Procedural Terrain Generation in UE4

BSc. Computer Games Programming

Clariana Justes, Oriol Marc
V8086261

Supervisor: Dr. Julien Cordry

Contents

Abstract

Procedural Generation is a feature becoming more popular each year in the videogame industry and for the infinity possibilities that it grants. In this project, using Unreal Engine 4 and the Procedural Algorithm Generation it has been created a Tool with a variety of options to customize the final result used in conjunction with a series of techniques to create high quality procedural terrain in real-time and pre-bake.

To showcase this, an easy-to-understand and well-structured interface was built and all necessary elements to visualize the product were created. This report explains the classes that were built in order to visualize the procedural terrain, how it is generated and the parameters to customize the terrain.

Acknowledgements

I would like to thank to my project supervisor Julien Cordry who provided support and advice on the product all throughout the development of this project. I would also like to thank my peers and friends on the course who provided support and feedback during the development process.

Introduction

Procedural Generation

Procedural Generation is becoming more popular each year between videogames. This is mainly because it does the work of designing and creating new assets and environments based simply on mathematical functions. This has its positive and negative side. For example, if a little and scape is needed for a game, it is better to get an actual designer to build in detail, but if what is needed is a gigantic landscape for an open world game it is probably best and more efficient to generate it procedurally. The power of making the procedural content generated more unique resides in how complex the mathematical functions that generate that content are and the customization of the options for that.

Many games have already used Procedural Generation to create new amazing environments because the huge scale of the map. Tittles like Minecraft (Mojang, 2009) use Procedural Generation to generate the game maps. No Man's Sky (Hello Games, 2016) goes even further by creating planets and galaxies. Other games use it to create procedural items and weapons like Borderlands Series (Gearbox Software).

Figure 1 – No Man's Sky

Product

The product is a custom tool developed in C++ in Unreal Engine 4 that use Perlin Noise to make possible the generation and visualization of a procedural generated terrain in in real-time and pre-baked with asset placement and semi-biomes.

The product has been built from scratch to have a higher control over all the components that generate and visualize the terrain, enabling a high efficiency and optimization. The only dependence is a Plugin called Runtime Mesh Component which help to create and modify meshes at run-time more fasten than Unreal Engine 4 do. Also using basic technique in videogames called Instanced Static Mesh for an optimization with the assets on the terrain.

Inspiration

The main reason and inspiration for choose this particular project is the immensity of possibilities give us the procedural generation whether in terrain generation, creatures generation, difficulty of game, materials and more. Also, have a great knowledge of this can open me doors in the industry because the procedural generation is in this time more used in the industry and speed up the content generation. For example, the pre-baked option accelerates the velocity of launch a new game generating a terrain adapted to the game specifications. Since I study how to do videogames, I am was really appassionato about this complexity of procedural content generation.

Figure 2 – Minecraft

Existing Technologies and Dependencies Used

Due to great interest in Unreal Engine and the potential in the industry I was decided to make the whole project was coded and compiled using Unreal Engine 4 and Microsoft Visual Studio 2017.

The Perlin Noise Algorithm is my own implementation and interpretation to avoid the number of dependencies and also has a really easy structure to change it for another different algorithm implementation.

The Plugin Realtime Mesh Component it is easy to change by the equivalent in Unreal Engine 4 called Procedural Mesh Component.

Methodology

Use the Scrum methodology to take advantage of the meetings with my project tutor to explain weekly my progress in the project, what I did the previous week, problems that I was encountering as I went along and how I solved them, besides explaining what I would do during the following week to keep a job with constant progress and in this way I could be monitored more easily and give me feedback on how I was carrying the project.

Scrum

Scrum is an agile development model. Work is divided into small phases known as sprints. Sprints will usually last 2-4 weeks, where teams will work on high priority requirements, and by the end of the sprint they will have a potentially shippable product increment.

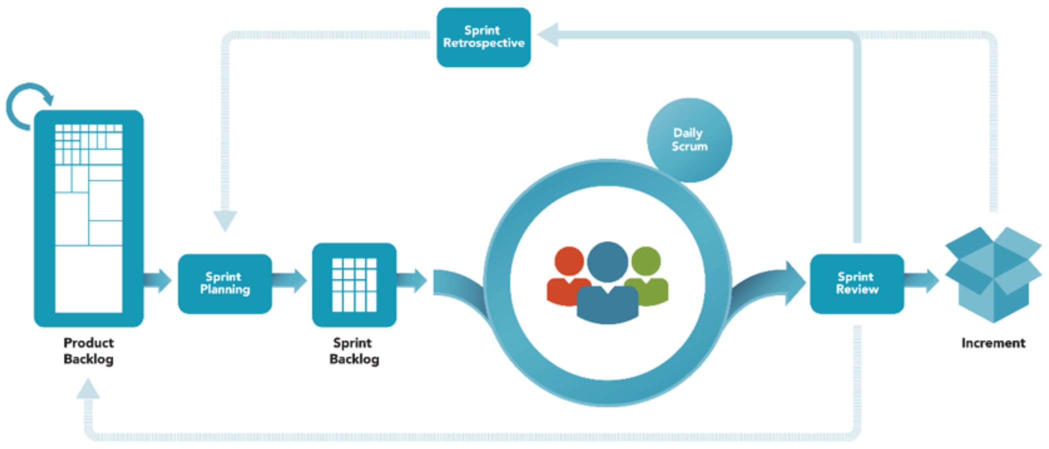


Figure 3 – Scrum Methodology

Advantages

Scrum can help teams complete project deliverables quickly and efficiently:

- Scrum ensures effective use of time and money
- Large projects are divided into easily manageable sprints
- Developments are coded and tested during the sprint review
- Works well for fast-moving development projects
- The team gets clear visibility through scrum meetings
- Scrum, being agile, adopts feedback from customers and stakeholders
- Short sprints enable changes based on feedback a lot more easily
- The individual effort of each team member is visible during daily scrum meetings

Disadvantages

Nothing is perfect, and the Scrum methodology is no exception. In some cases, Scrum is combined with other project management techniques that can help resolve some of these drawbacks:

- Scrum often leads to scope creep, due to the lack of a definite end-date
- The chances of project failure are high if individuals aren't very committed or cooperative
- Adopting the Scrum framework in large teams is challenging
- The framework can be successful only with experienced team members
- Daily meetings sometimes frustrate team members
- If any team member leaves in the middle of a project, it can have a huge negative impact on the project
- Quality is hard to implement, until the team goes through aggressive testing process

Conclusion

Being a completely individual project, the great part of the disadvantages implied by the Scrum methodology are solved by being a team with many members.
And the other disadvantages to be a project chosen by me, which gives me great motivation and great potential are also solved.

Research

Perlin Noise

Perlin noise is a type of gradient noise developed by Ken Perlin in 1983. The most common used ones are one, two and three dimensions. The frequency of the noise controls the distance at which samples are taken and it is used to control the scale. The amplitude controls the maximum and minimum values that the noise can take. In graphics programming Perlin noise can be used to generate two dimensional textures which later can be used as heightmaps for terrain generation or as a regular texture.

Heightmaps can be used to describe terrains by interpreting the values of the noise texture as heights. Perlin can be effectively used to generate a basic random terrain. From here, different approaches can be taken to modify the Perlin noise equation so as to make a terrain feel more realistic.

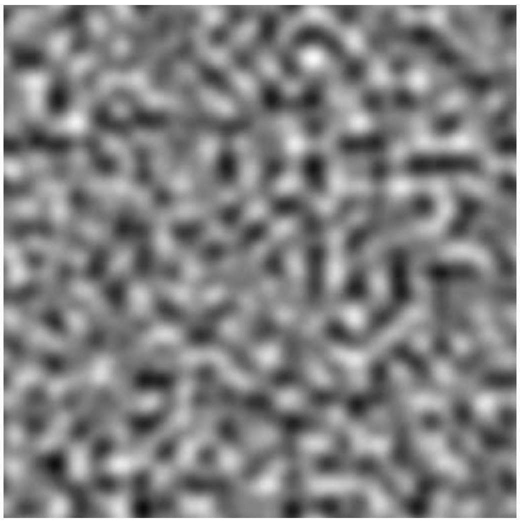


Figure 4 – Regular Perlin noise

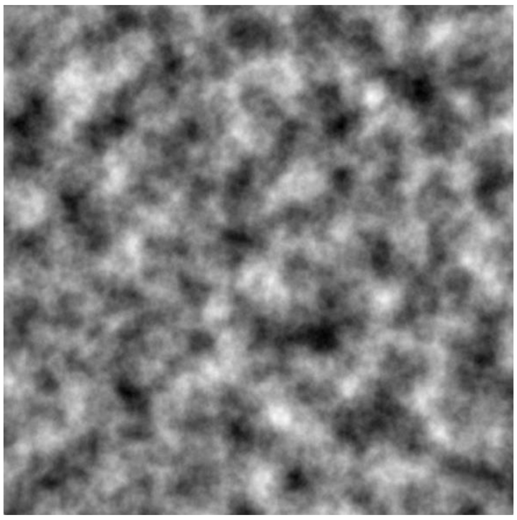


Figure 5 – Perlin noise with Octaves

The Figure 4 show a regular sample of the 2D texture of Perlin Noise.
The Figure 5 show the Perlin noise with octaves added.

It is possible to make the result of Perlin Noise more realistic / different by iterating several times over its values using different types of algorithms such as Fractals or Hydraulic erosion.

Level of Detail

The Level of Detail (LOD) involves decreasing the complexity of a 3D model representation as it moves away from the viewer or according to other metrics such as object importance, viewpoint-relative speed or position.

This technic is used to optimize the 3D environments to reducing the complexity of the models at different distances changing the number of triangles rendered an object as its distance from the camera increases. It allows reduce the consuming a huge number of resources on models far of the camera vision.

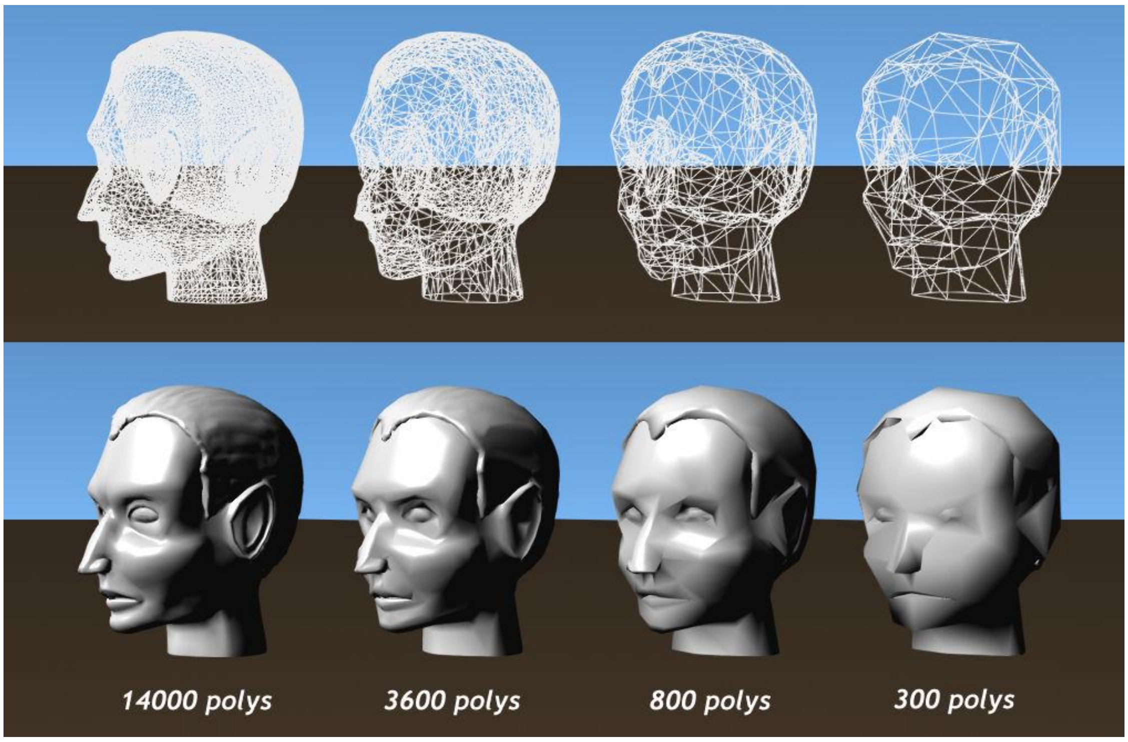


Figure 6 – Level of Detail

As seen in Figure 3, the LOD change the density of polygons is changed but also trying to keep the initial shape of the object. This technique can be used in terrain rendering due to the fact in a scene that is viewing a terrain, both close and far terrain can be viewed at the same time and do not require to have the same polygon quality. The further away the objects are the easier it is to describe with lesser polygons than the initial shape. And this simple concept allows to save resources at the rendering moment.

Tiles

The Tile involves split the terrain generation in small pieces to reduce the calculation time and the number of polygons rendered.

Combining with Perlin Noise and the Level of Detail (LOD) makes possible generate an infinity number of pieces of the terrain using the Perlin Noise value and make it works faster reducing the LOD on the far chunks.

Also allows the possibility to use a Pooling System to reduce the number of chunks on the world modifying the chunks not rendered. In this way allowing to avoid creating more memory for the generation of the land using the memory already created but not used.

Futures in Multithreading

Futures are a high-level mechanism for passing a value between threads, and allow a thread to wait for a result to be available without having to manage the locks directly.

Using the Futures and Promises allow me to manage a lot of different tasks asynchronous to speed up the terrain generation, the mathematical operations and calculations. In exception of the creation of the mesh because this not allow reserve the memory in a different thread of the main.

Mesh Instancing

In real-time computer graphics, mesh instancing is the practice of rendering multiple copies of the same mesh in a scene at once. This technique is primarily used for objects such as trees, grass, or buildings which can be represented as repeated geometry without appearing unduly repetitive, but may also be used for characters. Although vertex data is duplicated across all instanced meshes, each instance may have other differentiating parameters (such as colour, or skeletal animation pose) changed in order to reduce the appearance of repetition.

When rendering static meshes, a common CPU cost is the actual function call to render the mesh. Under the covers, this draw call often contains a reference to a VBO already loaded on the GPU, as well as transform information where to render the mesh. The call itself does not have much overhead as the real work is done by the GPU, but making this call a large number of times certainly adds up. Doing this thousands of times every frame causes CPU driver overhead. For example, imagine a wall that is made of 10000 individual bricks. This would result in 10000 draw calls.

Using the Instanced Static Mesh allow to store on the GPU memory in a Vertex Buffer the actual vertices of a static mesh. The draw call consists of telling the GPU to render the mesh at a specific transform and orientation. When rendering a batch of identical meshes, a further possible optimization would be to store the actual transforms on the GPU as well.

Using this approach, we can render all of them with a single draw call. An instanced static mesh leverages this approach.

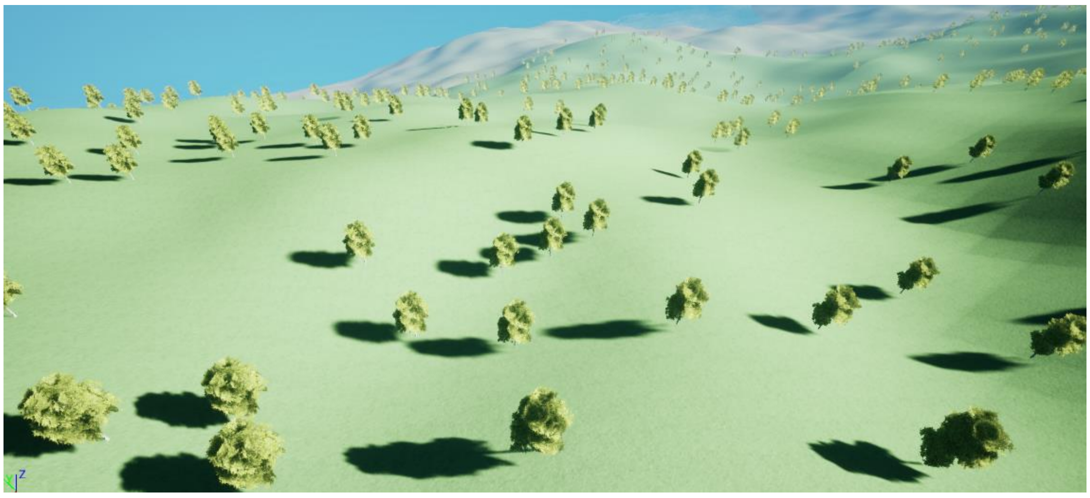


Figure 7 – Trees Instanced on the project

All the Trees/Bushes/Rocks on the Project are using this technique and allow me to have more than 3000 rendering on the screen without affect the framerate of the project.

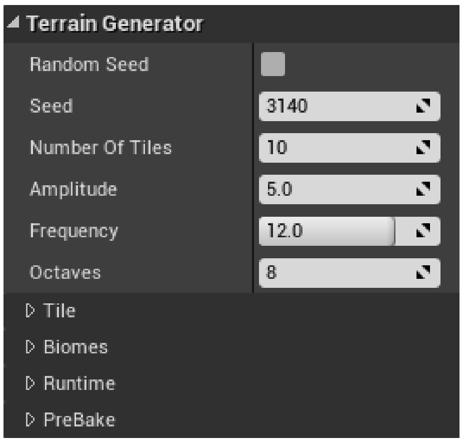
Design & Implementation

Overview

The Implementation has been made using a SCRUM methodology to keep in mind the task that should be achieved and to make development dynamic.

All the project was focused on made a unique tool/object with all the customisable settings to create the procedural terrain with a huge variety of options for personalize the entire world. In this way simplify the user interaction with the tool.

Terrain Generator



Terrain Generator

Random Seed	☐
Seed	3140
Number Of Tiles	10
Amplitude	5.0
Frequency	12.0
Octaves	8

▷ Tile
▷ Biomes
▷ Runtime
▷ PreBake

Figure 8 – Terrain Generator Basic

The first part of the object has de basic options for Perlin noise and the terrain. With the option of change the seed each time is generated the terrain or the option to specify the permanent value, with the number of tiles you want to the terrain with the minimum value of one.

Also contain the Amplitude, the Frequency and the Octaves which are common variables for this type of algorithms.

Amplitude

The Amplitude is a value for multiplier the noise number and vary the height of the map. More amplitude makes higher the values and less make the value lower or equal to the default Perlin noise.

Frequency

Noise can be generated at any frequency. It's sometimes useful to think of wavelength, which is the inverse of frequency. Doubling the frequency makes everything half the size. Doubling the wavelength makes everything twice the size. The wavelength is a distance, measured in pixels or tiles or meters or whatever you use for your maps.

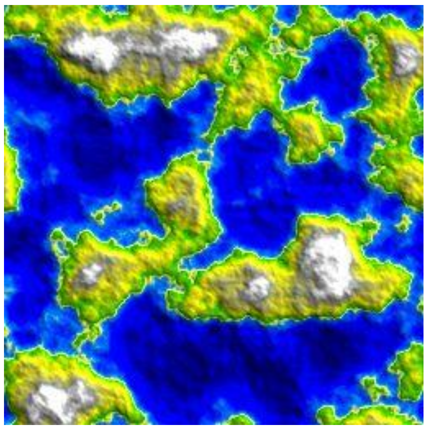


Figure 9 – Frequency value lower.

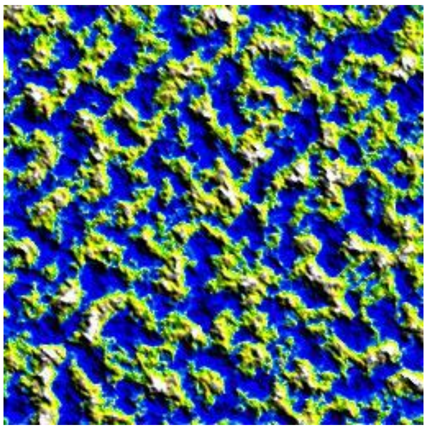


Figure 10 – Frequency value high.

Octaves

The number of octaves control the amount of detail of Perlin noise. Adding more octaves increases the detail of Perlin noise, with the added drawback of increasing the calculation time.

The amount of detail increases when the number of octaves increases. Beyond a certain number of octaves (determined by the resolution of the terrain height map), the frequency of the Perlin noise is greater than the resolution of the height map and you are doing more work than needed, for no effect.

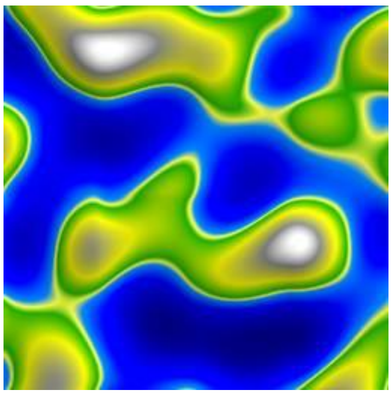


Figure 11 – Noise with Octaves to 1.

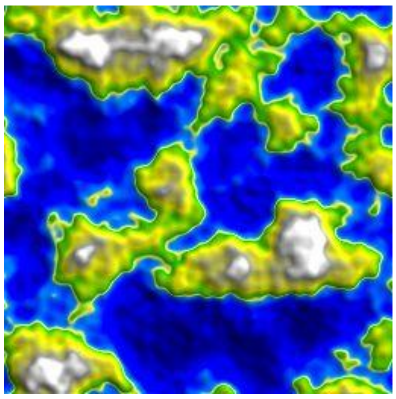
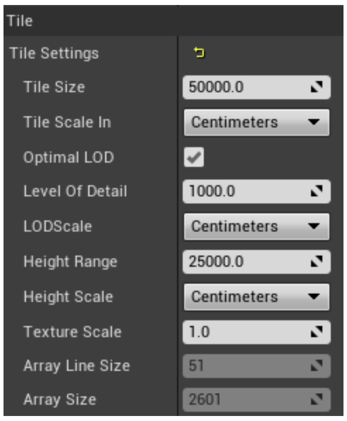


Figure 12 – Noise with Octaves to 6.

Tile



Tile	
Tile Settings	
Tile Size	50000.0
Tile Scale In	Centimeters
Optimal LOD	✓
Level Of Detail	1000.0
LODScale	Centimeters
Height Range	25000.0
Height Scale	Centimeters
Texture Scale	1.0
Array Line Size	51
Array Size	2601

Figure 13 – Tile Settings.

The Tile Settings is a struct with variety of variables with the only purpose of customize the Tile.
The Size of the tile allow us to choose the centimetres, meters or kilometres of each terrain tile, selecting the value of detail for the terrain with the level of detail option.
The Maximum height for the terrain also in centimetres, meters and kilometres and the texture scale of the material.

Biomes & Water

The **Biome tab** is the most extensive of all and the one that has more options with the only purpose of customize the terrain.

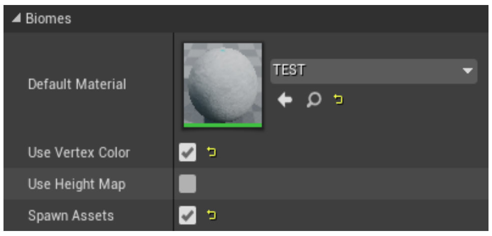

Biomes
Default Material: TEST
Use Vertex Color: ✓
Use Height Map: ☐
Spawn Assets: ✓

Figure 14 – Biomes tab & Water tab

The default material is the material for the entire Terrain that's contain the different textures the user wants.

The Use Vertex Colour and Use Height Map are additional options where the first one paint the terrain with the colours selected in each biome and the second one paint the entire terrain with a degraded black (lower elevation) to white (higher elevation).

The Boolean Spawn assets just activate or deactivate the option to spawn and place the assets for each biome.

Biome List

The Biome has a list of biomes where each element has different options:

- The Biome Name.
- The minimum and maximum height for this biome.
- Asset with more options:
  - Probability to spawn this asset in each vertices of the terrain of that biome.
  - The asset has collision or not. (this option can affect the performance)
  - If want random rotation for all angles or not.
  - If want random scale
  - The maximum scale value for each direction.
  - The mesh of the asset
- A list of colours in case you select the Use Vertex Colour
- A list of different materials for this terrain.

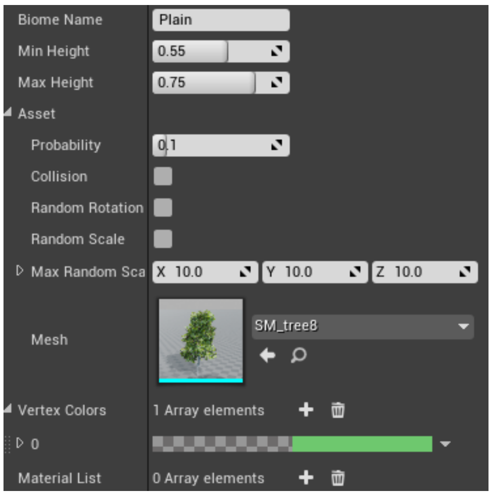
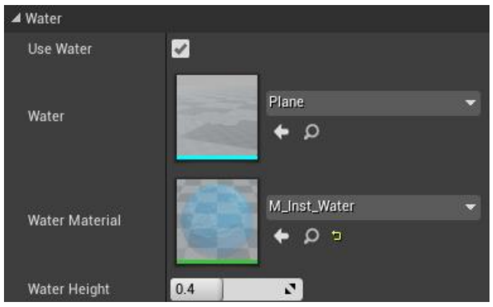

Biome Name: Plain
Min Height: 0.55
Max Height: 0.75
Asset
Probability: 0.1
Collision: ☐
Random Rotation: ☐
Random Scale: ☐
Max Random Sca: X 10.0 Y 10.0 Z 10.0
Mesh: SM_tree8
Vertex Colors: 1 Array elements
0
Material List: 0 Array elements

Figure 14 - Biomes tab & Water tab

Water

The **Water tab** has basic options.
The mesh the user want use for the water with the material of the water and the height of where the water is.

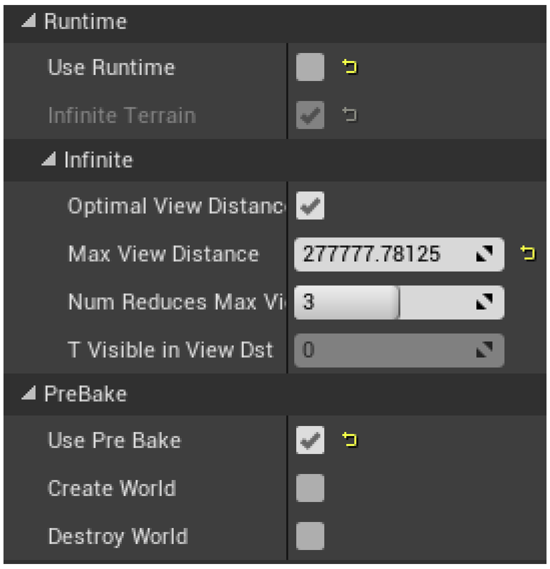


Water	
Use Water	✓
Water	Plane
Water Material	M_Inst_Water
Water Height	0.4

Figure 14 - Biomes tab & Water tab

Runtime & Prebake

This part of the object/tool just allow us to choose what type of terrain generation we need and the settings for this purpose.
The Pre-bake option just create all the terrain specified on the object outside the runtime, that's allow us the option to customize by hand the terrain adding more assets, the map border or whatever the user wants.
Instead the Runtime option works only in the "engine runtime" is on play, made the same as Prebake but with the exception of the Infinite terrain with different variables for customizable the max view distance of the terrain, the distance for the assets instanced but without the possibility of customize the map by hand.



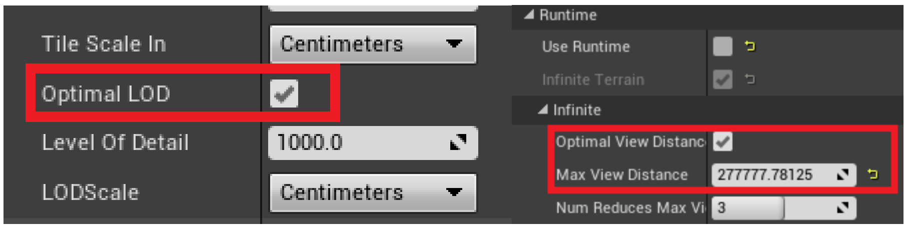
Runtime	
Use Runtime	
Infinite Terrain	✓
Infinite	
Optimal View Distanc	✓
Max View Distance	277777.78125
Num Reduces Max Vi	3
T Visible in View Dst	0
PreBake	
Use Pre Bake	✓
Create World	
Destroy World	

Figure 15 – Runtime and Pre-bake tab.

Optimization

Tiling

Since the terrain generation is subject to the variables that the user enters, this directly affects the yield of the terrain generation. To solve this, I offer the user options that calculate the optimal values for the number of vertices using the values contained in the Tile Settings.



Tile Scale In	Centimeters
Optimal LOD	✓
Level Of Detail	1000.0
LODScale	Centimeters

Runtime

Use Runtime	
Infinite Terrain	✓

Infinite

Optimal View Distanc	✓
Max View Distance	277777.78125
Num Reduces Max Vi	3

Figure 16 – Optimal Options

Multithreading

It was decided to implement multithreading to manage the project execution processes and, in this way, improve performance during the execution.

Due to the large amount of creation and modification of terrain meshes and in addition to the relevant calculations for the terrain generation and the distribution of assets, it was also decided to use promises and futures, which is a multi-thread management and use technique. It allows to function asynchronously.

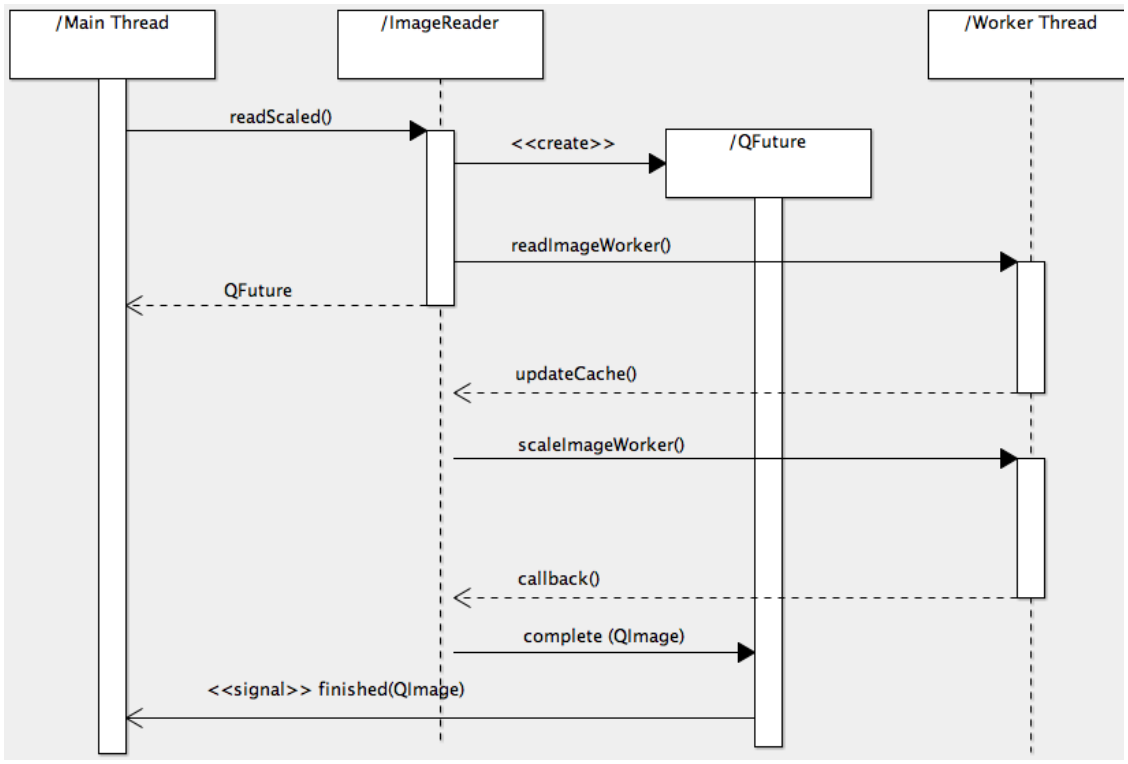


Figure 17 – Promises and Futures example

Instanced Mesh

Instanced Mesh enables us to increase the number of objects on the screen by drastically reducing the number of draw calls.

The image [Figure 17] shows the different variations of time it takes to draw the meshes in different ways.

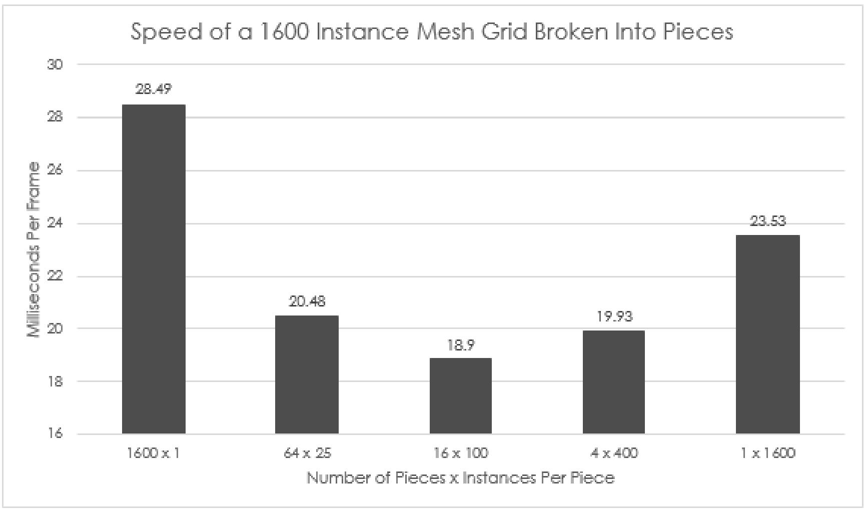


Figure 18 – Mesh Instancing Comparison

- The first 1600 x 1 graph indicates the time it takes to draw 1600 objects with 1 mesh each, which would be to draw an object.

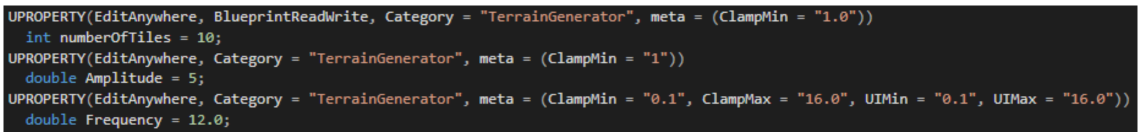
- The second graphic 64 x 25 already shows the performance of using Instanced Meshes, allowing to have 64 objects which group 25 meshes each.
  The third graph of 16 x 100 indicates that it is of the 5 examples that the best performance offers to have 16 objects which contain 100 instant meshes each.

- The last graphs show an increase in time compared to the previous two because they are no longer the groups of optimal instances but still offer better performance with respect to the first graph that does not use instant meshes.

Testing

Testing of the product was done heavily during the implementation process of the project. The testing was based around amount of meshes that could be built in the scene, the number of vertices for a runtime generation, the frame rates that the product ran at, the types of terrain generated by the noise and the overall quality of the meshes and the amount of assets for the terrain environment.

With the results obtained from all the testing, many of the variables that the user can modify within the optimal numbers have been limited, so that the user is aware of the limits so that his final product works at the maximum possible performance.



```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "TerrainGenerator", meta = (ClampMin = "1.0"))
  int numberOfTiles = 10;
UPROPERTY(EditAnywhere, Category = "TerrainGenerator", meta = (ClampMin = "1"))
  double Amplitude = 5;
UPROPERTY(EditAnywhere, Category = "TerrainGenerator", meta = (ClampMin = "0.1", ClampMax = "16.0", UIMin = "0.1", UIMax = "16.0"))
  double Frequency = 12.0;
```

Figure 19 - Clamped variables for the user

Also, given the result of spawning objects and the lower framerate I decide to implement the Instanced Static Mesh because of the numbers of assets placed on the terrain and the number of draw calls.

In this way also allows us to get to be able to have more than 3000 objects on screen with a very low cost.

Conclusion

The goal of the project was to study methods for procedurally terrain generation, plan the development phase and then actually implement my product and improve with customizable options.

Although the project has developed superbly and I feel really satisfied with the result product, I would have liked improve the optimisation and efficiency in how the terrain is generated making more testing because when you make a terrain more polished and realistic with the number of vertices the framerate is reduced and I would like know more ways to solve that but without more time is hard to do it, probably in my free time I will improve the project in this aspect.

Also, I am really interested on add more customizable options for the terrain and make an upgrade to the Biomes System to achieve to get the option to customize and create more biomes at the same height than others and with more asset placement variety for each one.

And even if it has nothing to do with the main idea of the project, I would like to add effects for the terrain when this appear or disappear at vision range, to give it a much more polished effect.

Anyway, I am really happy with the final product and for having managed to do it in Unreal Engine 4 considering that I had never gotten myself alone involved in such a complex project.

Bibliography

PCG Wiki. (2014). Algorithms for Procedural Content Generation. Available at: http://pcg.wikidot.com/category-pcg-algorithms (Accessed: 23 January 2019)

Moss, R. (2016). 7 uses of procedural generation that all developers should study. Available at: http://www.gamasutra.com/view/news/262869/7_uses_of_procedural_generation_that_all_developers_should_study.php (Accessed: 23 January 2019)

Grendel Games (2018) Procedural Generation. Available at: https://grendelgames.com/procedural-generation/?lang=en (Accessed: 23 January 2019)

Porter, B. (2017) Dynamic vs Static Procedural Generation. Available at: https://medium.com/@eigenbom/dynamic-vs-static-procedural-generation-ed3e7a7a68a3 (Accessed: 23 January 2019)

Green, D. (2016) Procedural Content Generation for C++ Game Development. Ed.Packt

Shaker,N. Togelius, J. Nelson, M. (2016). Procedural Content Generation in Games. Ed. Springer

References

Mojang. (2009) Minecraft. Available at: https://minecraft.net/ (Accessed: 30 April 2019).

Hello Games (2016) Available at: http://www.no-mans-sky.com/ (Accessed: 30 April 2019)

Gearbox Software (1999) Available at: http://www.gearboxsoftware.com/ (Accessed: 30 April 2019)

Runtime Mesh Component (2016), Available at: https://github.com/Koderz/RuntimeMeshComponent (Accessed: 30 April 2019)

Figures