A Maya Terrain Generator Based on Perlin Noise

Master's Project Thesis

Aslan Saparov
MSc Computer Animation and Visual Effects
30th August 2021

Abstract

This thesis report documents the terrain generation, the research on different methods and techniques possible to generate a terrain as well as the implementation of it in this project. The Maya tool produced for this project aims to generate a few types of terrain based on Perlin noise and found in the real world and appear semi-realistic as well as provide artists or game developers the means to produce areas for levels. The tool consists of a user interface with a number of parameters to control the output such as the frequency and amplitude of the noise forming the shape of the terrain and produces the results that are different on every generation. The tool then textures the generated mesh with different levels of biomes such as water, land, mountain and others. The tool achieves its goal to produce simple landscapes that are realistic and "believable" to an extent as well as providing a certain control over the outcome with the user interface. It could be noted the terrains generated with this tool could be particularly suitable for low poly games or games with low level of detail such as a strategy game for example where the majority of the camera time is arguably spent far away from the ground. It is recognised, however, that the tool could be improved with the use of more advanced techniques of developing terrain landscapes such as polygonal map generation or the use of erosion, etc. The technique itself, Perlin noise terrain generation, could be improved in ways that may not be documented online or otherwise.

Introduction

Generating terrain landscapes can be dated to as early as 1982 if not earlier and has since been a broad topic in computer graphics and video games industry. Though not limited to just terrain procedurally generating realistic terrain could be of a high interest to video game developers as it might be safe to assume that such production technique is favourable to resource and time management. Therefore, learning and applying such techniques might prove useful.

Previous Work

Some of the earliest work in landscape generation can be dated back to 1982 with the release of Star Trek 2: The Wrath of Khan and attributed to Loren Carpenter who developed a software for generating and rendering fractally generated landscapes (Nnart 2022). That was pertaining to cinema, whereas in regard to video games the first game to feature 3D fractal landscapes was "Rescue on Fractalus!", a 1985 first-person shooter game where the player must carefully traverse the fractal landscape's valleys in order to find injured pilots (Ahoy 2018). To touch on terrain generated specifically using Perlin, Simplex or any other type of noise maps, there wasn't much to be found other than demos made in Unity and the following examples shown in the figures 1 and 2, the work of which most of this project is focused and based on.

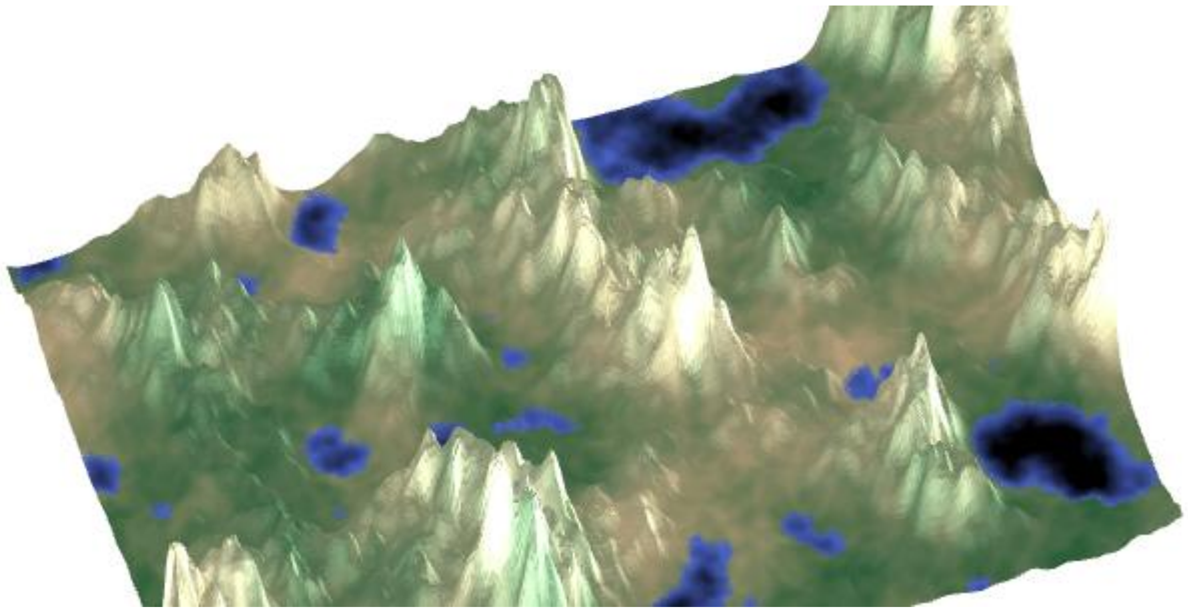


Figure 1. Perlin noise based terrain (RedBlobGames 2022)

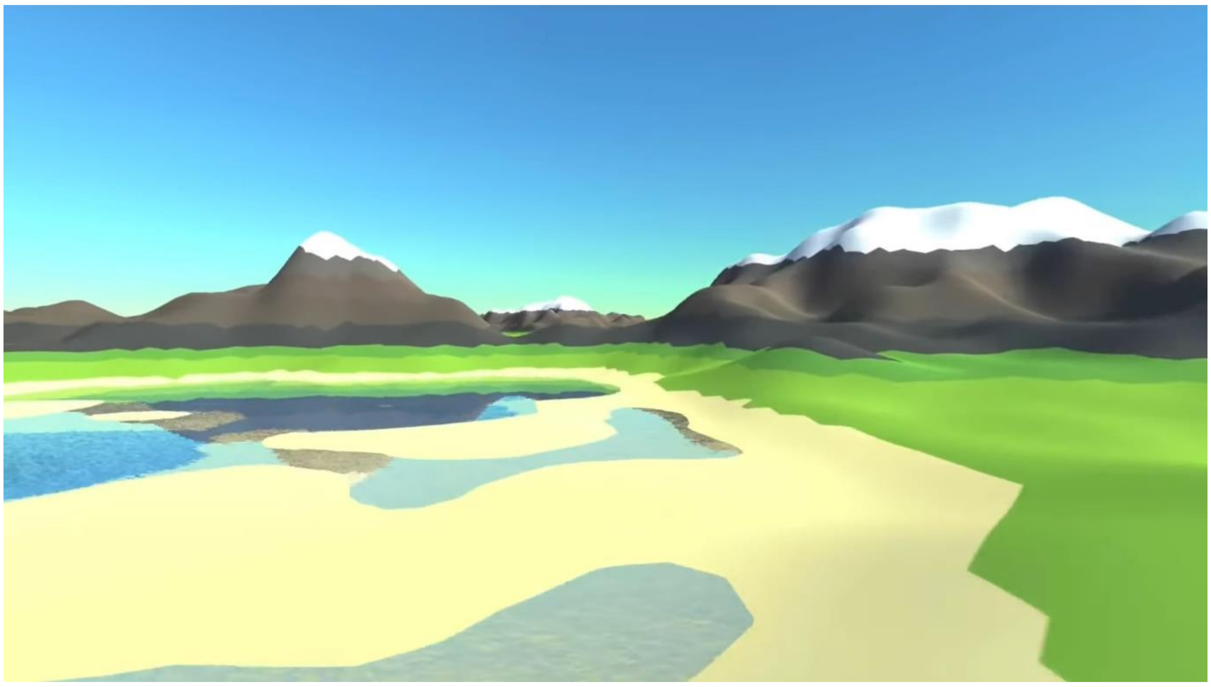


Figure 2. Terrain generation by Sebastian Lague (2022)

Technical Background

One of the first works that was found during the background research and then later applied in this project was Sebastian Lague's (2022) series of Unity 3D tutorials covering terrain generation using Perlin noise. Perlin noise is “a type of coherent noise” where “changes occur gradually” and could be used as a height map where values would commonly go from 0 to 1 where 0 represents dark areas or areas of negative elevation and 1 represents light areas or places of elevation which leaves valleys or hills in the middle. The figure 1 below demonstrates what Perlin noise usually looks like. It could be argued that Perlin noise is most suitable for generating terrain since gradual changes seem more natural. For the purposes of this project, it was not required to obtain more details on the calculation of Perlin noise itself. It might be noteworthy to add that Perlin noise can be described as a wave as shown in the figure 2. Lague (2022) goes on to describe a Perlin noise wave as an outline of a mountainous region which can then be used as a general shape for the terrain while overlaying more noise maps or octaves can add more detail to the terrain as to not for it to appear "too smooth".

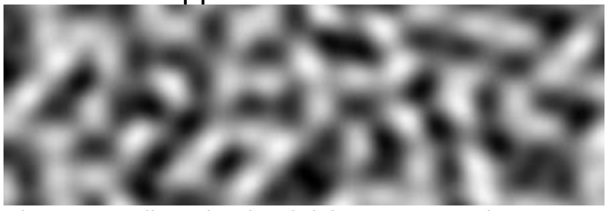


Figure 3. Perlin Noise (RedBlobGames, 2022)

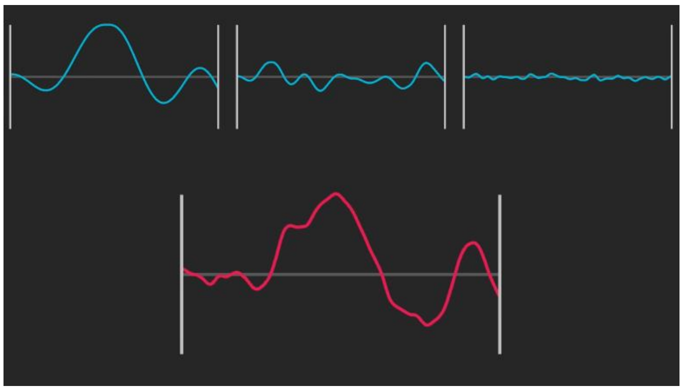


Figure 4. Perlin Noise as Waves or Octaves (Lague, 2022)

Those waves or noise maps can be also referred to octaves. To reiterate the basics of generating terrain shape is to overlay several octaves on top of each other. However, as was noted before, the first octave is used as basic outline of the overall terrain, therefore it would be reasonable to assume that it is desirable to preserve to an extent. Therefore, the frequency and amplitude of each progressing octave would have to be controlled. To be more precise, the frequency controls the length of the waves on the noise map in x-axis while amplitude determines height of the waves on the noise map in the y-axis.

To control the frequency and amplitude of each subsequent noise level it is suggested to use lacunarity and persistence.

Lacunarity variable "controls increase in frequency of octaves" making frequency go from the lowest to highest value in each new noise map adding more detail or waves. Lague (2022) demonstrates assigning frequency of the first octave lacunarity raised to the power of 0, the second octave lacunarity to the power of 1 and so on. Increasing the frequency with each map.

The Persistence variable therefore "controls decrease in amplitude of octaves" or reduces amplitude in each subsequent octave. This ensures that every new octave has less effect on the overall shape of the terrain. Similar to lacunarity, the persistence value is determined by raising the amplitude to the power of 0 in the first octave, to the power of 1 in a second octave and so on reducing the amplitude in every new map.

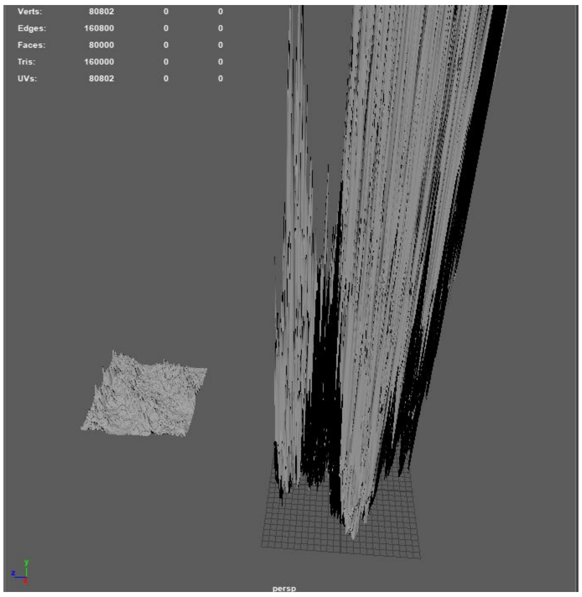
It is recommended to keep persistence variable from 0 to 1 for the optimal terrain generation.

Solution

This project is about creating a tool in Maya that generates terrain based on the given noise maps through mesh manipulation or displacement of vertices. The goal for this project was also to give a user the control over the terrain generation. This is achieved with a user interface written in PySide 2. The UI allows the user to control such parameters as width and height divisions and number of octaves to control the detail of the mesh.

Mesh Generation

To utilise the above-mentioned information on Perlin noise and use them as height maps for the terrain it is required to retrieve values from Perlin noise ranging from 0 to 1. To do that in Maya, this project makes use of previous coursework in Pipeline and Technical Direction which displaces every vertex on a mesh based on an alpha channel value of a corresponding UV of a given image. In this project's case, the same methodology was used to evaluate and calculate an array of values ranging from 0 to 1 taken off of one noise map per loop. To evaluate on that, one element of the array would read and add values of the corresponding UV on all iterations and all noise maps. The value on every iteration, however, would be reduced by persistence. Having combined all noise values together, every value in the array would divide by the amplitudes added together from every octave. To show the necessity of the above, figure 4 shows the absence of division by amplitudes.



```
Verts:      80802    0    0
Edges:     160800    0    0
Faces:      80000    0    0
Tris:      160000    0    0
UVs:        80802    0    0
```

Figure 5. Development bug on the right

Redistribution

Following the above, the tool produces terrain similar to what is demonstrated by figure 5. Although this terrain could potentially be deemed complete and realistic, the absence of valleys or flat areas restricts the variety limiting the use of the tool.

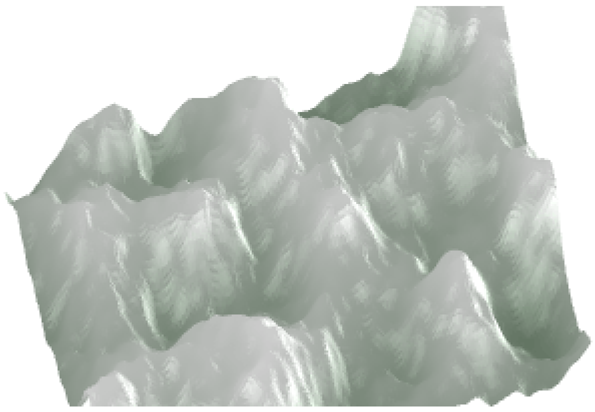


Figure 6. Terrain without redistribution

RedBlobGames (2022) recommends to raise the elevation or processed vertex height values to a power. “Higher values push middle elevations down into valleys and lower values pull middle elevations up towards mountain peaks”. This is where the modifier is also introduced allowing to alter the height of the elevations even further.

Terraces

Terraces can be formed by “rounding the elevation to the nearest of n levels”. In the implementation it would look more like this “round(elevation*n)/n”. The lesser the n value the less terraces are formed due to the higher range of elevation becoming equal. The parameter used to control number of terraces in the UI is named “Round Levels”

Ridged Noise

The Ridged noise technique mentioned in RedBlobGames (2022) blog was used sparingly as it was not fully understood. However, it is used to create sharp ridges and it is especially noticeable on higher elevation values such as mountains. The tool’s UI allows to turn the ridged noise on and off.

Island formation

The island formation according to RedBlobGames (2022) can be achieved by utilising the following two principles:

1. “A distance function assigns a distance to every position on the map, from 0 at the centre to 1 at the border.
2. A shaping function (as used in the Redistribution section) takes an elevation as input and chooses a new output elevation.”

The blog presents several ways to implement a distance function such as Square Bump, Euclidean, Hyperboloid, Trig Product and more. For the implementation of the tool in this project, Square Bump distance function was used. Since the distance function considers the width and height of the map and all noise maps as well as the terrain plane are of the same width and height distance is calculated on the first iteration or first octave only.

The linear shaping function is then applied further into the terrain generation after processing all noise maps combined. The UI has a island modifier parameter which allows to change how much the shaping function is affecting the island.

The UI

Most of the UI was written with the help of Chris Zurbrigg's (2022) courses on using PySide 2 for Maya. Other than the tutorial availability, PySide 2 was also chosen due to the fact that it was installed and available to use with the version of Maya provided in the labs. Although most of the knowledge regarding PySide 2 came from Zurbrigg’s (2022) tutorials, Stack Overflow was used substantially to aid with any import errors or questions regarding any particular class in PySide 2 or any problem with Python in general

Texturing

It was originally intended for the user to be able to upload texture files for the tool to apply to the generated mesh. An uploaded texture file could have been chosen for any level or biome of a terrain such as water, land, mountain or any user defined levels. Instead, a decision was made to keep texturing to the tool itself both due to the time constraints and lack of knowledge on the subject.

During the research on procedural texturing methods in Maya only a few options were found such as selecting and colouring each vertex with Maya's "polyColorPerVertex" command, generating a texture file with Python Imaging Library or PIL or using QImage class from PySide 2.

Maya's "polyColorPerVertex" command was the first method taken to develop a texture file for the terrain mesh in Maya. The process behind applying the command and colouring of every vertex was selecting a vertex after its position or height was displaced through noise and processed further. Having selected a processed vertex, its height is then evaluated to see where it lies within a range of hardcoded biome thresholds which then leads to colouring the vertex accordingly. The colours at the time were hardcoded as well for testing purposes. Having used "polyColorPerVertex", it was found that this texturing process took a considerable amount of time after the generation of the mesh, possibly tripling the amount of time required for the complete terrain generation. It's not clear whether the additional time taken is longer due to the fact that Maya's select command is ran to process every vertex and could be unoptimized or that the "polyColorPerVertex" command itself is a time consuming command. It might be possible to attribute such a delay to both.

PIL or Python Imaging Library is a python package that, as the name suggests, is used for image processing. Although Python Imaging Library could have been used to develop file textures for biomes of the terrain, it was decided to avoid using it partially due to the fact that it requires an installation process which was not possible to replicate due to the time constraints.

Closer to the end of the development of the project texturing was added with QImage from PySide 2. The tool uses QImage class to produce a texture with the width and height matching the number of width and height subdivisions of the terrain poly plane respectfully. The image is then flipped vertically as the image produced the first time is flipped and does not match the terrain. The image is then saved with a PNG file format and then the tool then creates a file node, a lambert shading node as well as a shading group. The saved image is uploaded into the file node the colour output of which is then connected to colour input of the lambert shader which is then subsequently connected to the shading group’s surface shader. The resultant material is then applied to the terrain mesh.

Saving created textures

To make texturing work on any platform “QDir.homepath” was used when naming the texture generated by QImage. It was intended for the tool to generate more than one terrain in Maya; therefore, textures are saved in the home directory on any platform reducing the risk of failure.

Originally, every time terrain generation procedure was executed, a new texture image would be generated. The problem with that was that every new texture would overwrite the existing file and therefore previously created terrain meshes would adopt the new texture and discard the old one due to the fact that every texture would have the same name “QImage”.

To solve that and to give the image texture a new name every time a generate button is clicked and keep old textures applied to the old meshes without overwriting them, python's time module is imported and used. To be more precise, every new texture would be named as the amount of time passed in milliseconds since January 1st 1970. That method was chosen instead of using the random python module functions to avoid possible a small chance of repetition.

Profiles

Profiles are a feature of this tool that allows the user to select a set of pre-existing values which are applied to the given parameters to formulate a look of a certain terrain type as well as act as a means to convenience. To get into more detail, the existing profiles are named "Island", "Hills, "Mountains" and "Desert" which aim to achieve the look of the named terrain types. This project stores each profile as a dictionary which is then stored in a list of dictionaries. This allows for modularity and creation of new profiles if needed. Each profile stores such data as number of octaves, lacunarity, persistence, the height modifier, exponent, round levels of the terrain, island boolean which determines whether or not the terrain should be mostly surrounded by water, a ridge noise boolean and a list or an array of levels or biomes.

Array of biomes

As mentioned above, each profile contains a number of parameters used to generate terrain, one of which is the array of biomes or list of levels as it is named in the

project code. The list of levels contains arrays of four elements used to describe a biome which includes a threshold value, red, green and blue colour values.
A threshold value is used in a loop and compared against the height of every vertex in a terrain plane mesh to determine whether or not the colour associated with that threshold should be applied to a pixel corresponding to the vertex in question. In case, the height of the processed vertex happens to be higher than the threshold value, the loop moves onto the next biome array where the process is repeated with that array's threshold value. The threshold value in the last biome in the biomes array should be the highest and should therefore be covering any other vertices unaffected by previous biomes. On another note, the threshold values are calculated in such a manner that they can be expressed as percentages. That is achieved by determining what the highest height value among all vertices is and then counting a percentage off of that.

To make texturing more modular it should be possible to add new levels or biomes to an existing list of levels. If statement running through a list of levels to colour a pixel.

Switching mesh and texturing options in the UI

A design decision was made to keep the dialog window of fixed size. Therefore to fit all of the UI widgets within the same window it was decided to separate them into ones that allow the control of mesh generation and ones that are responsible for texturing the mesh. To realize that two radio buttons were created - "Mesh" and "Texturing". Those buttons being radio buttons ensures that one of them is selected which is necessary for the switching to work. Selecting one of the radio buttons hides all UI elements tied to either "Mesh" or "Texturing". For example, when "Texturing" is selected parameters such as width and height subdivisions, number of octaves, lacunarity, persistence, modifier, exponent, number of round levels, island and ridge noise booleans are hidden while combo box for levels or biomes, red, green and blue channel values are shown.
It could be argued that the use of tabs would be preferable to using radio buttons as the widgets and layouts containing those widgets could be hidden in less lines of code. However, it was decided against it as it would have been consuming.

Switching from MEL to Python

This project is building off of the coursework for pipeline and technical direction unit which was written in MEL scripting language. However, Maya's Python was chosen to write this project as it potentially has more functionality and allows for use of third-party libraries as well as it being a useful learning experience. To expand more on the topic of functionality, at some point during the development it was required to clip a number of decimal points off a certain variable, the option that could not be found in MEL, but could, however, be found in Python.

Choosing Maya for this project

Maya 2020 was chosen as a primary software for this project as it was intended to learn more about the software and apply terrain generation techniques in an area seemingly untouched by the subject.

Conclusion

To reiterate, the goal of the tool written in this project was to procedurally generate a terrain mesh with minimal input and effort from the user and produce results that can be considered semi-realistic. The tool also provides texturing capabilities which develop a texture file using PySide 2 with dimensions set by the width and height subdivisions of the plane acting as terrain. The texturing itself aids in an objective to create visually semi-realistic terrain by applying biomes such as water, land, mountains and other biomes.

It could be argued that the goal has been achieved since the tool produces simple terrain landscapes which are visibly believable in a short amount of time. The amount of time it takes to produce a terrain with 100 subdivisions in width and height is around 4 to 5 seconds whereas generating a terrain with subdivisions of a 150 by a 150 in width and height takes around 10 seconds. In addition, the generated terrain could be exported into a game engine albeit with a few tweaks and without the use of the tool, manually.

The project is missing a few features mentioned in the RedBlobGames (2022) document such as biome variation with the use of noise maps and geometry placement of such meshes as rocks and trees. The blog also hints that the noise maps could provide more options and could be even used in ways that are undocumented which is something to consider in the future.

Of the feature goals that were set for the project itself: it was intended for this python tool to be imported as a maya plug-in, to add new levels to the levels list, add new profiles and write the changes to a text file that could be read in later on the tool start up. Those features were not complete and added to the project, but this could be something to implement in the future to improve the project and make it a complete tool.

Another possible way to improve this project’s artifact after its submission is to use more advanced techniques such as erosion.

Using Maya's noise textures as a base for generating a terrain only one drawback was noticed: as the frequency in higher octaves increases and reaches a certain point a uniform pattern can be observed in noise textures through Hypershade window. Maya's Perlin noise texture node appears to be missing an attribute that controls its variety. Using a dedicated python library to generate Perlin noise could have been an improvement and solved the problem with uniform patterns in octaves with higher frequencies. To counter that, however, the detail in later octaves is affected by the lower amplitude and therefore could be considered negligible.

One minor detail that the tool can fix is to manage the created shading nodes better. Although it is intended for the tool to keep terrain and its material in the scene as opposed to replacing it with the newly generated one, the tool still removes the previous noise textures.

This leads into another suggestion on how to improve the current project and its tool. Currently there is no way to affect the created terrain by adjusting the widgets of the user interface during run-time. Doing so could increase the convenience of the tool in that it would allow for immediate feedback and increase the precision and accuracy of the user to achieving the result that they would want.

For the future reference, Houdini can be taken as an example of generating realistic terrain with height fields. Houdini has a workflow for generating terrain using its heightfield nodes which follows these steps: Massing Model, Seeding, Lobing, Remapping, Up-sampling, shaping, re-seeding, erosion and scattering or adding shaders (Houdini n.d.)

References:

Ahoy, 2018. *RetroAhoy: The Secret of Monkey Island* [online]. Available from: https://youtu.be/9F9ahZQ7oP0 [Accessed 30 August 2022]

Lague, S., 2016. Procedural Landmass Generation (E01: Introduction) [online]. Available from: https://youtu.be/wbpMiKiSKm8 [Accessed 1 August 2022]

RedBlobGames, 2022. *Making maps with noise functions* [online]. Available from: https://www.redblobgames.com/maps/terrain-from-noise/ [Accessed 4 August 2022]

SideFx, n.d. *Realistic terrain with heightfields* [online]. Available from: https://www.sidefx.com/docs/houdini/model/terrain_workflow.html [Accessed 1 August 2022]

Zurbrig, C., 2022. *Courses* [online]. Available from: https://zurbrigg.teachable.com/courses [Accessed 20 August 2022]

Available from: https://stackoverflow.com/questions/20632841/qt-horizontalslider-send-float-values [Accessed 24 August 2022]

Nnart, n.d. *History of Fractals* [online]. Available from: https://nnart.org/history-of-fractals/ [Accessed 30 August 2022]