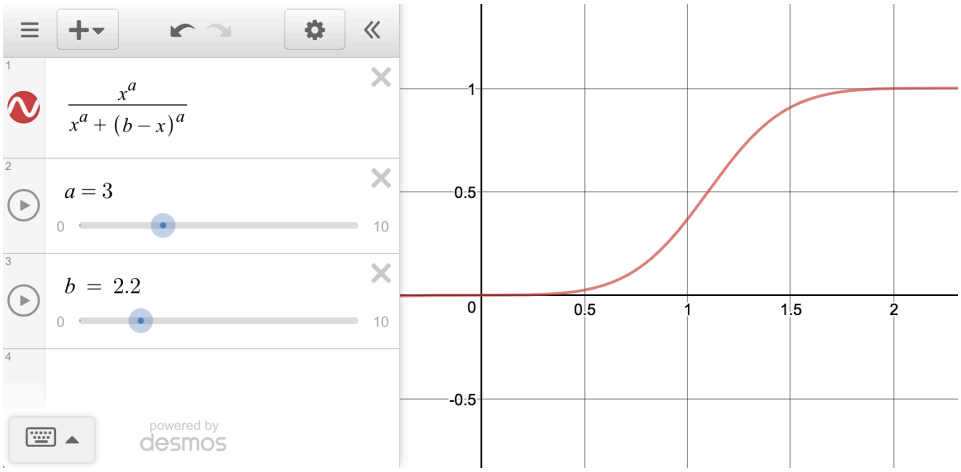




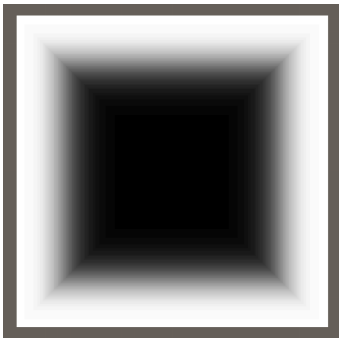
In order to create meshes that can stand alone as maps without landmasses that end partially at their edges, a falloff mask can be applied. This basically saturates the edges of the map with lower values, smoothly erasing influence from those areas. Falloff values were refined to leave the central area intact with another smoothing curve: x^a / (x^a + (b - x)^a



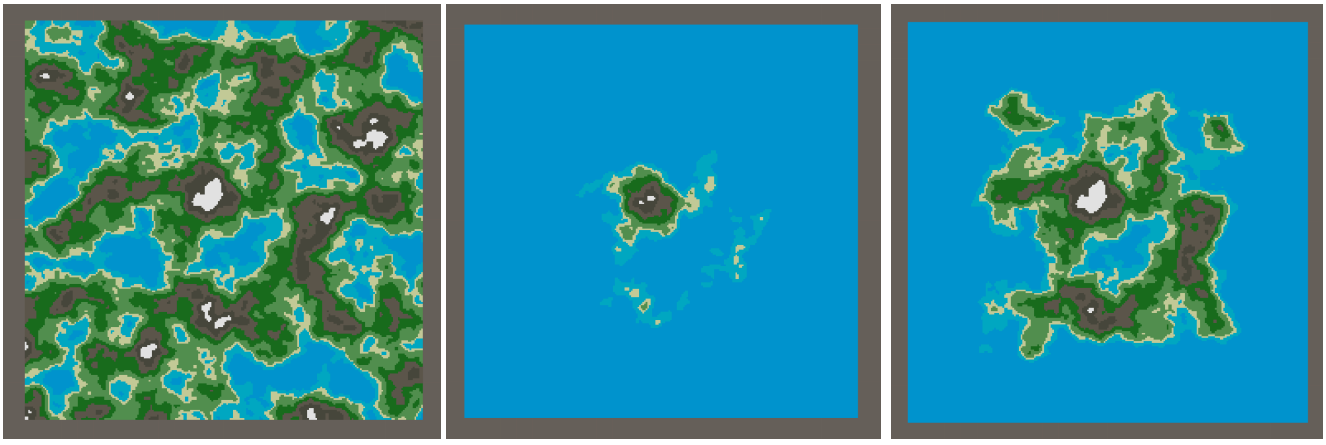
Falloff before curve applied



Falloff curve with values a = 3 and b = 2.2



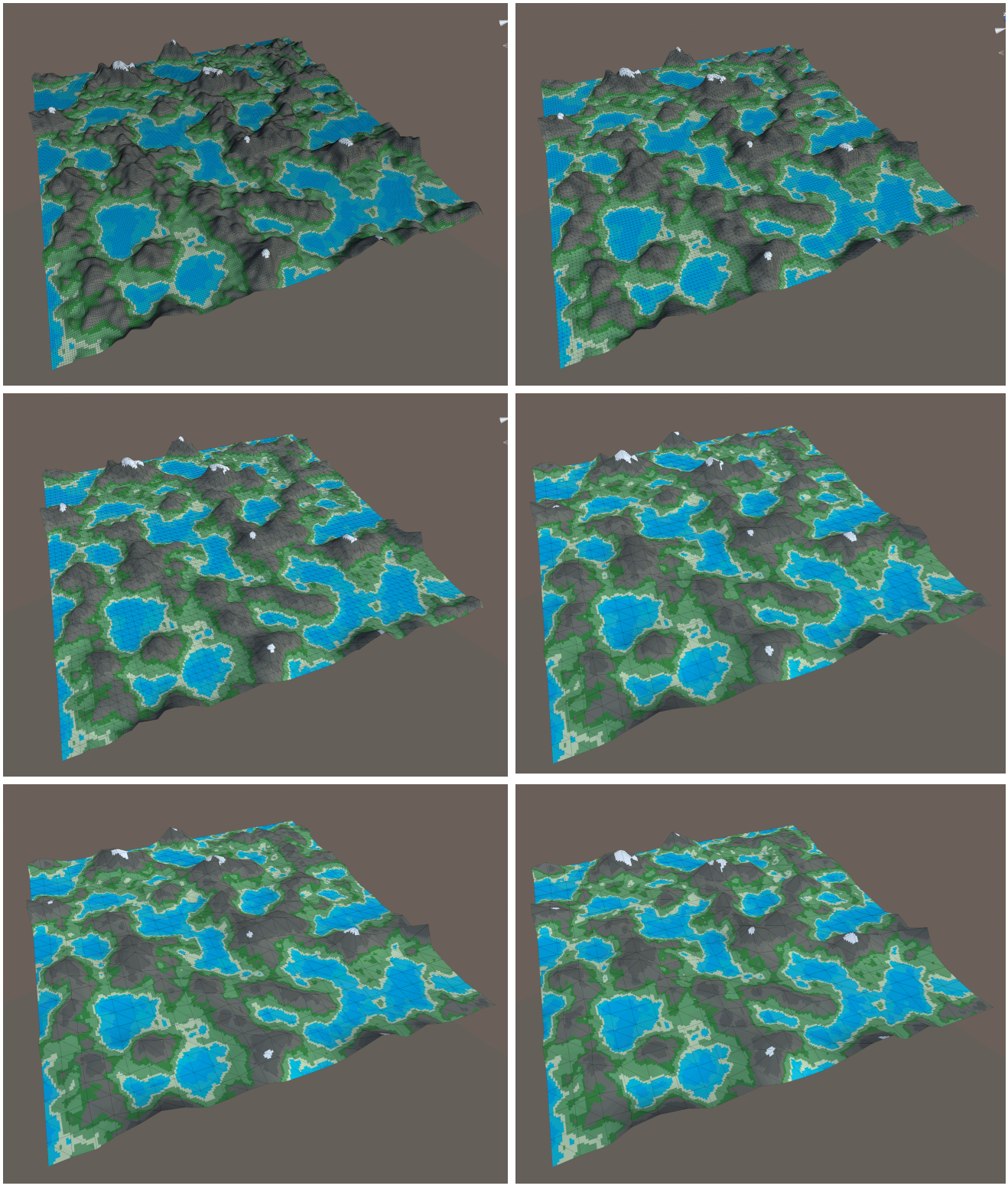
Falloff after curve applied



Example map before falloff applied, with falloff before curve applied and with falloff after curve applied

By defining a unit size for the terrain chunks of 241 * 241, it was possible to create a wide range of simplified versions of the mesh for differing Levels of Detail. This is because after subtracting the initial vertex (leaving 240) the remaining vertices can be factorised, and therefore traversed, through several different values. In this case: 1, 2, 4, 6, 8, 10 and 12.

To do this, when the mesh is constructed, data are only requested for the coordinates of vertices that will be included. When the viewer is far enough away, the lower resolution meshes can be used to conserve memory and processing time, while nearby meshes retain their full detail.



Level of detail meshes from the same data for vertex windows of 1, 2, 4, 8, 10 and 12

Within future exploration of this I would like to follow up on the research I began but unfortunately ran out of time before implementing in the time available. Firstly, on this list would be transferring the flat offset terrain maps onto a spherical body using the Cube Sphere method (Flick 2015, Nishry 2014). Furthermore, I would like to refine the colouring method with the use of barycentric coordinates, similar to the Phong model, to smoothly interpolate between vertex colours, removing some of the jagged appearance.

Bibliography

Biagioli, A. (2014) *Understanding Perlin Noise*. (accessed March 2017) Available online: flafla2.github.io/2014/08/09/perlinnoise.html

Dunn, F. and Parberry, I. (2011) *3D Math Primer for Graphics and Game Development*. 2nd ed. CRC Press.

Flick, J. (2015) *Cube Sphere - Better Roundness*. (accessed March 2017) Available online: catlikecoding.com/unity/tutorials/cube-sphere/

Lague, S. (2016) *Procedural Landmass Generation (Unity 5)*. Episodes 1 - 11. (accessed March 2017) Available online: youtube.com/playlist?list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3

Nishry, S. (2014) *Planet Generation*. Parts 1 - 2. (accessed March 2017) Available online: shaneenishry.com/blog/2014/08/02/planet-generation-part-ii/